Qorivva MPC5675K Microcontroller Reference Manual

Devices Supported:

MPC5673K
MPC5674K
MPC5675K

32-bit Qorivva dual-core MCU built on Power Architecture® technology, suitable for ISO26262 ASIL-D chassis and safety applications

MPC5675KRM
Rev 10
Nov 2013

Chapter 1
Introduction

Chapter 2
Memory Map

Chapter 3
Signal Description

Chapter 4
Functional Safety

Chapter 5
Boot and Operating Modes

Chapter 6
e200z7 Core

Chapter 7
General-Purpose Static RAM (SRAM)

Chapter 8
Flash Memory

Chapter 9
Multi-Layer AHB Crossbar Switch (XBAR)

Chapter 10
Peripheral Bridge (PBRIDGE)

Chapter 11
Analog-to-Digital Converter (ADC)

Chapter 12
Boot Assist Module (BAM)

Chapter 13
Clock Architecture

Chapter 14
Clock Generation Module (MC_CGM)

Chapter 15
Clock Monitor Unit (CMU)

Chapter 16
Cyclic Redundancy Checker (CRC) Unit

Chapter 17
Coherency Unit (CU)

Chapter 18
Cross-Triggering Unit (CTU)

Chapter 19
Enhanced Direct Memory Access (eDMA)

Chapter 20
eDMA Channel Mux (DMACHMUX)

Chapter 21
Deserial Serial Peripheral Interface (DSPI)

Chapter 22
External Bus Interface (EBI)

Chapter 23
Error Correction Status Module (ECSM)

Chapter 24
Enhanced Motor Control Timer (eTimer)

Chapter 25
Fault Collection and Control Unit (FCCU)

Chapter 26
Fast Ethernet Controller (FEC)

Chapter 27
FlexCAN Module

Chapter 28
Motor Control Pulse Width Modulator Module (FlexPWM)

Chapter 29
FlexRay Communication Controller (FLEXRAY)

Chapter 30
Frequency-Modulated Phase-Locked Loop (FMPLL)

Chapter 31
Inter-Integrated Circuit Bus Controller Module (I^2C)

Chapter 32
Interrupt Controller (INTC)

Chapter 33
JTAG Controller (JTAGC)

Chapter 34
LIN Controller (LINFlexD)

Chapter 35
Mode Entry Module (MC_ME)

Chapter 36
Multi-Port DDR DRAM Controller (MDDRC)

Chapter 37
Multi-Port DRAM Controller Priority Manager (PRIOMAN)

Chapter 38
Memory Protection Unit (MPU)

Chapter 39
Nexus Development Interface (NDI)

Chapter 40
Oscillators

Chapter 41
Parallel Digital Interface (PDI)

Chapter 42
Periodic Interrupt Timer (PIT)

Chapter 43
Power Control Unit (MC_PCU)

Chapter 44
Power Management Controller (PMC)

Chapter 45
Register Protection (REG_PROT)

Chapter 46
Reset Generation Module (MC_RGM)

Chapter 47
Semaphore Unit (SEMA4)

Chapter 48
System Integration Unit Lite (SIUL)

Chapter 49
System Status and Configuration Module (SSCM)

Chapter 50
Self-Test Control Unit (STCU)

Chapter 51
System Timer Module (STM)

Chapter 52
Software Watchdog Timer (SWT)

Chapter 53
Temperature Sensor (TSENS)

Chapter 54
Wakeup Unit (WKPU)

Appendix A
Revision History1727

Preface

Overview

The primary objective of this document is to define the functionality of the MPC5675K microcontroller for use by software and hardware developers. The MPC5675K is built on Power Architecture® technology and integrates technologies that are important for today's electric power steering, chassis, advanced driver assist, and safety applications, which require a high safety integrity level.

The information in this book is subject to change without notice, as described in the disclaimer to this document. As with any technical documentation, it is the reader's responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the Freescale web site at http://www.freescale.com/.

Audience

This manual is intended for hardware developers, system software developers, and applications programmers who want to develop products with the MPC5675K microcontroller. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

Chapter organization and device-specific information

This document includes chapters that describe:

- The microcontroller as a whole
- The functionality of the individual modules on the microcontroller

When the microcontroller is specified as "MPC5675K," the reader is instructed to apply this information to all of the microcontrollers specified on the front cover of this manual, unless individual device-specific details are provided in that chapter.

Information about different device versions ("cuts")

The MPC5675K device is available in two silicon versions, or "cuts". These are referred to as "cut1" and "cut2" throughout this document. Functional differences between the two cuts are clearly identified with the labels "cut1" and "cut2."

Document Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.
reserved	When a bit or address is reserved, it should not be written. If read, its value is not guaranteed. Reading or writing to reserved bits or addresses may cause unexpected results.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles in text are set in italics.
0x	Prefix to denote hexadecimal number
0b	Prefix to denote binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields, or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
halfword	A 16-bit data unit[1]
word	A 32-bit data unit
doubleword	A 64-bit data unit
x	In some contexts, such as signal encodings, x (without italics) indicates a “don’t care” condition.
x	With italics, used to express an undefined alphanumeric value (for example, a variable in an equation); or a variable alphabetic character in a bit, register, or module name (for example, DSPI_*x* could refer to DSPI_A or DSPI_B).
n	Used to express an undefined numerical value; or a variable numeric character in a bit, register, or module name (for example, EIF*n* could refer to EIF1 or EIF0).
~	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Field concatenation operator
$\overline{\text{OVERBAR}}$	An overbar indicates that a signal is active-low.

1. The only exceptions to this appear in the discussion of serial communication modules that support variable-length data transmission units. To simplify the discussion these units are referred to as words regardless of length.

Register Figure Conventions

This document uses the following conventions for the register reset values in register figures:

—	Bit value is undefined at reset.
U	Bit value is unchanged by reset. Previous value preserved during reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

The following descriptions are used in register bit field description tables:

R 0 W	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 0.
R 1 W	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 1.
R FIELDNAME W	Indicates a read/write bit in a memory-mapped register.
R FIELDNAME W	Indicates a read-only bit field in a memory-mapped register.
R W FIELDNAME	Indicates a write-only bit field in a memory-mapped register.
R FIELDNAME W w1c	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
R 0 W FIELDNAME	Indicates a self-clearing bit.

Acronyms and abbreviated terms

The following table lists some acronyms and abbreviations used in this document.

Term	Meaning
GPIO	General-purpose I/O
IEEE	Institute for Electrical and Electronics Engineers
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group
Mux	Multiplex
Rx	Receive

Term	Meaning (continued)
RTL	Register transfer language
Tx	Transmit
UART	Universal asynchronous/synchronous receiver transmitter

References

In addition to this reference manual, the following documents provide additional information on the operation of the MPC5675K:

- IEEE-ISTO 5001-2003 Standard for a Global Embedded Processor Interface (Nexus)
- IEEE 1149.1-2001 standard — IEEE Standard Test Access Port and Boundary-Scan Architecture
- Power Architecture Book E V1.0 (Book E: Enhanced PowerPC Architecture Version 1.0 (BOOK_EUM))
- e200z760n3 Power Architecture Core Reference Manual (e200z760RM)

Chapter 1
Introduction

1.1 The MPC5675K microcontroller

The Qorivva MPC5675K microcontroller is a 32-bit embedded controller based on the Power Architecture® and designed for:

- Advanced driver assistance systems with radar, CMOS imaging, LIDAR, and ultrasonic sensors
- Multiple 3-phase motor control applications as in hybrid electric vehicles (HEV) in automotive and high temperature industrial applications
- Safety applications that require a high safety integrity level

The MPC5675K chassis family is a new concept family developed around the central requirement for functional safety in the chassis space. The MPC5675K is a SafeAssure solution.

All MPC5675K microcontrollers are built around a dual-core safety platform with an innovative safety concept capable of being used in applications targeting ISO26262 ASILD and IEC61508 SIL3 integrity levels. To minimize additional software and module level features to reach this target, on-chip redundancy is provided for the critical components of the microcontroller (CPU core, DMA controller, interrupt controller, crossbar bus system, memory protection unit, flash memory and RAM controllers, peripheral bus bridge, system timers, and software watchdog timer). Lock-step Redundancy Checking Units are implemented at each output of this Sphere of Replication (SoR). Error correcting code memory (ECC) is available for on-chip RAM and flash memories. A programmable fault collection and control unit monitors the integrity status of the microcontroller and provides flexible safe state control.

The host processor core of the MPC5675K is a CPU from the e200z7d family of compatible Power Architecture® cores. The microcontroller’s 10-stage pipeline dual issue core is highly efficient, allowing high performance with minimum power dissipation.

1.2 MPC5675K microcontroller comparison

Table 1-1. MPC5675K family device comparison

Features		MPC5673K	MPC5674K	MPC5675K
CPU	Type	2 × e200z7d (SoR[1]) in lock-step or decoupled operation		
	Architecture	Harvard		
	Execution speed	0–150 MHz (+2% FM)	0–180 MHz (+2% FM)	0–180 MHz (+2% FM)
	Nominal platform frequency (in 1:1, 1:2, and 1:3 modes)	0–75 MHz (+2% FM)	0–90 MHz (+2% FM)	0–90 MHz (+2% FM)
	MMU	64 entries (SoR)		
	Instruction set PPC	Yes		
	Instruction set VLE	Yes		
	Instruction cache	16 KB, 4-way with EDC (SoR)		
	Data cache	16 KB, 4-way with Parity (SoR)		
	MPU	Yes (SoR)		
Buses	Core bus	32-bit address, 64-bit data		
	Internal periphery bus	32-bit address, 32-bit data		
XBAR	Master × slave ports	Yes (SoR)		
Memory	Static RAM (SRAM)	256 KB (ECC)	384 KB (ECC)	512 KB (ECC)
	Code flash memory	1 MB[2] (ECC)	1.5 MB[2] (ECC)	2 MB[2] (ECC)
	Data flash memory	64 KB[2] (ECC)		
Modules	Analog-to-Digital Converter (ADC)	257 pin pkg: 4 × 12 bit (22 external channels) 473 pin pkg: 4 × 12 bit (up to 34 external channels)		
	CRC unit	2 (3 contexts each)		
	Cross Triggering Unit (CTU)	2 modules		
	Deserial Serial Peripheral Interface (DSPI)	2 modules (3 chip selects)[3]	3 modules[4]	
	Digital I/Os	≥ 16		
	DRAM Controller (DRAMC)	No	Yes[5]	
	Enhanced Direct Memory Access (eDMA)	2 modules, 32 channels each		
	eTimer	3 modules, 6 channels each		

Table 1-1. MPC5675K family device comparison (continued)

Features		MPC5673K	MPC5674K	MPC5675K
Modules (cont.)	External Bus Interface (EBI)	1 module[5] 16-bit Data + Address or 32-bit Data with Address bus muxed[8]		
	Fast Ethernet Controller (FEC)	1 module		
	Fault Collection and Control Unit (FCCU)	1 module		
	FlexCAN	4 modules (32 message buffers each)		
	FlexPWM	3 modules (each 4 × 3 channels)		
	FlexRay	Optional		Yes
	I^2C	2 modules[6]	3 modules	
	Interrupt Controller (INTC)	Yes (SoR)		
	LINFlex	3 modules[7]	4 modules	
	Parallel Data Interface (PDI)	1 module[8]		
	Periodic Interrupt Timer (PIT)	1 module, 4 channels		
	Software Watchdog Timer (SWT)	Yes (SoR)		
	System Timer Module (STM)	Yes (SoR)		
	Temperature sensor	1 module		
	Wakeup Unit (WKPU)	Yes		
	Crossbar switch (XBAR)	3 modules, 2 are user-configurable		
Clocking	Clock monitor unit (CMU)	3 modules		
	Frequency-modulated phase-locked loop (FMPLL)	2 modules (system and auxiliary)		
	IRCOSC – 16 MHz	1		
	XOSC 4–40 MHz	1		
Supply	Power management unit (PMU)	Yes		
	1.2 V low-voltage detector (LVD12)	1		
	1.2 V high-voltage detector (HVD12)	1		
	2.7 V low-voltage detector (LVD27)	4		

Table 1-1. MPC5675K family device comparison (continued)

Features		MPC5673K	MPC5674K	MPC5675K
Debug	Nexus	Class 3+ (for cores and SRAM ports)		
Packages	MAPBGA	257 pins 473 pins		
Temperature	Ambient	See the T_A recommended operating condition in the device data sheet		

1 Sphere of Replication.

2 Does not include Test or Shadow Flash memory space. Regardless of LSM or DPM operating modes, all MPC5675K part numbers have contiguous flash memory spaces starting at the same start address.

3 DSPI_0 and DSPI_1.

4 DSPI_0 has 8 chip selects; DSPI_1 and DSPI_2 have 4 chip selects each.

5 Available only on 473-pin package.

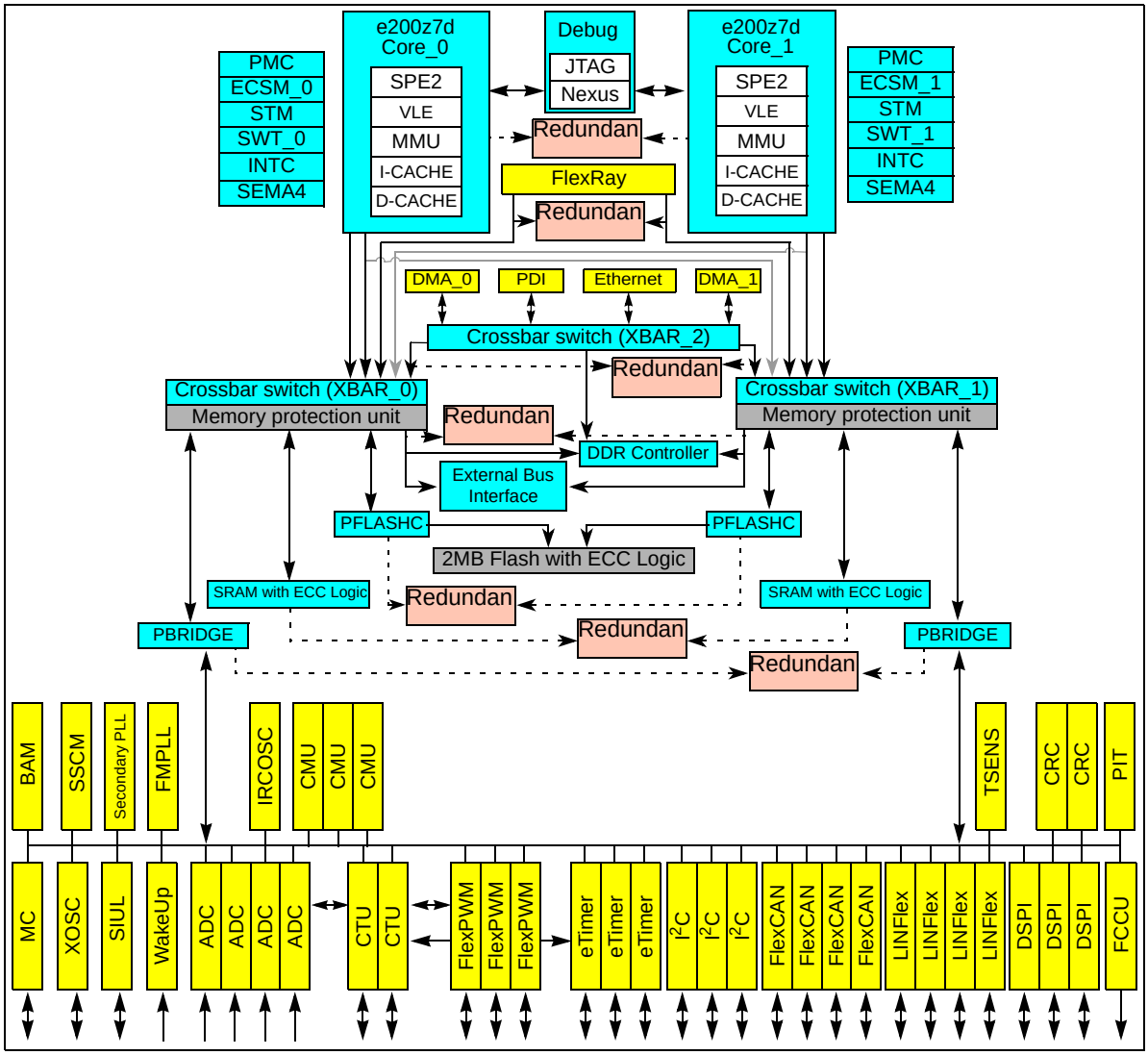
6 Any two of the three I2C can be chosen.

7 LinFlex_0, LinFlex_1, and LinFlex_2.

8 DDR available only on 473 package. Other modules available as follows:
EBI or DDR on 473 package
EBI + PDI on 473 package
DDR + PDI on 473 package
PDI only on 257 package

1.2.1 Block diagram

Figure 1-1 shows a top-level block diagram of the MPC5675K.



ADC – Analog-to-digital converter
BAM – Boot assist module
CMU – Clock monitoring unit
CRC – Cyclic redundancy check unit
CTU – Cross triggering unit
DSPI – Deserial serial peripheral interface
EBI – External bus interface
ECC – Error correction code
ECSM – Error correction status module
eDMA – Enhanced direct memory access controller
FCCU – Fault collection and control unit
FEC – Fast Ethernet controller
FlexCAN – Controller area network controller
FlexPWM – Pulse width modulator module
FMPLL – Frequency-modulated phase-locked loop
I2C – Inter-integrated circuit controller
INTC – Interrupt controller
IRCOSC – Internal RC oscillator
JTAG – Joint Test Action Group interface
MC – Mode entry, clock, reset, and power modules
mDDR – Mobile double data rate dynamic RAM
PBRIDGE – Peripheral bridge
PDI – Parallel data interface
PIT – Periodic interrupt timer
PMC – Power management controller
RC – Redundancy checker
RTC – Real time clock
SEMA4 – Semaphore unit
SIUL – System integration unit Lite
SSCM – System status and configuration module
STM – System timer module
SWT – Software watchdog timer
TSENS – Temperature sensor
XOSC – Crystal oscillator

Figure 1-1. MPC5675K block diagram

1.3 Feature summary

- High-performance e200z7d dual core
  - 32-bit Power Architecture® technology CPU
  - Up to 180 MHz core frequency
  - Dual-issue core
  - Variable length encoding (VLE)
  - Memory management unit (MMU) with 64 entries
  - 16 KB instruction cache and 16 KB data cache
- Memory available
  - Up to 2 MB code flash memory with ECC
  - 64 KB data flash memory with ECC
  - Up to 512 KB on-chip SRAM with ECC
- SIL3/ASILD innovative safety concept: LockStep mode and fail-safe protection
  - Sphere of replication (SoR) for key components
  - Redundancy checking units on outputs of the SoR connected to FCCU
  - Fault collection and control unit (FCCU)
  - Boot-time built-in self-test for memory (MBIST) and logic (LBIST) triggered by hardware
  - Boot-time built-in self-test for ADC and flash memory
  - Replicated safety-enhanced watchdog timer
  - Silicon substrate (die) temperature sensor
  - Non-maskable interrupt (NMI)
  - 16-region memory protection unit (MPU)
  - Clock monitoring units (CMU)
  - Power management unit (PMU)
  - Cyclic redundancy check (CRC) units
- Decoupled Parallel mode for high-performance use of replicated cores
- Nexus Class 3+ interface
- Interrupts
  - Replicated 16-priority interrupt controller
- GPIOs individually programmable as input, output, or special function
- 3 general-purpose eTimer units (6 channels each)
- 3 FlexPWM units with four 16-bit channels per module
- Communications interfaces
  - 4 LINFlex modules
  - 3 DSPI modules with automatic chip select generation
  - 4 FlexCAN interfaces (2.0B Active) with 32 message objects
  - FlexRay module (V2.1) with dual channel, up to 128 message objects and up to 10 Mbit/s

  - Fast Ethernet Controller (FEC)
  - 3 I^2C modules
- Four 12-bit analog-to-digital converters (ADCs)
  - 22 input channels
  - Programmable cross triggering unit (CTU) to synchronize ADC conversion with timer and PWM
- External bus interface
- 16-bit external DDR memory controller
- Parallel digital interface (PDI)
- On-chip CAN/UART bootstrap loader
- Capable of operating on a single 3.3 V voltage supply
  - 3.3 V-only modules: I/O, oscillators, flash memory
  - 3.3 V or 5 V modules: ADCs, supply to internal VREG
  - 1.8–3.3 V supply range: DRAM/PDI
- Operating junction temperature range –40 to 150 °C

1.4 Feature details

1.4.1 High-performance e200z7d core processor

- Dual 32-bit Power Architecture® processor core
- Loose or tight core coupling
- Freescale Variable Length Encoding (VLE) enhancements for code size footprint reduction
- Thirty-two 64-bit general-purpose registers (GPRs)
- Memory management unit (MMU) with 64-entry fully-associative translation look-aside buffer (TLB)
- Branch processing unit
- Fully pipelined load/store unit
- 16 KB Instruction and 16 KB Data caches per core with line locking
  - Four way set associative
  - Two 32-bit fetches per clock
  - Eight-entry store buffer
  - Way locking
  - Supports tag and data cache parity
  - Supports EDC for instruction cache
- Vectored interrupt support

- Signal processing engine 2 (SPE2) auxiliary processing unit (APU) operating on 64-bit general purpose registers
- Floating point
  - IEEE® 754 compatible with software wrapper
  - Single precision in hardware; double precision with software library
  - Conversion instructions between single precision floating point and fixed point
- Long cycle time instructions (except for guarded loads) do not increase interrupt latency in the MPC5675K
- To reduce latency, long cycle time instructions are aborted upon interrupt requests
- Extensive system development support through Nexus debug module

1.4.2 Crossbar Switch (XBAR)

- 32-bit address bus, 64-bit data bus
- Simultaneous accesses from different masters to different slaves (there is no clock penalty when a parked master accesses a slave)

1.4.3 Memory Protection Unit (MPU)

Each master (eDMA, FlexRay, CPU) can be assigned different access rights to each region.

- 16-region MPU with concurrent checks against each master access
- 32-byte granularity for protected address region

1.4.4 Enhanced Direct Memory Access (eDMA) controller

- 32 channels support independent 8-, 16-, 32-bit single value or block transfers
- Supports variable-sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer

1.4.5 Interrupt Controller (INTC)

- 208 peripheral interrupt requests
- 8 software settable sources
- Unique 9-bit vector per interrupt source
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Priority elevation for shared resources

1.4.6 Frequency-Modulated Phase-Locked Loop (FMPLL)

Two FMPLLs are available on each device.

Each FMPLL allows the user to generate high speed system clocks starting from a minimum reference of 4 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor and output clock divider ratio are software configurable. The FMPLLs have the following major features:

- Input frequency: 4–40 MHz continuous range (limited by the crystal oscillator)
- Voltage controlled oscillator (VCO) range: 256–512 MHz
- Frequency modulation via software control to reduce and control emission peaks
  - Modulation depth ±2% if centered or 0% to –4% if downshifted via software control register
  - Modulation frequency: triangular modulation with 25 kHz nominal rate
- Option to switch modulation on and off via software interface
- Reduced frequency divider (RFD) for reduced frequency operation without re-lock
- 2 modes of operation
  - Normal PLL mode with crystal reference (default)
  - Normal PLL mode with external reference
- Lock monitor circuitry with lock status
- Loss-of-lock detection for reference and feedback clocks
- Self-clocked mode (SCM) operation
- Auxiliary FMPLL
  - Used for FlexRay due to precise symbol rate requirement by the protocol
  - Used for motor control periphery and connected IP (A/D digital interface CTU) to allow independent frequencies of operation for PWM and timers as well as jitter-free control
  - Option to enable/disable modulation to avoid protocol violation on jitter and/or potential unadjusted error in electric motor control loop
  - Allows running motor control periphery at different (precisely lower, equal, or higher, as required) frequency than the system to ensure higher resolution

1.4.7 External Bus Interface (EBI)

- Available on 473-pin devices
- Data and address options:
  - 16-bit data and address (non-muxed)
  - 32-bit data and address (bus-muxed)
- MPC5561 324 BGA compatibility mode: 16-bit data bus, 24-bit address bus is default ADDR[8:31], but configurable to 26-bit address bus
- Memory controller with support for various memory types
  - Non-burst and burst mode SDR flash and SRAM
  - Asynchronous/legacy flash and SRAM

- Configurable bus speed modes
- Support for 2 MB address space
- Chip select and write/byte enable options as presented in the pin-muxing table in the “Signal Description” chapter of the MPC5675K reference manual
- Configurable wait states (via chip selects)
- Optional automatic CLKOUT gating to save power and reduce EMI

1.4.8 On-chip flash memory

- Up to 2 MB code flash memory with ECC
- 64 KB data flash memory with ECC
- Censorship protection scheme to prevent flash content visibility
- Multiple block sizes to support features such as boot block, operating system block, and EEPROM emulation
- Read-while-write with multiple partitions
- Parallel programming mode to support rapid end-of-line programming
- Hardware programming state machine

1.4.9 Cache memory

- Harvard architecture cache
- 16 KB instruction / 16 KB data
- Four-way set-associative Harvard (instruction and data) 256-bit long cache
  - — Two 32-bit fetches per clock
  - — Eight-entry store buffer
  - — Way locking
  - — Supports tag and data cache parity
  - — Supports EDC for instruction cache

1.4.10 On-chip internal static RAM (SRAM)

- Up to 512 KB general-purpose SRAM
- ECC performs single-bit correction, double-bit error detection
  - — Address included in ECC checkbase

1.4.11 DRAM controller

The DRAM controller (available only on 473-pin devices) is a multi-port controller that monitors incoming requests on the three AHB slave ports and decides (at each rising clock edge) what command needs to be sent to the external DRAM.

The DRAM controller on this device supports the following types of memories:

- Mobile DDR (mDDR)
- DDR 1
- DDR 2 (optional)
- SDR

The controller has the following features:

- Optimized timing for 32-byte bursts and single read accesses on the AHB interface
- Optimized timing for 8-byte and 16-byte bursts on the DRAMC interface
- Supports priority elevation on the slave ports for single accesses
- 16-bit wide DRAM interface
- One chip select (CS)
- mDDR memory controller
  - 16-bit external interface
  - Address range up to 8 MB

1.4.12 Boot Assist Module (BAM)

- Enables booting via serial mode (FlexCAN, LINFlex)
- Handles static mode in case of an erroneous boot procedure
- Implemented in 8 KB ROM
- Supports Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM)

1.4.13 Parallel Data Interface (PDI)

- Support for external ADC and CMOS image sensors
- Parallel interface operation up to MCU system bus frequency
- Selectable data capture from rising or falling edge
- Receive FIFO with adjustable trigger thresholds
- Data width for 8, 10, 12, 14, and 16 bits
- Data Packing Unit to pack input data on 64-bit words — data packed on 8- or 16- bit boundary, depending on input data width
- Binary increasing channel select that allows as many as eight channels to be selected
- Frame synchronization through Vsync, Hsync, PIXCLK

1.4.14 Deserial Serial Peripheral Interface (DSPI) modules

- Three serial peripheral interfaces
  - Full duplex communication ports with interrupt and eDMA request support
  - Support for all functional modes from QSPI submodule of QSMCM (MPC5xx family)
  - Support for queues in RAM
  - Six chip selects, expandable to 64 with external demultiplexers

  - Programmable frame size, baud rate, clock delay, and clock phase on a per-frame basis
  - Modified SPI mode for interfacing to peripherals with longer setup time requirements
- Support for up to 60 Mbit/s in slave only Rx mode

1.4.15 Serial Communication Interface Module (LINFlex)

The LINFlex on this device features the following:

- Supports LIN Master mode, LIN Slave mode, and UART mode
- LIN state machine compliant to LIN1.3, 2.0, and 2.1 specifications
- Manages LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store as many as 8 data bytes
  - Supports messages as long as 64 bytes
  - Detection and flagging of LIN errors (Sync field, delimiter, ID parity, bit framing, checksum and timeout errors)
  - Classic or extended checksum calculation
  - Configurable break duration of up to 36-bit times
  - Programmable baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features (loop back, LIN bus stuck dominant detection)
  - Interrupt-driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-byte receive, 4-byte transmit
  - Configurable word length (8-bit, 9-bit, or 16-bit words)
  - Configurable parity scheme: none, odd, even, always 0
  - Speed as fast as 2 Mbit/s
  - Error detection and flagging (parity, noise, and framing errors)
  - Interrupt-driven operation with four interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud-rate modulus counter and 16-bit fractional
  - Two receiver wake-up methods
- Support for DMA-enabled transfers

1.4.16 FlexCAN

- Thirty-two message buffers each
- Full implementation of the CAN protocol specification, Version 2.0B
- Programmable acceptance filters
- Individual Rx filtering per message buffer
- Short latency time for high priority transmit messages
- Arbitration scheme according to message ID or message buffer number
- Listen-only mode capabilities
- Programmable clock source: system clock or oscillator clock
- Reception queue possible by setting more than one Rx message buffer with the same ID
- Backwards compatible with previous FlexCAN modules
- Safety CAN features on 1 CAN module as implemented on MPC5604P

1.4.17 Dual-channel FlexRay controller

- Full implementation of FlexRay Protocol Specification 2.1
- Sixty-four configurable message buffers can be handled
- Message buffers configurable as Tx, Rx, or RxFIFO
- Message buffer size configurable
- Message filtering for all message buffers based on FrameID, cycle count, and message ID
- Programmable acceptance filters for RxFIFO message buffers
- Dual channel, each at up to 10 Mbit/s data rate

1.4.18 Periodic Interrupt Timer (PIT)

The PIT module implements the features below:

- Four general-purpose interrupt timers
- 32-bit counter resolution
- Clocked by system clock frequency
- 32-bit counter for real time interrupt, clocked from main external oscillator
- Can be used for software tick or DMA trigger operation

1.4.19 System Timer Module (STM)

The STM implements the features below:

- Replicated periphery to provide safety measures respective to high safety integrity levels (for example, SIL 3, ASIL D)
- Up-counter with four output compare registers

- OS task protection and hardware tick implementation as per current state-of-the-art AUTOSAR requirement

1.4.20 Motor control (MOTC) peripherals

The peripherals in this section can be used for general-purpose applications, but are specifically designed for motor control (MOTC) applications.

1.4.20.1 FlexPWM

The pulse width modulator module (FlexPWM) contains three PWM channels, each of which is configured to control a single half-bridge power stage. There may also be one or more fault channels.

This PWM is capable of controlling most motor types: AC induction motors (ACIM), permanent magnet AC motors (PMAC), both brushless (BLDC) and brush DC motors (BDC), switched (SRM) and variable reluctance motors (VRM), and stepper motors.

A FlexPWM module implements the following features:

- 16 bits of resolution for center, edge aligned, and asymmetrical PWMs
- Maximum operating frequency lower than or equal to platform frequency
- Clock source not modulated and independent from system clock (generated via auxiliary PLL)
- Fine granularity control for enhanced resolution of the PWM period
- PWM outputs can operate as complementary pairs or independent channels
- Ability to accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM is supported
- Double-buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half-cycle reload capability
- Multiple ADC trigger events can be generated per PWM cycle via hardware
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software control for each PWM output
- All outputs can be forced to a value simultaneously
- PWMX pin can optionally output a third signal from each channel
- Channels not used for PWM generation can be used for:
  - buffered output compare functions
  - input capture functions

- Enhanced dual-edge capture functionality
- Option to supply the source for each complementary PWM signal pair from any of the following:
  - External digital pin
  - Internal timer channel
  - External ADC input, taking into account values set in ADC high and low limit registers
- Supports safety measures using DMA

1.4.20.2 Cross Triggering Unit (CTU)

The CTU provides automatic generation of ADC conversion requests on user-selected conditions without CPU load during the PWM period and with minimized CPU load for dynamic configuration.

The CTU implements the following features:

- Cross triggering between ADC, FlexPWM, eTimer, and external pins
- Double-buffered trigger generation unit with as many as eight independent triggers generated from external triggers
- Maximum operating frequency lower than or equal to platform
- Trigger generation unit configurable in sequential mode or in triggered mode
- Trigger delay unit to compensate the delay of external low-pass filter
- Double-buffered global trigger unit allowing eTimer synchronization and/or ADC command generation
- Double-buffered ADC command list pointers to minimize ADC trigger unit update
- Double-buffered ADC conversion command list with as many as twenty-four ADC commands
- Each trigger has the capability to generate consecutive commands
- ADC conversion command allows controlling ADC channel from each ADC, single or synchronous sampling, independent result queue selection
- Supports safety measures using DMA

1.4.20.3 Analog-To-Digital Converter (ADC)

- Four independent ADCs with 12-bit A/D resolution
- Common mode conversion range of 0–5 V or 0–3.3 V
- Twenty-two single-ended input channels
- Supports eight FIFO queues with fixed priority
- Queue modes with priority-based preemption; initiated by software command, internal, or external triggers
- DMA and interrupt request support

1.4.20.4 eTimer module

Three 16-bit general purpose up/down timer/counters per module are implemented with the following features:

- Ability to operate up to platform frequency
- Individual channel capability
  - Input capture trigger
  - Output compare
  - Double buffer (to capture rising edge and falling edge)
  - Separate prescaler for each counter
  - Selectable clock source
  - 0–100% pulse measurement
  - Rotation direction flag (quad decoder mode)
- Maximum count rate
  - Equals peripheral clock/2 for external event counting
  - Equals peripheral clock for internal clock counting
- Cascadeable counters
- Programmable count modulo
- Quadrature decode capabilities
- Counters can share available input pins
- Count once or repeatedly
- Preloadable counters
- Pins available as GPIO when timer functionality is not in use
- DMA support

1.4.21 Redundancy Control and Checker Unit (RCCU)

The RCCU checks all outputs of the sphere of replication (addresses, data, control signals). It has the following features:

- Duplicated module to enable high diagnostic coverage (check of checker)
- Replicated IP to be used as checkers on the PBRIDGE output, Flash Controller output, SRAM output, DMA Channel Mux inputs

1.4.22 Software Watchdog Timer (SWT)

This module implements the features below:

- Replicated periphery to provide safety measures respective to high safety integrity levels (for example, SIL 3, ASIL D)
- Fault-tolerant output
- Safe internal RC oscillator as reference clock
- Windowed watchdog
- Program flow control monitor with 16-bit pseudorandom key generation
- Provides measures to target high safety integrity levels (for example, SIL 3, ASIL D)

1.4.23 Fault Collection and Control Unit (FCCU)

The FCCU module has the following features:

- Redundant collection of hardware checker results
- Redundant collection of error information and latch of faults from critical modules on the device
- Collection of test results
- Configurable and graded fault control
  - Internal reactions (no internal reaction, NMI, reset, or safe mode)
  - External reaction (failure is reported to the outside world via configurable output pins)

1.4.24 System Integration Unit Lite (SIUL)

The SIUL controls MCU reset configuration, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, and the system reset operation. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU.

The SIUL provides the following features:

- Centralized pad control on a per-pin basis
  - Pin function selection
  - Configurable weak pullup/pulldown
  - Configurable slew rate control (slow/medium/fast)
  - Hysteresis on GPIO pins
  - Configurable automatic safe mode pad control
- Input filtering for external interrupts

1.4.25 Cyclic Redundancy Checker (CRC) unit

The CRC module is a configurable multiple data flow unit to compute CRC signatures on data written to an input register.

The CRC unit has the following features:

- Three sets of registers to allow three concurrent contexts with possibly different CRC computations, each with a selectable polynomial and seed
- Computes 16- or 32-bit wide CRC on the fly (single-cycle computation) and stores the result in an internal register
- Implements the following standard CRC polynomials:
  - $x^{16} + x^{12} + x^5 + 1$ [16-bit CRC-CCITT]
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ [32-bit CRC-ethernet(32)]

- Key engine to be coupled with communication periphery where CRC application is added to support implementation of safe communication protocol
- Offloads the core from cycle-consuming CRC and helps in checking the configuration signature for safe start-up or periodic procedures
- Connected as a peripheral on the internal peripheral bus
- Provides DMA support

1.4.26 Non-Maskable Interrupt (NMI)

The non-maskable interrupt with de-glitching filter is available to support high priority core exceptions.

1.4.27 System Status and Configuration Module (SSCM)

The SSCM on the MPC5675K features the following:

- System configuration and status
- Debug port status and debug port enable
- Multiple boot code starting locations out of reset through implementation of search for valid reset configuration halfword
- Sets up the MMU to allow user boot code to execute as either Classic Power Architecture Book E code (default) or as Freescale VLE code out of flash
- Supports serial bootloading of either Classic Power Architecture Book E code (default) or Freescale VLE code
- Detection of user boot code
- Automatic switch to serial boot mode if internal flash is blank or invalid

1.4.28 Nexus Development Interface (NDI)

- Per IEEE-ISTO 5001-2008
- Real-time development support for Power Architecture core through Nexus class 3 (some class 4 support)
- Nexus support to snoop system SRAM traffic
- Data trace of FlexRay accesses
- Read and write access
- Configured via the IEEE 1149.1 (JTAG) port
- High bandwidth mode for fast message transmission
- Reduced bandwidth mode for reduced pin usage

1.4.29 IEEE 1149.1 JTAG Controller (JTAGC)

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- JCOMP input that provides the ability to share the TAP —selectable modes of operation include JTAGC/debug or normal system operation

- 5-bit instruction register that supports IEEE 1149.1-2001 defined instructions
- 5-bit instruction register that supports additional public instructions
- Three test data registers:
  - Bypass register
  - Boundary scan register
  - Device identification register
- TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry

This page is intentionally left blank.

Chapter 2
Memory Map

2.1 Introduction

Table 2-1 shows the high-level memory organization of the MPC5675K microcontroller.

Table 2-1. MPC5675K system memory map overview

Start address	Size (KB)	Region
Flash memory		
0x0000_0000	[1]	Flash memory
EBI		
0x2000_0000	[1]	EBI
SRAM		
0x4000_0000	[1]	SRAM
DRAM		
0x6000_0000	[1]	DRAM
On-platform peripherals		
0x8000_0000	[1]	Peripherals on PBRIDGE
Not allocated		
0xA000_0000	[1]	Not allocated
Off-platform peripherals		
0xC000_0000	[1]	Peripherals on PBRIDGE
Off-platform peripherals		
0xE000_0000	[1]	Peripherals on PBRIDGE

[1] For size, please refer to the actual memory map in Table 2-2 below.

Table 2-2 shows the memory map for the MPC5675K. All addresses on the MPC5675K, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each region or module name. The table also identifies the associated peripheral control register (PCTL) number and the differences in the memory map between Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM).

Table 2-2. MPC5675K memory map

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
Flash memory				
0x0000_0000	16	—	—	CFM0 Code flash memory array Bank 0
0x0000_4000	16	—	—	CFM0 Code flash memory array Bank 0

Table 2-2. MPC5675K memory map (continued)

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
0x0000_8000	32	—	—	CFM0 Code flash memory array Bank 0
0x0001_0000	32	—	—	CFM0 Code flash memory array Bank 0
0x0001_8000	16	—	—	CFM0 Code flash memory array Bank 0
0x0001_C000	16	—	—	CFM0 Code flash memory array Bank 0
0x0002_0000	64	—	—	CFM0 Code flash memory array Bank 0
0x0003_0000	64	—	—	CFM0 Code flash memory array Bank 0
0x0004_0000	16	—	—	CFM1 Code flash memory array Bank 0
0x0004_4000	16	—	—	CFM1 Code flash memory array Bank 0
0x0004_8000	32	—	—	CFM1 Code flash memory array Bank 0
0x0005_0000	32	—	—	CFM1 Code flash memory array Bank 0
0x0005_8000	16	—	—	CFM1 Code flash memory array Bank 0
0x0005_C000	16	—	—	CFM1 Code flash memory array Bank 0
0x0006_0000	64	—	—	CFM1 Code flash memory array Bank 0
0x0007_0000	64	—	—	CFM1 Code flash memory array Bank 0
0x0008_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 0
0x000C_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 0
0x0010_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 1
0x0014_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 1
0x0018_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 1
0x001C_0000	256	—	—	CFM0/CFM1 Code flash memory array Bank 1
0x0020_0000	16	—	—	CFM0 Code flash memory array shadow sector
0x0020_4000	496	Reserved		
0x0028_0000	16	—	—	CFM1 Code flash memory array shadow sector
0x0028_4000	1520	Reserved		
0x0040_0000	16	—	—	CFM0 Code flash memory array test sector
0x0040_4000	4080	Reserved		
0x0080_0000	16	—	—	DFM0 Data flash memory array
0x0080_4000	16	—	—	DFM0 Data flash memory array
0x0080_8000	16	—	—	DFM0 Data flash memory array
0x0080_C000	16	—	—	DFM0 Data flash memory array
0x0081_0000	4032	Reserved		
0x00C0_0000	8	—	—	DFM0 Data flash memory array test sector

Table 2-2. MPC5675K memory map (continued)

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
0x00C0_2000	4088	Reserved		
Emulation mapping				
0x0100_0000	507904	—		Emulation
EBI				
0x2000_0000	524288	—		EBI
Static RAM (SRAM)				
0x4000_0000	512	—	LS	MPC5675K
	384	—	LS	MPC5674K
	256	—	LS	MPC5673K
	256	—	DP	MPC5675K
	192	—	DP	MPC5674K
	128	—	DP	MPC5673K
0x5000_0000	256	—	DP	MPC5675K
	192	—	DP	MPC5674K
	128	—	DP	MPC5673K
Dynamic RAM (DRAM)				
0x6000_0000	785408	—	—	DRAM
Peripherals				
0x8FF0_0000	16	—	DP	PBRIDGE_1
0x8FF0_4000[2]	16	—	DP	XBAR_1
0x8FF0_8000	32	Reserved		
0x8FF1_0000	16	—	DP	Memory Protection Unit (MPU_1)
0x8FF1_4000	64	Reserved		
0x8FF2_4000	16	—	DP	Semaphores (SEMA4_1)
0x8FF2_8000	64	Reserved		
0x8FF3_8000	16	—	DP	Software Watchdog (SWT_1)
0x8FF3_C000	16	—	DP	System Timer Module (STM_1)
0x8FF4_0000	16	—	DP	Error Correction Status Module (ECSM_1)
0x8FF4_4000	16	—	DP	Direct Memory Access Controller (DMA_1)
0x8FF4_8000	16	—	DP	Interrupt Controller (INTC_1)
0x8FF4_C000	850896	Reserved		
0xC3E4_0000	16	112	LS/DP	Parallel Digital Interface (PDI)

Table 2-2. MPC5675K memory map (continued)

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
0xC3E4_4000	48	Reserved		
0xC3E5_0000	16	116	LS/DP	Analog to Digital Converter 2 (ADC_2)
0xC3E5_4000	16	117	LS/DP	Analog to Digital Converter 3 (ADC_3)
0xC3E5_8000	16	Reserved		
0xC3E5_C000	16	119	LS/DP	Cross Trigger Unit 1 (CTU_1)
0xC3E6_0000	16	120	LS/DP	Cyclic Redundancy Check Unit 1 (CRCU_1)
0xC3E6_4000	64	Reserved		
0xC3E7_4000	16	125	LS/DP	FlexPWM_2
0xC3E7_8000	1072	Reserved		
0xC3F8_4000	16	65	LS/DP	External Bus Interface (EBI)
0xC3F8_8000	16	66	LS/DP	Code flash memory 0 Configuration (CFLASH0)
0xC3F8_C000	16	67	LS/DP	DFM0 Data flash memory 0 Configuration (DFLASH0)
0xC3F9_0000	16	68	LS/DP	System Integration Unit Lite (SIUL)
0xC3F9_4000	16	69	LS/DP	WakeUp Unit (WKPU)
0xC3F9_8000	16	70	LS/DP	Multi-Port DDR DRAM Controller (MDDRC)
0xC3F9_C000	64	Reserved		
0xC3FA_C000	16	75	DP	DMA Channel Multiplexer 1 (DMACHMUX_1)
0xC3FB_0000	16	76	LS/DP	Code flash memory 1 Configuration (CFLASH1)
0xC3FB_4000	144	Reserved		
0xC3FD_8000	16	86	LS/DP	System Status and Configuration Module (SSCM)
0xC3FD_C000	16	87	LS/DP	Mode Entry Module (MC_ME)
0xC3FE_0000	16	88	LS/DP	Clock Generation Module (MC_CGM)
0xC3FE_4000	16	89	LS/DP	Reset Generation Module (MC_RGM)
0xC3FE_8000	16	90	LS/DP	Power Control Unit (MC_PCU)
0xC3FE_C000	16	Reserved		
0xC3FF_0000	16	92	LS/DP	Periodic Interrupt Timer (PIT)
0xC3FF_4000	16	93	LS/DP	Self-Test Control Unit (STCU)
0xC3FF_8000	981024	Reserved		
0xFFE0_0000	16	32	LS/DP	Analog to Digital Converter 0 (ADC_0)
0xFFE0_4000	16	33	LS/DP	Analog to Digital Converter 1 (ADC_1)
0xFFE0_8000	16	Reserved		
0xFFE0_C000	16	35	LS/DP	Cross Triggering Unit (CTU_0)

Table 2-2. MPC5675K memory map (continued)

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
0xFFE1_0000	32	Reserved		
0xFFE1_8000	16	38	LS/DP	eTimer_0
0xFFE1_C000	16	39	LS/DP	eTimer_1
0xFFE2_0000	16	40	LS/DP	eTimer_2
0xFFE2_4000	16	41	LS/DP	FlexPWM_0
0xFFE2_8000	16	42	LS/DP	FlexPWM_1
0xFFE2_C000	16	Reserved		
0xFFE3_0000	16	44	LS/DP	Inter-IC Bus Interface Controller 0 (I^2C0)
0xFFE3_4000	16	45	LS/DP	Inter-IC Bus Interface Controller 1 (I^2C1)
0xFFE3_8000	16	46	LS/DP	Inter-IC Bus Interface Controller 2 (I^2C2)
0xFFE3_C000	16	Reserved		
0xFFE4_0000	16	48	LS/DP	LINFlexD_0
0xFFE4_4000	16	49	LS/DP	LINFlexD_1
0xFFE4_8000	16	50	LS/DP	LINFlexD_2
0xFFE4_C000	16	51	LS/DP	LINFlexD_3
0xFFE5_0000	96	Reserved		
0xFFE6_8000	16	58	LS/DP	Cyclic Redundancy Check Unit 0 (CRCU_0)
0xFFE6_C000	16	59	LS/DP	Fault Collection and Control Unit (FCCU)
0xFFE7_0000	576	Reserved		
0xFFF0_0000	16	—	LS	PBRIDGE_0, PBRIDGE_1
			DP	PBRIDGE_0
0xFFF0_4000[3]	16	—	LS	XBAR_0, XBAR_1
			DP	XBAR_0
0xFFF0_8000	32	Reserved		
0xFFF1_0000	16	—	LS	Memory Protection Unit (MPU_0), Memory Protection Unit (MPU_1)
			DP	Memory Protection Unit (MPU_0)
0xFFF1_4000	64	Reserved		
0xFFF2_4000	16	—	LS	Reserved
			DP	Semaphores (SEMA4_0)
0xFFF2_8000	64	Reserved		

Table 2-2. MPC5675K memory map (continued)

Start address	Size (KB)	PCTL number	Mode[1]	Region / module name
0xFFF3_8000	16	—	LS	Software Watchdog (SWT_0), Software Watchdog (SWT_1)
			DP	Software Watchdog (SWT_0)
0xFFF3_C000	16	—	LS	System Timer Module (STM_0), System Timer Module (STM_1)
			DP	System Timer Module (STM_0)
0xFFF4_0000	16	—	LS	Error Correction Status Module (ECSM_0), Error Correction Status Module (ECSM_1)
			DP	Error Correction Status Module (ECSM_0)
0xFFF4_4000	16	—	LS	Direct Memory Access Controller (DMA_0), Direct Memory Access Controller (DMA_1)
			DP	Direct Memory Access Controller (DMA_0)
0xFFF4_8000	16	—	LS	Interrupt Controller (INTC_0), Interrupt Controller (INTC_1)
			DP	Interrupt Controller (INTC_0)
0xFFF4_C000	16	—	LS/DP	FEC (Ethernet)
0xFFF5_0000	16	—	LS/DP	Coherency Module
0xFFF5_4000	240	Reserved		
0xFFF9_0000	16	4	LS/DP	DSPI_0
0xFFF9_4000	16	5	LS/DP	DSPI_1
0xFFF9_8000	16	6	LS/DP	DSPI_2
0xFFF9_C000	144	Reserved		
0xFFFC_0000	16	16	LS/DP	FlexCAN 0 (CAN0)
0xFFFC_4000	16	17	LS/DP	FlexCAN 1 (CAN1)
0xFFFC_8000	16	18	LS/DP	FlexCAN 2 (CAN2)
0xFFFC_C000	16	19	LS/DP	FlexCAN 3 (CAN3)
0xFFFD_0000	48	Reserved		
0xFFFD_C000	16	23	LS	DMA Channel Multiplexer 0 (DMACHMUX_0), DMA Channel Multiplexer 1 (DMACHMUX_1)
			DP	DMA Channel Multiplexer 0 (DMACHMUX_0)
0xFFFE_0000	16	24	LS/DP	FlexRay
0xFFFE_4000	96	Reserved		
0xFFFF_C000	16	31	LS/DP	BAM

1 LS = Lock Step Mode, DP = Decoupled Parallel Mode.

2 Peripherals from address 0x8FF0_4000 to address 0xFFEF_FFFF are tied to PBRIDGE_1

3 Peripherals from address 0xFFF0_4000 to address 0xFFFF_FFFF are tied to PBRIDGE_0.

Chapter 3
Signal Description

3.1 Package pinouts

Figure 3-1 shows the MPC5675K in the 257 MAPBGA package. Figure 3-2, Figure 3-3, Figure 3-4, and Figure 3-5 show the MPC5675K in the 473 MAPBGA package.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
A	VSS_HV_IO	VSS_HV_IO	VDD_HV_IO	nexus MDO[5]	nexus MDO[7]	nexus MDO[9]	flexray CB_TX	flexray CA_TR_EN	VDD_HV_IO	fec RXD[2]	fec RX_CLK	fec RXD[0]	fec MDIO	fec TX_EN	fec TXD[3]	VSS_HV_IO	VSS_HV_IO	A
B	VSS_HV_IO	VSS_HV_IO	mc_cgl clk_out	can1 TXD	nexus MDO [14]	dspi2 CS1	flexray CB_TR_EN	flexray CA_TX	VSS_HV_IO	fec RXD[3]	fec RX_ER	fec RXD[1]	fec TX_ER	fec TX_CLK	can0 TXD	VDD_HV_IO	VSS_HV_IO	B
C	VDD_HV_IO	nexus MDO [15]	VSS_HV_IO	FCCU_F[1]	flexray CB_RX	etimer0 ETC[0]	etimer0 ETC[1]	etimer0 ETC[2]	etimer0 ETC[3]	JCOMP	fec CRS	fec TXD[0]	fec COL	can0 RXD	VSS_HV_PDI	pdi DATA [5]	pdi CLOCK	C
D	nexus MDO [2]	nexus MDO [3]	can1 RXD	dspi0 SOUT	RESERVED	etimer0 ETC[5]	etimer0 ETC[4]	VDD_HV_FLA	VSS_HV_FLA	fec TXD[2]	fec TXD[1]	fec RX_DV	fec MDC	VDD_HV_PDI	VSS_HV_IO	pdi DATA [0]	pdi DATA [1]	D
E	nexus MDO [0]	nexus MDO [1]	flexray CA_RX	NMI										pdi LINE_V	pdi DATA [2]	pdi DATA [3]	pdi DATA [4]	E
F	nexus MDO[6]	nexus MDO [11]	dspi1 SOUT	dspi1 SIN		VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR		mc_cgl clk_out	pdi DATA [6]	pdi DATA [7]	pdi DATA [8]	F
G	nexus MDO [4]	VDD_HV_IO	dspi0 SCK	dspi1 SCK		VDD_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VDD_LV_COR		pdi DATA [9]	pdi DATA [10]	pdi DATA [11]	pdi FRAME_V	G
H	nexus MDO [10]	VSS_HV_IO	dspi0 CS0	dspi1 CS0		VDD_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VDD_LV_COR		pdi DATA [12]	pdi DATA [13]	VDD_HV_PDI	flexpwm 0 X[0]	H
J	nexus MCKO	nexus MDO[8]	dspi2 CS0	dspi2 CS2		VDD_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VDD_LV_COR		pdi DATA [14]	pdi DATA [15]	VSS_HV_PDI	flexpwm 0 X[1]	J
K	nexus MSEO_B[0]	nexus MSEO_B[1]	nexus RDY_B	dspi0 SIN		VDD_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VDD_LV_COR		flexpwm 0 X[2]	flexpwm 0 X[3]	flexpwm 0 A[1]	flexpwm 0 B[0]	K
L	nexus EVTO_B	nexus EVTI_B	dspi2 SCK	nexus MDO [13]		VDD_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VSS_LV_COR	VDD_LV_COR		VDD_HV_DRAM_VREF	TCK	flexpwm 0 B[1]	TDO	L
M	VDD_HV_OSC	VDD_HV_IO	dspi1 CS2	nexus MDO [12]		VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR	VDD_LV_COR		flexpwm 0 B[2]	TDI	TMS	flexpwm 1 A[1]	M
N	XTALIN	VSS_HV_IO	dspi0 CS3	VSS_LV_PLL										flexpwm 0 B[3]	flexpwm 0 A[2]	flexpwm 1 A[0]	flexpwm 1 B[0]	N
P	VSS_HV_OSC	RESET	dspi0 CS2	VDD_LV_PLL	etimer1 ETC[1]	etimer1 ETC[2]	adc0 AN[0]	etimer1 ETC[3]	VSS_HV_IO	VDD_HV_IO	adc0_adc1 AN[14]	etimer1 ETC[4]	etimer1 ETC[5]	VDD_HV_IO	flexpwm 0 A[3]	flexpwm 0 A[0]	flexpwm 1 B[1]	P
R	XTAL OUT	FCCU_F[0]	VSS_HV_IO	dspi1 CS3	adc2 AN[0]	adc2 AN[3]	VDD_HV_ADR_13	adc2_adc3 AN[14]	VDD_HV_ADR_02	adc0 AN[2]	adc0_adc1 AN[13]	adc1 AN[1]	VREG_CTRL	lin0 TXD	VSS_HV_IO	flexpwm 1 A[2]	flexpwm 1 B[2]	R
T	VSS_HV_IO	VDD_HV_IO	dspi2 SOUT	adc3 AN[0]	adc3 AN[3]	adc2 AN[2]	VSS_HV_ADR_13	adc2_adc3 AN[13]	VSS_HV_ADR_02	adc0 AN[1]	adc0_adc1 AN[12]	adc1 AN[0]	adc1 AN[2]	lin0 RXD	etimer1 ETC[0]	VDD_HV_IO	VSS_HV_IO	T
U	VSS_HV_IO	VSS_HV_IO	dspi2 SIN	adc3 AN[1]	adc3 AN[2]	adc2 AN[1]	adc2_adc3 AN[11]	adc2_adc3 AN[12]	VDD_HV_ADV	VSS_HV_ADV	adc0_adc1 AN[11]	VREG_INT_ENABLE	RESET_SUP	VDD_HV_PMU	VSS_HV_PMU	VSS_HV_IO	VSS_HV_IO	U
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Figure 3-1. MPC5675K 257 MAPBGA pinout (top view)

	1	2	3	4	5	6	7	8	9	10	11	12
A	VSS_ HV_IO	VSS_ HV_IO	VDD_ HV_IO	nexus MDO[5]	nexus MDO[7]	nexus MDO[9]	flexray CB_TX	flexray CA_TR_EN	fec RX_DV	fec MDIO	fec TX_CLK	fec TX_EN
B	VSS_ HV_IO	VSS_ HV_IO	mc_cgl clk_out	can1 TXD	nexus MDO[14]	dspi2 CS1	flexray CB_TR_EN	flexray CA_TX	fec RXD[3]	fec RX_ER	fec TXD[0]	fec RXD[0]
C	VDD_ HV_IO	nexus MDO[15]	VSS_ HV_IO	FCCU_ F[1]	flexray CB_RX	etimer0 ETC[4]	etimer0 ETC[1]	etimer0 ETC[2]	etimer0 ETC[3]	fec TXD[2]	fec TXD[1]	fec CRS
D	nexus MDO[1]	nexus MDO[3]	can1 RXD	dspi0 SOUT	RESERVED	etimer0 ETC[5]	etimer0 ETC[0]	VDD_ HV_IO	VSS_ HV_IO	JCOMP	VSS_ HV_IO	VSS_ HV_FLA
E	nexus MDO[0]	nexus MDO[2]	flexray CA_RX	NMI								
F	nexus MDO[10]	nexus MDO[11]	nexus MDO[6]	nexus MDO[4]		VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR
G	nexus MCKO	VDD_ HV_IO	nexus MDO[8]	nexus MSEO_B[1]		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
H	nexus EVTO_B	VSS_ HV_IO	nexus MSEO_B[0]	nexus EVTI_B		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
J	nexus RDY_B	nexus MDO[13]	nexus MDO[12]	dspi1 SIN		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
K	dspi0 SCK	dspi1 CS0	dspi1 SCK	dspi1 SOUT		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
L	dspi0 CS0	dspi2 CS2	dspi2 CS0	VSS_ HV_IO		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
M	flexpwm0 X[0]	VDD_ HV_IO	dspi0 SIN	VDD_ HV_IO		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR

Figure 3-2. MPC5675K 473 MAPBGA pinout (northwest, viewed from above)

	1	2	3	4	5	6	7	8	9	10	11	12
N	flexpwm0 A[0]	VSS_ HV_IO	flexpwm0 X[1]	flexpwm0 B[2]		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
P	flexpwm0 B[0]	flexpwm0 B[1]	flexpwm0 A[2]	flexpwm0 A[3]		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
R	flexpwm0 X[2]	flexpwm0 X[3]	flexpwm0 A[1]	VSS_ HV_IO		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
T	flexpwm0 B[3]	flexpwm1 A[0]	flexpwm1 A[1]	VDD_ HV_IO		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
U	flexpwm1 B[0]	flexpwm1 B[1]	flexpwm1 A[2]	dspi2 SCK		VDD_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR
V	VDD_ HV_OSC	VDD_ HV_IO	flexpwm1 B[2]	dspi1 CS2		VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR
W	XTALIN	VSS_ HV_IO	dspi0 CS3	VSS_ LV_PLL								
Y	VSS_ HV_OSC	$\overline{\text{RESET}}$	dspi0 CS2	VDD_ LV_PLL	flexpwm1 X[0]	adc3 AN[0]	adc2_adc3 AN[11]	adc2_adc3 AN[14]	etimer1 ETC[1]	etimer1 ETC[2]	etimer1 ETC[3]	VSS_ HV_IO
AA	XTALOUT	FCCU_ F[0]	VSS_ HV_IO	dspi1 CS3	flexpwm1 X[1]	adc3 AN[1]	adc2_adc3 AN[12]	adc2 AN[0]	VDD_ HV_ADV	VSS_ HV_ADV	adc0 AN[2]	adc0 AN[5]
AB	VSS_ HV_IO	VDD_ HV_IO	dspi2 SOUT	flexpwm1 X[2]	flexpwm1 X[3]	adc3 AN[2]	adc2_adc3 AN[13]	adc2 AN[1]	adc2 AN[2]	adc0 AN[0]	adc0 AN[4]	adc0 AN[6]
AC	VSS_ HV_IO	VSS_ HV_IO	dspi2 SIN	flexpwm1 A[3]	flexpwm1 B[3]	adc3 AN[3]	VDD_HV_ ADR_23	VSS_HV_ ADR_23	adc2 AN[3]	adc0 AN[1]	adc0 AN[3]	VDD_ HV_ADR_0

Figure 3-3. MPC5675K 473 MAPBGA pinout (southwest, viewed from above)

	13	14	15	16	17	18	19	20	21	22	23
A	fec TXD[3]	VDD_ HV_IO	pdi DATA[3]	pdi DATA[1]	pdi CLOCK	pdi DATA[7]	pdi DATA[10]	pdi DATA[13]	pdi DATA[15]	VSS_ HV_IO	VSS_ HV_IO
B	fec TX_ER	VSS_ HV_IO	pdi DATA[6]	pdi DATA[4]	pdi DATA[0]	pdi LINE_V	pdi DATA[9]	pdi DATA[14]	can0 TXD	VDD_ HV_IO	VSS_ HV_IO
C	fec RX_CLK	fec RXD[1]	fec COL	pdi DATA[5]	pdi DATA[2]	pdi DATA[8]	pdi DATA[12]	can0 RXD	VSS_ HV_PDI	siul GPIO[197]	dramc CAS
D	VDD_ HV_FLA	fec RXD[2]	fec MDC	VDD_ HV_PDI	VSS_ HV_PDI	pdi DATA[11]	pdi FRAME_V	VDD_ HV_PDI	dramc BA[1]	siul GPIO[195]	dramc BA[0]
E								mc_cgl clk_out	siul GPIO[149]	dramc CS0	dramc BA[2]
F	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR		dramc RAS	siul GPIO[194]	siul GPIO[148]	dramc D[5]
G	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		siul GPIO[196]	dramc DQS[0]	dramc DM[0]	dramc D[7]
H	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc D[2]	VDD_HV_ DRAM_VTT	VDD_HV_ DRAM	VSS_HV_ DRAM
J	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc D[0]	dramc D[1]	dramc D[3]	dramc D[6]
K	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		VSS_ HV_IO	dramc D[4]	dramc D[8]	dramc D[9]
L	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		VDD_ HV_IO	VDD_HV_ DRAM_VTT	VSS_HV_ DRAM	VDD_HV_ DRAM
M	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc ODT	dramc WEB	dramc D[11]	dramc D[10]

Figure 3-4. MPC5675K 473 MAPBGA pinout (northeast, viewed from above)

VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc DQS[1]	dramc DM[1]	dramc D[13]	dramc D[12]	N
VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc D[14]	dramc D[15]	VSS_HV_ DRAM	VDD_HV_ DRAM	P
VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		VDD_HV_ DRAM_VREF	dramc ADD[3]	dramc CKE	dramc CLKB	R
VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc ADD[8]	dramc ADD[9]	dramc ADD[1]	dramc CLK	T
VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VSS_ LV_COR	VDD_ LV_COR		dramc ADD[6]	dramc ADD[12]	VDD_HV_ DRAM	dramc ADD[0]	U
VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR	VDD_ LV_COR		lin0 TXD	dramc ADD[13]	VSS_HV_ DRAM	dramc ADD[2]	V
							lin0 RXD	dramc ADD[14]	dramc ADD[7]	dramc ADD[4]	W
VDD_ HV_IO	adc0_adc1 AN[11]	etimer1 ETC[5]	etimer1 ETC[4]	adc1 AN[8]	adc1 AN[6]	TCK	VDD_HV_IO	dramc ADD[15]	dramc ADD[11]	dramc ADD[5]	Y
adc0 AN[8]	adc0_adc1 AN[12]	adc1 AN[0]	adc1 AN[2]	adc1 AN[5]	adc1 AN[7]	TDI	etimer1 ETC[0]	VSS_HV_IO	lin1 TXD	dramc ADD[10]	AA
adc0 AN[7]	adc0_adc1 AN[13]	adc1 AN[1]	adc1 AN[3]	adc1 AN[4]	TDO	TMS	RESERVED	lin1 RXD	VDD_ HV_IO	VSS_ HV_IO	AB
VSS_ HV_ADR_0	adc0_adc1 AN[14]	VDD_ HV_ADR_1	VSS_ HV_ADR_1	VDD_ HV_PMU	VREG_CTRL	VSS_ HV_PMU	$\overline{\text{RESET}}$_ SUP	VREG_INT_ ENABLE	VSS_ HV_IO	VSS_ HV_IO	AC
13	14	15	16	17	18	19	20	21	22	23	

Figure 3-5. MPC5675K 473 MAPBGA pinout (southeast, viewed from above)

3.2 Pin descriptions

The following sections provide signal descriptions and related information about the functionality and configuration for this microcontroller.

3.2.1 Multiplexed pins

Table 3-1 shows the pin multiplexing for the MPC5675K in the 257 MAPBGA package. Table 3-2 shows the pin multiplexing for the MPC5675K in the 473 MAPBGA package.

Table 3-1. 257 MAPBGA pin multiplexing

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
A4	GPIO	nexus MDO[5][1]	A0: siul_GPIO[114] A1: _ A2: npc_wrapper_MDO[5] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A5	GPIO	nexus MDO[7][1]	A0: siul_GPIO[112] A1: _ A2: npc_wrapper_MDO[7] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A6	GPIO	nexus MDO[9][1]	A0: siul_GPIO[110] A1: _ A2: npc_wrapper_MDO[9] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A7	GPIO	flexray CB_TX	A0: siul_GPIO[51] A1: flexray_CB_TX A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
A8	GPIO	flexray CA_TR_EN	A0: siul_GPIO[47] A1: flexray_CA_TR_EN A2: _ A3: _	I: ctu0_EXT_IN I: flexpwm0_EXT_SYNC I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
A10	GPIO	fec RXD[2]	A0: siul_GPIO[213] A1: _ A2: _ A3: dspi2_SOUT	I: fec_RXD[2] I: _ I: siul_EIRQ[21]	—	disabled	GP Slow/ Medium	VDD_HV_IO
A11	GPIO	fec RX_CLK	A0: siul_GPIO[209] A1: flexray_DBG2 A2: etimer2_ETC[2] A3: dspi0_CS6	I: fec_RX_CLK I: _ I: siul_EIRQ[25]	—	disabled	GP Slow/ Medium	VDD_HV_IO
A12	GPIO	fec RXD[0]	A0: siul_GPIO[211] A1: i2c1_clock A2: _ A3: _	I: fec_RXD[0] I: _ I: siul_EIRQ[27]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
A13	GPIO	fec MDIO	A0: siul_GPIO[198] A1: fec_MDIO A2: _ A3: dspi2_CS0	I: _ I: _ I: siul_EIRQ[28]	—	disabled	GP Slow/ Medium	VDD_HV_IO
A14	GPIO	fec TX_EN	A0: siul_GPIO[200] A1: fec_TX_EN A2: _ A3: lin0_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
A15	GPIO	fec TXD[3]	A0: siul_GPIO[204] A1: fec_TXD[3] A2: _ A3: dspi2_CS2	I: flexpwm1_FAULT[2] I: _ I: siul_EIRQ[29]	—	disabled	GP Slow/ Medium	VDD_HV_IO
B3	GPIO	mc_cgl clk_out	A0: siul_GPIO[22] A1: mc_cgl_clk_out A2: etimer2_ETC[5] A3: _	I: _ I: _ I: siul_EIRQ[18]	—	disabled	GP Slow/ Fast	VDD_HV_IO
B4	GPIO	can1 TXD	A0: siul_GPIO[14] A1: can1_TXD A2: _ A3: _	I: _ I: _ I: siul_EIRQ[13]	—	disabled	GP Slow/ Medium	VDD_HV_IO
B5	GPIO	nexus MDO[14][1]	A0: siul_GPIO[219] A1: _ A2: npc_wrapper_MDO[14] A3: can3_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
B6	GPIO	dspi2 CS1	A0: siul_GPIO[9] A1: dspi2_CS1 A2: _ A3: _	I: flexpwm0_FAULT[0] I: lin3_RXD I: can2_RXD	—	disabled	GP Slow/ Medium	VDD_HV_IO
B7	GPIO	flexray CB_TR_EN	A0: siul_GPIO[52] A1: flexray_CB_TR_EN A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
B8	GPIO	flexray CA_TX	A0: siul_GPIO[48] A1: flexray_CA_TX A2: _ A3: _	I: ctu1_EXT_IN I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
B10	GPIO	fec RXD[3]	A0: siul_GPIO[214] A1: i2c1_data A2: _ A3: _	I: fec_RXD[3] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B11	GPIO	fec RX_ER	A0: siul_GPIO[215] A1: _ A2: _ A3: dspi0_CS1	I: fec_RX_ER I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B12	GPIO	fec RXD[1]	A0: siul_GPIO[212] A1: dspi1_CS1 A2: etimer2_ETC[5] A3: _	I: fec_RXD[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B13	GPIO	fec TX_ER	A0: siul_GPIO[205] A1: fec_TX_ER A2: dspi2_CS3 A3: _	I: flexpwm1_FAULT[3] I: lin0_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B14	GPIO	fec TX_CLK	A0: siul_GPIO[207] A1: flexray_DBG0 A2: etimer2_ETC[4] A3: dspi0_CS4	I: fec_TX_CLK I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B15	GPIO	can0 TXD	A0: siul_GPIO[16] A1: can0_TXD A2: _ A3: sscm_DEBUG[0]	I: _ I: _ I: siul_EIRQ[15]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C2	GPIO	nexus MDO[15][1]	A0: siul_GPIO[220] A1: _ A2: npc_wrapper_MDO[15] A3: _	I: can3_RXD I: can2_RXD I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
C5	GPIO	flexray CB_RX	A0: siul_GPIO[50] A1: _ A2: ctu1_EXT_TGR A3: _	I: flexray_CB_RX I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C6	GPIO	etimer0 ETC[0]	A0: siul_GPIO[0] A1: etimer0_ETC[0] A2: _ A3: _	I: dspi2_SIN I: _ I: siul_EIRQ[0]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C7	GPIO	etimer0 ETC[1]	A0: siul_GPIO[1] A1: etimer0_ETC[1] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[1]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C8	GPIO	etimer0 ETC[2]	A0: siul_GPIO[2] A1: etimer0_ETC[2] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[2]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C9	GPIO	etimer0 ETC[3]	A0: siul_GPIO[3] A1: etimer0_ETC[3] A2: _ A3: _	I: _ I: mc_rgm_ABS[2] I: siul_EIRQ[3]	—	pulldown	GP Slow/ Medium	VDD_HV_IO
C11	GPIO	fec CRS	A0: siul_GPIO[208] A1: flexray_DBG1 A2: etimer2_ETC[3] A3: dspi0_CS5	I: fec_CRS I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C12	GPIO	fec TXD[0]	A0: siul_GPIO[201] A1: fec_TXD[0] A2: etimer2_ETC[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C13	GPIO	fec COL	A0: siul_GPIO[206] A1: fec_COL A2: _ A3: lin1_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
C14	GPIO	can0 RXD	A0: siul_GPIO[17] A1: _ A2: _ A3: sscm_DEBUG[1]	I: can0_RXD I: can1_RXD I: siul_EIRQ[16]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C16	GPIO	pdi DATA[5]	A0: siul_GPIO[136] A1: flexpwm2_A[0] A2: _ A3: etimer1_ETC[0]	I: pdi_DATA[5] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
C17	GPIO	pdi CLOCK	A0: siul_GPIO[128] A1: flexpwm2_B[1] A2: _ A3: etimer1_ETC[3]	I: pdi_CLOCK I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
D1	GPIO	nexus MDO[2][1]	A0: siul_GPIO[85] A1: _ A2: npc_wrapper_MDO[2] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
D2	GPIO	nexus MDO[3][1]	A0: siul_GPIO[84] A1: _ A2: npc_wrapper_MDO[3] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
D3	GPIO	can1 RXD	A0: siul_GPIO[15] A1: _ A2: _ A3: _	I: can1_RXD I: can0_RXD I: siul_EIRQ[14]	—	disabled	GP Slow/ Medium	VDD_HV_IO
D4	GPIO	dspi0 SOUT	A0: siul_GPIO[38] A1: dspi0_SOUT A2: _ A3: sscm_DEBUG[6]	I: _ I: _ I: siul_EIRQ[24]	—	disabled	GP Slow/ Medium	VDD_HV_IO
D6	GPIO	etimer0 ETC[5]	A0: siul_GPIO[44] A1: etimer0_ETC[5] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
D7	GPIO	etimer0 ETC[4]	A0: siul_GPIO[43] A1: etimer0_ETC[4] A2: _ A3: _	I: _ I: mc_rgm_ABS[0] I: _	—	pulldown	GP Slow/ Medium	VDD_HV_IO
D10	GPIO	fec TXD[2]	A0: siul_GPIO[203] A1: fec_TXD[2] A2: _ A3: _	I: flexpwm1_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D11	GPIO	fec TXD[1]	A0: siul_GPIO[202] A1: fec_TXD[1] A2: _ A3: dspi2_SCK	I: flexpwm1_FAULT[0] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D12	GPIO	fec RX_DV	A0: siul_GPIO[210] A1: flexray_DBG3 A2: etimer2_ETC[0] A3: dspi0_CS7	I: fec_RX_DV I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D13	GPIO	fec MDC	A0: siul_GPIO[199] A1: fec_MDC A2: _ A3: _	I: _ I: lin1_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D16	GPIO	pdi DATA[0]	A0: siul_GPIO[131] A1: _ A2: lin3_TXD A3: _	I: pdi_DATA[0] I: _ I: flexpwm2_FAULT[2]	—	disabled	PDI Medium	VDD_HV_PDI
D17	GPIO	pdi DATA[1]	A0: siul_GPIO[132] A1: flexpwm2_B[3] A2: _ A3: _	I: pdi_DATA[1] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
E2	GPIO	nexus MDO[1][1]	A0: siul_GPIO[86] A1: _ A2: npc_wrapper_MDO[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
E3	GPIO	flexray CA_RX	A0: siul_GPIO[49] A1: _ A2: ctu0_EXT_TGR A3: _	I: flexray_CA_RX I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
E14	GPIO	pdi LINE_V	A0: siul_GPIO[129] A1: _ A2: lin2_TXD A3: _	I: pdi_LINE_V I: _ I: flexpwm2_FAULT[0]	—	disabled	PDI Medium	VDD_HV_PDI
E15	GPIO	pdi DATA[2]	A0: siul_GPIO[133] A1: flexpwm2_A[1] A2: _ A3: etimer1_ETC[2]	I: pdi_DATA[2] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
E16	GPIO	pdi DATA[3]	A0: siul_GPIO[134] A1: flexpwm2_X[1] A2: _ A3: _	I: pdi_DATA[3] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
E17	GPIO	pdi DATA[4]	A0: siul_GPIO[135] A1: flexpwm2_A[2] A2: _ A3: etimer1_ETC[4]	I: pdi_DATA[4] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
F1	GPIO	nexus MDO[6][1]	A0: siul_GPIO[113] A1: _ A2: npc_wrapper_MDO[6] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
F2	GPIO	nexus MDO[11][1]	A0: siul_GPIO[108] A1: _ A2: npc_wrapper_MDO[11] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
F3	GPIO	dspi1 SOUT	A0: siul_GPIO[7] A1: dspi1_SOUT A2: _ A3: _	I: _ I: _ I: siul_EIRQ[7]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
F4	GPIO	dspi1 SIN	A0: siul_GPIO[8] A1: _ A2: _ A3: _	I: dspi1_SIN I: _ I: siul_EIRQ[8]	—	disabled	GP Slow/ Medium	VDD_HV_IO
F14	GPIO	mc_cgl clk_out	A0: siul_GPIO[233] A1: mc_cgl_clk_out A2: etimer2_ETC[5] A3: _	I: _ I: _ I: _	—	disabled	PDI Fast	VDD_HV_PDI
F15	GPIO	pdi DATA[6]	A0: siul_GPIO[137] A1: flexpwm2_B[0] A2: _ A3: etimer1_ETC[1]	I: pdi_DATA[6] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
F16	GPIO	pdi DATA[7]	A0: siul_GPIO[138] A1: flexpwm2_B[2] A2: _ A3: etimer1_ETC[5]	I: pdi_DATA[7] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
F17	GPIO	pdi DATA[8]	A0: siul_GPIO[139] A1: flexpwm2_A[3] A2: _ A3: _	I: pdi_DATA[8] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
G1	GPIO	nexus MDO[4][1]	A0: siul_GPIO[115] A1: _ A2: npc_wrapper_MDO[4] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
G3	GPIO	dspi0 SCK	A0: siul_GPIO[37] A1: dspi0_SCK A2: _ A3: sscm_DEBUG[5]	I: flexpwm0_FAULT[3] I: _ I: siul_EIRQ[23]	—	disabled	GP Slow/ Medium	VDD_HV_IO
G4	GPIO	dspi1 SCK	A0: siul_GPIO[6] A1: dspi1_SCK A2: _ A3: _	I: _ I: _ I: siul_EIRQ[6]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
G14	GPIO	pdi DATA[9]	A0: siul_GPIO[140] A1: flexpwm2_X[2] A2: _ A3: _	I: pdi_DATA[9] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
G15	GPIO	pdi DATA[10]	A0: siul_GPIO[141] A1: flexpwm2_X[3] A2: _ A3: _	I: pdi_DATA[10] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
G16	GPIO	pdi DATA[11]	A0: siul_GPIO[142] A1: flexpwm2_X[0] A2: _ A3: _	I: pdi_DATA[11] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
G17	GPIO	pdi FRAME_V	A0: siul_GPIO[130] A1: _ A2: _ A3: _	I: pdi_FRAME_V I: lin2_RXD I: flexpwm2_FAULT[1]	—	disabled	PDI Medium	VDD_HV_PDI
H1	GPIO	nexus MDO[10][1]	A0: siul_GPIO[109] A1: _ A2: npc_wrapper_MDO[10] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
H3	GPIO	dspi0 CS0	A0: siul_GPIO[36] A1: dspi0_CS0 A2: _ A3: sscm_DEBUG[4]	I: _ I: _ I: siul_EIRQ[22]	—	disabled	GP Slow/ Medium	VDD_HV_IO
H4	GPIO	dspi1 CS0	A0: siul_GPIO[5] A1: dspi1_CS0 A2: _ A3: dspi0_CS7	I: _ I: _ I: siul_EIRQ[5]	—	disabled	GP Slow/ Medium	VDD_HV_IO
H14	GPIO	pdi DATA[12]	A0: siul_GPIO[143] A1: _ A2: _ A3: _	I: pdi_DATA[12] I: lin3_RXD I: flexpwm2_FAULT[3]	—	disabled	PDI Medium	VDD_HV_PDI

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
H15	GPIO	pdi DATA[13]	A0: siul_GPIO[144] A1: pdi_SENS_SEL[2] A2: ctu1_EXT_TGR A3: _	I: pdi_DATA[13] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
H17	GPIO	flexpwm0 X[0]	A0: siul_GPIO[194] A1: flexpwm0_X[0] A2: ebi_AD28 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
J1	GPIO	nexus MCKO	A0: siul_GPIO[87] A1: _ A2: npc_wrapper_MCKO A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
J2	GPIO	nexus MDO[8][1]	A0: siul_GPIO[111] A1: _ A2: npc_wrapper_MDO[8] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
J3	GPIO	dspi2 CS0	A0: siul_GPIO[10] A1: dspi2_CS0 A2: _ A3: can3_TXD	I: _ I: _ I: siul_EIRQ[9]	—	disabled	GP Slow/ Medium	VDD_HV_IO
J4	GPIO	dspi2 CS2	A0: siul_GPIO[42] A1: dspi2_CS2 A2: lin3_TXD A3: can2_TXD	I: flexpwm0_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
J14	GPIO	pdi DATA[14]	A0: siul_GPIO[145] A1: pdi_SENS_SEL[1] A2: i2c2_clock A3: _	I: pdi_DATA[14] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
J15	GPIO	pdi DATA[15]	A0: siul_GPIO[146] A1: pdi_SENS_SEL[0] A2: i2c2_data A3: _	I: pdi_DATA[15] I: ctu1_EXT_IN I: _	—	disabled	PDI Medium	VDD_HV_PDI

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
J17	GPIO	flexpwm0 X[1]	A0: siul_GPIO[195] A1: flexpwm0_X[1] A2: ebi_AD29 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
K1	GPIO	nexus MSEO_B[0][1]	A0: siul_GPIO[89] A1: _ A2: npc_wrapper_MSEO_B[0] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
K2	GPIO	nexus MSEO_B[1][1]	A0: siul_GPIO[88] A1: _ A2: npc_wrapper_MSEO_B[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
K3	GPIO	nexus RDY_B	A0: siul_GPIO[216] A1: _ A2: nexus_RDY_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
K4	GPIO	dspi0 SIN	A0: siul_GPIO[39] A1: _ A2: _ A3: sscm_DEBUG[7]	I: dspi0_SIN I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
K14	GPIO	flexpwm0 X[2]	A0: siul_GPIO[196] A1: flexpwm0_X[2] A2: ebi_AD30 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
K15	GPIO	flexpwm0 X[3]	A0: siul_GPIO[197] A1: flexpwm0_X[3] A2: ebi_AD31 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
K16	GPIO	flexpwm0 A[1]	A0: siul_GPIO[149] A1: _ A2: ebi_RD_WR A3: flexpwm0_A[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
K17	GPIO	flexpwm0 B[0]	A0: siul_GPIO[148] A1: _ A2: ebi_CLKOUT A3: flexpwm0_B[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
L1	GPIO	nexus EVTO_B	A0: siul_GPIO[90] A1: _ A2: npc_wrapper_EVTO_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
L2	GPIO	nexus EVTI_B	A0: siul_GPIO[91] A1: _ A2: leo_sor_proxy_EVTI_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
L3	GPIO	dspi2 SCK	A0: siul_GPIO[11] A1: dspi2_SCK A2: _ A3: _	I: can3_RXD I: _ I: siul_EIRQ[10]	—	disabled	GP Slow/ Medium	VDD_HV_IO
L4	GPIO	nexus MDO[13][1]	A0: siul_GPIO[218] A1: _ A2: npc_wrapper_MDO[13] A3: _	I: can2_RXD I: can3_RXD I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
L16	GPIO	flexpwm0 B[1]	A0: siul_GPIO[150] A1: dramc_CS0 A2: ebi_TS A3: flexpwm0_B[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
L17	GPIO	TDO	A0: siul_GPIO[20] A1: jtagc_TDO A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
M3	GPIO	dspi1 CS2	A0: siul_GPIO[56] A1: dspi1_CS2 A2: _ A3: dspi0_CS5	I: flexpwm0_FAULT[3] I: lin2_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
M4	GPIO	nexus MDO[12][1]	A0: siul_GPIO[217] A1: _ A2: npc_wrapper_MDO[12] A3: can2_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
M14	GPIO	flexpwm0 B[2]	A0: siul_GPIO[152] A1: dramc_CAS A2: ebi_WE_BE_1 A3: flexpwm0_B[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
M15	GPIO	TDI	A0: siul_GPIO[21] A1: _ A2: _ A3: _	I: jtagc_TDI I: _ I: _	—	pullup	GP Slow/ Medium	VDD_HV_IO
M17	GPIO	flexpwm1 A[1]	A0: siul_GPIO[157] A1: dramc_ODT A2: ebi_CS1 A3: flexpwm1_A[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
N3	GPIO	dspi0 CS3	A0: siul_GPIO[53] A1: dspi0_CS3 A2: i2c2_clock A3: _	I: flexpwm0_FAULT[2] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
N14	GPIO	flexpwm0 B[3]	A0: siul_GPIO[154] A1: dramc_BA[0] A2: ebi_WE_BE_3 A3: flexpwm0_B[3]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
N15	GPIO	flexpwm0 A[2]	A0: siul_GPIO[151] A1: dramc_RAS A2: ebi_WE_BE_0 A3: flexpwm0_A[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
N16	GPIO	flexpwm1 A[0]	A0: siul_GPIO[155] A1: dramc_BA[1] A2: ebi_BDIP A3: flexpwm1_A[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
N17	GPIO	flexpwm1 B[0]	A0: siul_GPIO[156] A1: dramc_BA[2] A2: ebi_CS0 A3: flexpwm1_B[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
P3	GPIO	dspi0 CS2	A0: siul_GPIO[54] A1: dspi0_CS2 A2: i2c2_data A3: _	I: flexpwm0_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
P5	GPIO	etimer1 ETC[1]	A0: siul_GPIO[45] A1: etimer1_ETC[1] A2: _ A3: _	I: ctu0_EXT_IN I: flexpwm0_EXT_SYNC I: ctu1_EXT_IN	—	disabled	GP Slow/ Medium	VDD_HV_IO
P6	GPIO	etimer1 ETC[2]	A0: siul_GPIO[46] A1: etimer1_ETC[2] A2: ctu0_EXT_TGR A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
P7	ANA	adc0 AN[0]	—	siul_GPI[23] lin0_RXD	AN: adc0_AN[0]		Analog	VDD_HV_ADR02
P8	GPIO	etimer1 ETC[3]	A0: siul_GPIO[92] A1: etimer1_ETC[3] A2: _ A3: _	I: ctu1_EXT_IN I: mc_rgm_FAB I: siul_EIRQ[30]	—	pulldown	GP Slow/ Medium	VDD_HV_IO
P11	ANA	adc0_adc1 AN[14]	—	siul_GPI[28]	AN: adc0_adc1_AN[14]		Analog Shared	VDD_HV_ADR02
P12	GPIO	etimer1 ETC[4]	A0: siul_GPIO[93] A1: etimer1_ETC[4] A2: ctu1_EXT_TGR A3: _	I: _ I: _ I: siul_EIRQ[31]	—	disabled	GP Slow/ Medium	VDD_HV_IO
P13	GPIO	etimer1 ETC[5]	A0: siul_GPIO[78] A1: etimer1_ETC[5] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[26]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
P15	GPIO	flexpwm0 A[3]	A0: siul_GPIO[153] A1: dramc_WEB A2: ebi_WE_BE_2 A3: flexpwm0_A[3]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
P16	GPIO	flexpwm0 A[0]	A0: siul_GPIO[147] A1: dramc_CKE A2: ebi_OE A3: flexpwm0_A[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
P17	GPIO	flexpwm1 B[1]	A0: siul_GPIO[163] A1: dramc_ADD[5] A2: ebi_ADD13 A3: flexpwm1_B[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
R4	GPIO	dspi1 CS3	A0: siul_GPIO[55] A1: dspi1_CS3 A2: lin2_TXD A3: dspi0_CS4	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
R5	ANA	adc2 AN[0]	—	siul_GPI[221]	AN: adc2_AN[0]	—	Analog	VDD_HV_ADR02
R6	ANA	adc2 AN[3]	—	siul_GPI[224]	AN: adc2_AN[3]	—	Analog	VDD_HV_ADR02
R8	ANA	adc2_adc3 AN[14]	—	siul_GPI[228]	AN: adc2_adc3_AN[14]	—	Analog Shared	VDD_HV_ADR13
R10	ANA	adc0 AN[2]	—	siul_GPI[33]	AN: adc0_AN[2]	—	Analog	VDD_HV_ADR02
R11	ANA	adc0_adc1 AN[13]	—	siul_GPI[27]	AN: adc0_adc1_AN[13]	—	Analog Shared	VDD_HV_ADR02
R12	ANA	adc1 AN[1]	—	siul_GPI[30] etimer0_ETC[4] siul_EIRQ[19]	AN: adc1_AN[1]	—	Analog	VDD_HV_ADR13

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
R14	GPIO	lin0 TXD	A0: siul_GPIO[18] A1: lin0_TXD A2: i2c0_clock A3: sscm_DEBUG[2]	I: _ I: _ I: siul_EIRQ[17]	—	disabled	GP Slow/ Medium	VDD_HV_IO
R16	GPIO	flexpwm1 A[2]	A0: siul_GPIO[164] A1: dramc_ADD[6] A2: ebi_ADD14 A3: flexpwm1_A[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
R17	GPIO	flexpwm1 B[2]	A0: siul_GPIO[165] A1: dramc_ADD[7] A2: ebi_ADD15 A3: flexpwm1_B[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_IO
T3	GPIO	dspi2 SOUT	A0: siul_GPIO[12] A1: dspi2_SOUT A2: _ A3: _	I: _ I: _ I: siul_EIRQ[11]	—	disabled	GP Slow/ Medium	VDD_HV_IO
T4	ANA	adc3 AN[0]	—	siul_GPI[229]	AN: adc3_AN[0]	—	Analog	VDD_HV_ADR13
T5	ANA	adc3 AN[3]	—	siul_GPI[232]	AN: adc3_AN[3]	—	Analog	VDD_HV_ADR13
T6	ANA	adc2 AN[2]	—	siul_GPI[223]	AN: adc2_AN[2]	—	Analog	VDD_HV_ADR02
T8	ANA	adc2_adc3 AN[13]	—	siul_GPI[227]	AN: adc2_adc3_AN[13]	—	Analog Shared	VDD_HV_ADR02
T10	ANA	adc0 AN[1]	—	siul_GPI[24] etimer0_ETC[5]	AN: adc0_AN[1]	—	Analog	VDD_HV_ADR02
T11	ANA	adc0_adc1 AN[12]	—	siul_GPI[26]	AN: adc0_adc1_AN[12]	—	Analog Shared	VDD_HV_ADR02

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
T12	ANA	adc1 AN[0]	—	siul_GPI[29] lin1_RXD	AN: adc1_AN[0]	—	Analog	VDD_HV_ADR13
T13	ANA	adc1 AN[2]	—	siul_GPI[31] siul_EIRQ[20]	AN: adc1_AN[2]	—	Analog	VDD_HV_ADR13
T14	GPIO	lin0 RXD	A0: siul_GPIO[19] A1: _ A2: i2c0_data A3: sscm_DEBUG[3]	I: lin0_RXD I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
T15	GPIO	etimer1 ETC[0]	A0: siul_GPIO[4] A1: etimer1_ETC[0] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[4]	—	disabled	GP Slow/ Medium	VDD_HV_IO
U3	GPIO	dspi2 SIN	A0: siul_GPIO[13] A1: _ A2: _ A3: _	I: dspi2_SIN I: flexpwm0_FAULT[0] I: siul_EIRQ[12]	—	disabled	GP Slow/ Medium	VDD_HV_IO
U4	ANA	adc3 AN[1]	—	siul_GPI[230]	AN: adc3_AN[1]	—	Analog	VDD_HV_ADR13
U5	ANA	adc3 AN[2]	—	siul_GPI[231]	AN: adc3_AN[2]	—	Analog	VDD_HV_ADR13
U6	ANA	adc2 AN[1]	—	siul_GPI[222]	AN: adc2_AN[1]	—	Analog	VDD_HV_ADR02
U7	ANA	adc2_adc3 AN[11]	—	siul_GPI[225]	AN: adc2_adc3_AN[11]	—	Analog Shared	VDD_HV_ADR13
U8	ANA	adc2_adc3 AN[12]	—	siul_GPI[226]	AN: adc2_adc3_AN[12]	—	Analog Shared	VDD_HV_ADR13

Table 3-1. 257 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional inputs	Analog inputs	Weak pull during reset	Pad type	Power domain
U11	ANA	adc0_adc1 AN[11]	—	siul_GPI[25]	AN: adc0_adc1_AN[11]	—	Analog Shared	VDD_HV_ADR02
END OF 257 MAPBGA PIN MULTIPLEXING TABLE								

1 Do not connect pin directly to a power supply or ground.

Table 3-2. 473 MAPBGA pin multiplexing

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
A4	GPIO	nexus MDO[5][1]	A0: siul_GPIO[114] A1: _ A2: npc_wrapper_MDO[5] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A5	GPIO	nexus MDO[7][1]	A0: siul_GPIO[112] A1: _ A2: npc_wrapper_MDO[7] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A6	GPIO	nexus MDO[9][1]	A0: siul_GPIO[110] A1: _ A2: npc_wrapper_MDO[9] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
A7	GPIO	flexray CB_TX	A0: siul_GPIO[51] A1: flexray_CB_TX A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
A8	GPIO	flexray CA_TR_EN	A0: siul_GPIO[47] A1: flexray_CA_TR_EN A2: _ A3: _	I: ctu0_EXT_IN I: flexpwm0_EXT_SYNC I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
A9	GPIO	fec RX_DV	A0: siul_GPIO[210] A1: flexray_DBG3 A2: etimer2_ETC[0] A3: dspi0_CS7	I: fec_RX_DV I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
A10	GPIO	fec MDIO	A0: siul_GPIO[198] A1: fec_MDIO A2: _ A3: dspi2_CS0	I: _ I: _ I: siul_EIRQ[28]	—	disabled	GP Slow/ Medium	VDD_HV_IO
A11	GPIO	fec TX_CLK	A0: siul_GPIO[207] A1: flexray_DBG0 A2: etimer2_ETC[4] A3: dspi0_CS4	I: fec_TX_CLK I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
A12	GPIO	fec TX_EN	A0: siul_GPIO[200] A1: fec_TX_EN A2: _ A3: lin0_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
A13	GPIO	fec TXD[3]	A0: siul_GPIO[204] A1: fec_TXD[3] A2: _ A3: dspi2_CS2	I: flexpwm1_FAULT[2] I: _ I: siul_EIRQ[29]	—	disabled	GP Slow/ Medium	VDD_HV_IO
A15	GPIO	pdi DATA[3]	A0: siul_GPIO[134] A1: flexpwm2_X[1] A2: _ A3: _	I: pdi_DATA[3] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
A16	GPIO	pdi DATA[1]	A0: siul_GPIO[132] A1: flexpwm2_B[3] A2: _ A3: _	I: pdi_DATA[1] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
A17	GPIO	pdi CLOCK	A0: siul_GPIO[128] A1: flexpwm2_B[1] A2: _ A3: etimer1_ETC[3]	I: pdi_CLOCK I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
A18	GPIO	pdi DATA[7]	A0: siul_GPIO[138] A1: flexpwm2_B[2] A2: _ A3: etimer1_ETC[5]	I: pdi_DATA[7] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
A19	GPIO	pdi DATA[10]	A0: siul_GPIO[141] A1: flexpwm2_X[3] A2: _ A3: _	I: pdi_DATA[10] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
A20	GPIO	pdi DATA[13]	A0: siul_GPIO[144] A1: pdi_SENS_SEL[2] A2: ctu1_EXT_TGR A3: _	I: pdi_DATA[13] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
A21	GPIO	pdi DATA[15]	A0: siul_GPIO[146] A1: pdi_SENS_SEL[0] A2: i2c2_data A3: _	I: pdi_DATA[15] I: ctu1_EXT_IN I: _	—	disabled	PDI Medium	VDD_HV_PDI
B3	GPIO	mc_cgl clk_out	A0: siul_GPIO[22] A1: mc_cgl_clk_out A2: etimer2_ETC[5] A3: _	I: _ I: _ I: siul_EIRQ[18]	—	disabled	GP Slow/ Fast	VDD_HV_IO
B4	GPIO	can1 TXD	A0: siul_GPIO[14] A1: can1_TXD A2: _ A3: _	I: _ I: _ I: siul_EIRQ[13]	—	disabled	GP Slow/ Medium	VDD_HV_IO
B5	GPIO	nexus MDO[14][1]	A0: siul_GPIO[219] A1: _ A2: npc_wrapper_MDO[14] A3: can3_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
B6	GPIO	dspi2 CS1	A0: siul_GPIO[9] A1: dspi2_CS1 A2: _ A3: _	I: flexpwm0_FAULT[0] I: lin3_RXD I: can2_RXD	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
B7	GPIO	flexray CB_TR_EN	A0: siul_GPIO[52] A1: flexray_CB_TR_EN A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
B8	GPIO	flexray CA_TX	A0: siul_GPIO[48] A1: flexray_CA_TX A2: _ A3: _	I: ctu1_EXT_IN I: _ I: _	—	disabled	GP Slow/ Symmetric	VDD_HV_IO
B9	GPIO	fec RXD[3]	A0: siul_GPIO[214] A1: i2c1_data A2: _ A3: _	I: fec_RXD[3] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B10	GPIO	fec RX_ER	A0: siul_GPIO[215] A1: _ A2: _ A3: dspi0_CS1	I: fec_RX_ER I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B11	GPIO	fec TXD[0]	A0: siul_GPIO[201] A1: fec_TXD[0] A2: etimer2_ETC[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B12	GPIO	fec RXD[0]	A0: siul_GPIO[211] A1: i2c1_clock A2: _ A3: _	I: fec_RXD[0] I: _ I: siul_EIRQ[27]	—	disabled	GP Slow/ Medium	VDD_HV_IO
B13	GPIO	fec TX_ER	A0: siul_GPIO[205] A1: fec_TX_ER A2: dspi2_CS3 A3: _	I: flexpwm1_FAULT[3] I: lin0_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
B15	GPIO	pdi DATA[6]	A0: siul_GPIO[137] A1: flexpwm2_B[0] A2: _ A3: etimer1_ETC[1]	I: pdi_DATA[6] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
B16	GPIO	pdi DATA[4]	A0: siul_GPIO[135] A1: flexpwm2_A[2] A2: _ A3: etimer1_ETC[4]	I: pdi_DATA[4] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
B17	GPIO	pdi DATA[0]	A0: siul_GPIO[131] A1: _ A2: lin3_TXD A3: _	I: pdi_DATA[0] I: _ I: flexpwm2_FAULT[2]	—	disabled	PDI Medium	VDD_HV_PDI
B18	GPIO	pdi LINE_V	A0: siul_GPIO[129] A1: _ A2: lin2_TXD A3: _	I: pdi_LINE_V I: _ I: flexpwm2_FAULT[0]	—	disabled	PDI Medium	VDD_HV_PDI
B19	GPIO	pdi DATA[9]	A0: siul_GPIO[140] A1: flexpwm2_X[2] A2: _ A3: _	I: pdi_DATA[9] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
B20	GPIO	pdi DATA[14]	A0: siul_GPIO[145] A1: pdi_SENS_SEL[1] A2: i2c2_clock A3: _	I: pdi_DATA[14] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
B21	GPIO	can0 TXD	A0: siul_GPIO[16] A1: can0_TXD A2: _ A3: sscm_DEBUG[0]	I: _ I: _ I: siul_EIRQ[15]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C2	GPIO	nexus MDO[15][1]	A0: siul_GPIO[220] A1: _ A2: npc_wrapper_MDO[15] A3: _	I: can3_RXD I: can2_RXD I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
C5	GPIO	flexray CB_RX	A0: siul_GPIO[50] A1: _ A2: ctu1_EXT_TGR A3: _	I: flexray_CB_RX I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
C6	GPIO	etimer0 ETC[4]	A0: siul_GPIO[43] A1: etimer0_ETC[4] A2: _ A3: _	I: _ I: mc_rgm_ABS[0] I: _	—	pulldown	GP Slow/ Medium	VDD_HV_IO
C7	GPIO	etimer0 ETC[1]	A0: siul_GPIO[1] A1: etimer0_ETC[1] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[1]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C8	GPIO	etimer0 ETC[2]	A0: siul_GPIO[2] A1: etimer0_ETC[2] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[2]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C9	GPIO	etimer0 ETC[3]	A0: siul_GPIO[3] A1: etimer0_ETC[3] A2: _ A3: _	I: _ I: mc_rgm_ABS[2] I: siul_EIRQ[3]	—	pulldown	GP Slow/ Medium	VDD_HV_IO
C10	GPIO	fec TXD[2]	A0: siul_GPIO[203] A1: fec_TXD[2] A2: _ A3: _	I: flexpwm1_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C11	GPIO	fec TXD[1]	A0: siul_GPIO[202] A1: fec_TXD[1] A2: _ A3: dspi2_SCK	I: flexpwm1_FAULT[0] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C12	GPIO	fec CRS	A0: siul_GPIO[208] A1: flexray_DBG1 A2: etimer2_ETC[3] A3: dspi0_CS5	I: fec_CRS I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C13	GPIO	fec RX_CLK	A0: siul_GPIO[209] A1: flexray_DBG2 A2: etimer2_ETC[2] A3: dspi0_CS6	I: fec_RX_CLK I: _ I: siul_EIRQ[25]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
C14	GPIO	fec RXD[1]	A0: siul_GPIO[212] A1: dspi1_CS1 A2: etimer2_ETC[5] A3: _	I: fec_RXD[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C15	GPIO	fec COL	A0: siul_GPIO[206] A1: fec_COL A2: _ A3: lin1_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
C16	GPIO	pdi DATA[5]	A0: siul_GPIO[136] A1: flexpwm2_A[0] A2: _ A3: etimer1_ETC[0]	I: pdi_DATA[5] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
C17	GPIO	pdi DATA[2]	A0: siul_GPIO[133] A1: flexpwm2_A[1] A2: _ A3: etimer1_ETC[2]	I: pdi_DATA[2] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
C18	GPIO	pdi DATA[8]	A0: siul_GPIO[139] A1: flexpwm2_A[3] A2: _ A3: _	I: pdi_DATA[8] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
C19	GPIO	pdi DATA[12]	A0: siul_GPIO[143] A1: _ A2: _ A3: _	I: pdi_DATA[12] I: lin3_RXD I: flexpwm2_FAULT[3]	—	disabled	PDI Medium	VDD_HV_PDI
C20	GPIO	can0 RXD	A0: siul_GPIO[17] A1: _ A2: _ A3: sscm_DEBUG[1]	I: can0_RXD I: can1_RXD I: siul_EIRQ[16]	—	disabled	GP Slow/ Medium	VDD_HV_IO
C22	GPIO	siul GPIO[197]	A0: siul_GPIO[197] A1: flexpwm0_X[3] A2: ebi_AD31 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
C23	GPIO	dramc CAS	A0: siul_GPIO[152] A1: dramc_CAS A2: ebi_WE_BE_1 A3: flexpwm0_B[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
D1	GPIO	nexus MDO[1][1]	A0: siul_GPIO[86] A1: _ A2: npc_wrapper_MDO[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
D2	GPIO	nexus MDO[3][1]	A0: siul_GPIO[84] A1: _ A2: npc_wrapper_MDO[3] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
D3	GPIO	can1 RXD	A0: siul_GPIO[15] A1: _ A2: _ A3: _	I: can1_RXD I: can0_RXD I: siul_EIRQ[14]	—	disabled	GP Slow/ Medium	VDD_HV_IO
D4	GPIO	dspi0 SOUT	A0: siul_GPIO[38] A1: dspi0_SOUT A2: _ A3: sscm_DEBUG[6]	I: _ I: _ I: siul_EIRQ[24]	—	disabled	GP Slow/ Medium	VDD_HV_IO
D6	GPIO	etimer0 ETC[5]	A0: siul_GPIO[44] A1: etimer0_ETC[5] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D7	GPIO	etimer0 ETC[0]	A0: siul_GPIO[0] A1: etimer0_ETC[0] A2: _ A3: _	I: dspi2_SIN I: _ I: siul_EIRQ[0]	—	disabled	GP Slow/ Medium	VDD_HV_IO
D14	GPIO	fec RXD[2]	A0: siul_GPIO[213] A1: _ A2: _ A3: dspi2_SOUT	I: fec_RXD[2] I: _ I: siul_EIRQ[21]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
D15	GPIO	fec MDC	A0: siul_GPIO[199] A1: fec_MDC A2: _ A3: _	I: _ I: lin1_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
D18	GPIO	pdi DATA[11]	A0: siul_GPIO[142] A1: flexpwm2_X[0] A2: _ A3: _	I: pdi_DATA[11] I: _ I: _	—	disabled	PDI Medium	VDD_HV_PDI
D19	GPIO	pdi FRAME_V	A0: siul_GPIO[130] A1: _ A2: _ A3: _	I: pdi_FRAME_V I: lin2_RXD I: flexpwm2_FAULT[1]	—	disabled	PDI Medium	VDD_HV_PDI
D21	GPIO	dramc BA[1]	A0: siul_GPIO[155] A1: dramc_BA[1] A2: ebi_BDIP A3: flexpwm1_A[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
D22	GPIO	siul GPIO[195]	A0: siul_GPIO[195] A1: flexpwm0_X[1] A2: ebi_AD29 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
D23	GPIO	dramc BA[0]	A0: siul_GPIO[154] A1: dramc_BA[0] A2: ebi_WE_BE_3 A3: flexpwm0_B[3]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
E2	GPIO	nexus MDO[2][1]	A0: siul_GPIO[85] A1: _ A2: npc_wrapper_MDO[2] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
E3	GPIO	flexray CA_RX	A0: siul_GPIO[49] A1: _ A2: ctu0_EXT_TGR A3: _	I: flexray_CA_RX I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
E20	GPIO	mc_cgl clk_out	A0: siul_GPIO[233] A1: mc_cgl_clk_out A2: etimer2_ETC[5] A3: _	I: _ I: _ I: _	—	disabled	PDI Fast	VDD_HV_PDI
E21	GPIO	siul GPIO[149]	A0: siul_GPIO[149] A1: _ A2: ebi_RD_WR A3: flexpwm0_A[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
E22	GPIO	dramc CS0	A0: siul_GPIO[150] A1: dramc_CS0 A2: ebi_TS A3: flexpwm0_B[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
E23	GPIO	dramc BA[2]	A0: siul_GPIO[156] A1: dramc_BA[2] A2: ebi_CS0 A3: flexpwm1_B[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
F1	GPIO	nexus MDO[10][1]	A0: siul_GPIO[109] A1: _ A2: npc_wrapper_MDO[10] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
F2	GPIO	nexus MDO[11][1]	A0: siul_GPIO[108] A1: _ A2: npc_wrapper_MDO[11] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
F3	GPIO	nexus MDO[6][1]	A0: siul_GPIO[113] A1: _ A2: npc_wrapper_MDO[6] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
F4	GPIO	nexus MDO[4][1]	A0: siul_GPIO[115] A1: _ A2: npc_wrapper_MDO[4] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
F20	GPIO	dramc RAS	A0: siul_GPIO[151] A1: dramc_RAS A2: ebi_WE_BE_0 A3: flexpwm0_A[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
F21	GPIO	siul GPIO[194]	A0: siul_GPIO[194] A1: flexpwm0_X[0] A2: ebi_AD28 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
F22	GPIO	siul GPIO[148]	A0: siul_GPIO[148] A1: _ A2: ebi_CLKOUT A3: flexpwm0_B[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
F23	GPIO	dramc D[5]	A0: siul_GPIO[179] A1: dramc_D[5] A2: ebi_AD13 A3: ebi_ADD29	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
G1	GPIO	nexus MCKO	A0: siul_GPIO[87] A1: _ A2: npc_wrapper_MCKO A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
G3	GPIO	nexus MDO[8][1]	A0: siul_GPIO[111] A1: _ A2: npc_wrapper_MDO[8] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
G4	GPIO	nexus MSEO_B[1][1]	A0: siul_GPIO[88] A1: _ A2: npc_wrapper_MSEO_B[1] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
G20	GPIO	siul GPIO[196]	A0: siul_GPIO[196] A1: flexpwm0_X[2] A2: ebi_AD30 A3: _	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
G21	GPIO	dramc DQS[0]	A0: siul_GPIO[190] A1: dramc_DQS[0] A2: ebi_AD24 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
G22	GPIO	dramc DM[0]	A0: siul_GPIO[192] A1: dramc_DM[0] A2: ebi_AD26 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
G23	GPIO	dramc D[7]	A0: siul_GPIO[181] A1: dramc_D[7] A2: ebi_AD15 A3: ebi_ADD31	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
H1	GPIO	nexus EVTO_B	A0: siul_GPIO[90] A1: _ A2: npc_wrapper_EVTO_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
H3	GPIO	nexus MSEO_B[0] [1]	A0: siul_GPIO[89] A1: _ A2: npc_wrapper_MSEO_B[0] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
H4	GPIO	nexus EVTI_B	A0: siul_GPIO[91] A1: _ A2: leo_sor_proxy_EVTI_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
H20	GPIO	dramc D[2]	A0: siul_GPIO[176] A1: dramc_D[2] A2: ebi_AD10 A3: ebi_ADD26	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
J1	GPIO	nexus RDY_B	A0: siul_GPIO[216] A1: _ A2: nexus_RDY_B A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
J2	GPIO	nexus MDO[13][1]	A0: siul_GPIO[218] A1: _ A2: npc_wrapper_MDO[13] A3: _	I: can2_RXD I: can3_RXD I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
J3	GPIO	nexus MDO[12][1]	A0: siul_GPIO[217] A1: _ A2: npc_wrapper_MDO[12] A3: can2_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
J4	GPIO	dspi1 SIN	A0: siul_GPIO[8] A1: _ A2: _ A3: _	I: dspi1_SIN I: _ I: siul_EIRQ[8]	—	disabled	GP Slow/ Medium	VDD_HV_IO
J20	GPIO	dramc D[0]	A0: siul_GPIO[174] A1: dramc_D[0] A2: ebi_AD8 A3: ebi_ADD24	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
J21	GPIO	dramc D[1]	A0: siul_GPIO[175] A1: dramc_D[1] A2: ebi_AD9 A3: ebi_ADD25	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
J22	GPIO	dramc D[3]	A0: siul_GPIO[177] A1: dramc_D[3] A2: ebi_AD11 A3: ebi_ADD27	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
J23	GPIO	dramc D[6]	A0: siul_GPIO[180] A1: dramc_D[6] A2: ebi_AD14 A3: ebi_ADD30	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
K1	GPIO	dspi0 SCK	A0: siul_GPIO[37] A1: dspi0_SCK A2: _ A3: sscm_DEBUG[5]	I: flexpwm0_FAULT[3] I: _ I: siul_EIRQ[23]	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
K2	GPIO	dspi1 CS0	A0: siul_GPIO[5] A1: dspi1_CS0 A2: _ A3: dspi0_CS7	I: _ I: _ I: siul_EIRQ[5]	—	disabled	GP Slow/ Medium	VDD_HV_IO
K3	GPIO	dspi1 SCK	A0: siul_GPIO[6] A1: dspi1_SCK A2: _ A3: _	I: _ I: _ I: siul_EIRQ[6]	—	disabled	GP Slow/ Medium	VDD_HV_IO
K4	GPIO	dspi1 SOUT	A0: siul_GPIO[7] A1: dspi1_SOUT A2: _ A3: _	I: _ I: _ I: siul_EIRQ[7]	—	disabled	GP Slow/ Medium	VDD_HV_IO
K21	GPIO	dramc D[4]	A0: siul_GPIO[178] A1: dramc_D[4] A2: ebi_AD12 A3: ebi_ADD28	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
K22	GPIO	dramc D[8]	A0: siul_GPIO[182] A1: dramc_D[8] A2: ebi_AD16 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
K23	GPIO	dramc D[9]	A0: siul_GPIO[183] A1: dramc_D[9] A2: ebi_AD17 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
L1	GPIO	dspi0 CS0	A0: siul_GPIO[36] A1: dspi0_CS0 A2: _ A3: sscm_DEBUG[4]	I: _ I: _ I: siul_EIRQ[22]	—	disabled	GP Slow/ Medium	VDD_HV_IO
L2	GPIO	dspi2 CS2	A0: siul_GPIO[42] A1: dspi2_CS2 A2: lin3_TXD A3: can2_TXD	I: flexpwm0_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
L3	GPIO	dspi2 CS0	A0: siul_GPIO[10] A1: dspi2_CS0 A2: _ A3: can3_TXD	I: _ I: _ I: siul_EIRQ[9]	—	disabled	GP Slow/ Medium	VDD_HV_IO
M1	GPIO	flexpwm0 X[0]	A0: siul_GPIO[57] A1: flexpwm0_X[0] A2: lin2_TXD A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
M3	GPIO	dspi0 SIN	A0: siul_GPIO[39] A1: _ A2: _ A3: sscm_DEBUG[7]	I: dspi0_SIN I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
M20	GPIO	dramc ODT	A0: siul_GPIO[157] A1: dramc_ODT A2: ebi_CS1 A3: flexpwm1_A[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
M21	GPIO	dramc WEB	A0: siul_GPIO[153] A1: dramc_WEB A2: ebi_WE_BE_2 A3: flexpwm0_A[3]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
M22	GPIO	dramc D[11]	A0: siul_GPIO[185] A1: dramc_D[11] A2: ebi_AD19 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
M23	GPIO	dramc D[10]	A0: siul_GPIO[184] A1: dramc_D[10] A2: ebi_AD18 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
N1	GPIO	flexpwm0 A[0]	A0: siul_GPIO[58] A1: flexpwm0_A[0] A2: _ A3: _	I: _ I: etimer0_ETC[0] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
N3	GPIO	flexpwm0X[1]	A0: siul_GPIO[60] A1: flexpwm0_X[1] A2: _ A3: _	I: lin2_RXD I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
N4	GPIO	flexpwm0B[2]	A0: siul_GPIO[100] A1: flexpwm0_B[2] A2: _ A3: _	I: _ I: etimer0_ETC[5] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
N20	GPIO	dramcDQS[1]	A0: siul_GPIO[191] A1: dramc_DQS[1] A2: ebi_AD25 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
N21	GPIO	dramcDM[1]	A0: siul_GPIO[193] A1: dramc_DM[1] A2: ebi_AD27 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
N22	GPIO	dramcD[13]	A0: siul_GPIO[187] A1: dramc_D[13] A2: ebi_AD21 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
N23	GPIO	dramcD[12]	A0: siul_GPIO[186] A1: dramc_D[12] A2: ebi_AD20 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
P1	GPIO	flexpwm0B[0]	A0: siul_GPIO[59] A1: flexpwm0_B[0] A2: _ A3: _	I: _ I: etimer0_ETC[1] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
P2	GPIO	flexpwm0B[1]	A0: siul_GPIO[62] A1: flexpwm0_B[1] A2: _ A3: _	I: _ I: etimer0_ETC[3] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
P3	GPIO	flexpwm0 A[2]	A0: siul_GPIO[99] A1: flexpwm0_A[2] A2: _ A3: _	I: _ I: etimer0_ETC[4] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
P4	GPIO	flexpwm0 A[3]	A0: siul_GPIO[102] A1: flexpwm0_A[3] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
P20	GPIO	dramc D[14]	A0: siul_GPIO[188] A1: dramc_D[14] A2: ebi_AD22 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
P21	GPIO	dramc D[15]	A0: siul_GPIO[189] A1: dramc_D[15] A2: ebi_AD23 A3: _	I: _ I: _ I: _	—	disabled	DRAM DQ	VDD_HV_DRAM
R1	GPIO	flexpwm0 X[2]	A0: siul_GPIO[98] A1: flexpwm0_X[2] A2: lin3_TXD A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
R2	GPIO	flexpwm0 X[3]	A0: siul_GPIO[101] A1: flexpwm0_X[3] A2: _ A3: _	I: lin3_RXD I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
R3	GPIO	flexpwm0 A[1]	A0: siul_GPIO[80] A1: flexpwm0_A[1] A2: _ A3: _	I: _ I: etimer0_ETC[2] I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
R21	GPIO	dramc ADD[3]	A0: siul_GPIO[161] A1: dramc_ADD[3] A2: ebi_ADD11 A3: ebi_TEA	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
R22	GPIO	dramc CKE	A0: siul_GPIO[147] A1: dramc_CKE A2: ebi_OE A3: flexpwm0_A[0]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
T1	GPIO	flexpwm0 B[3]	A0: siul_GPIO[103] A1: flexpwm0_B[3] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
T2	GPIO	flexpwm1 A[0]	A0: siul_GPIO[117] A1: flexpwm1_A[0] A2: _ A3: can2_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
T3	GPIO	flexpwm1 A[1]	A0: siul_GPIO[120] A1: flexpwm1_A[1] A2: _ A3: can3_TXD	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
T20	GPIO	dramc ADD[8]	A0: siul_GPIO[166] A1: dramc_ADD[8] A2: ebi_AD0 A3: ebi_ADD16	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
T21	GPIO	dramc ADD[9]	A0: siul_GPIO[167] A1: dramc_ADD[9] A2: ebi_AD1 A3: ebi_ADD17	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
T22	GPIO	dramc ADD[1]	A0: siul_GPIO[159] A1: dramc_ADD[1] A2: ebi_ADD9 A3: ebi_CS3	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
U1	GPIO	flexpwm1 B[0]	A0: siul_GPIO[118] A1: flexpwm1_B[0] A2: _ A3: _	I: can2_RXD I: can3_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
U2	GPIO	flexpwm1 B[1]	A0: siul_GPIO[121] A1: flexpwm1_B[1] A2: _ A3: _	I: can3_RXD I: can2_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
U3	GPIO	flexpwm1 A[2]	A0: siul_GPIO[123] A1: flexpwm1_A[2] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
U4	GPIO	dspi2 SCK	A0: siul_GPIO[11] A1: dspi2_SCK A2: _ A3: _	I: can3_RXD I: _ I: siul_EIRQ[10]	—	disabled	GP Slow/ Medium	VDD_HV_IO
U20	GPIO	dramc ADD[6]	A0: siul_GPIO[164] A1: dramc_ADD[6] A2: ebi_ADD14 A3: flexpwm1_A[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
U21	GPIO	dramc ADD[12]	A0: siul_GPIO[170] A1: dramc_ADD[12] A2: ebi_AD4 A3: ebi_ADD20	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
U23	GPIO	dramc ADD[0]	A0: siul_GPIO[158] A1: dramc_ADD[0] A2: ebi_ADD8 A3: ebi_CS2	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
V3	GPIO	flexpwm1 B[2]	A0: siul_GPIO[124] A1: flexpwm1_B[2] A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
V4	GPIO	dspi1 CS2	A0: siul_GPIO[56] A1: dspi1_CS2 A2: _ A3: dspi0_CS5	I: flexpwm0_FAULT[3] I: lin2_RXD I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
V20	GPIO	lin0 TXD	A0: siul_GPIO[18] A1: lin0_TXD A2: i2c0_clock A3: sscm_DEBUG[2]	I: _ I: _ I: siul_EIRQ[17]	—	disabled	GP Slow/ Medium	VDD_HV_IO
V21	GPIO	dramc ADD[13]	A0: siul_GPIO[171] A1: dramc_ADD[13] A2: ebi_AD5 A3: ebi_ADD21	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
V23	GPIO	dramc ADD[2]	A0: siul_GPIO[160] A1: dramc_ADD[2] A2: ebi_ADD10 A3: ebi_TA	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
W3	GPIO	dspi0 CS3	A0: siul_GPIO[53] A1: dspi0_CS3 A2: i2c2_clock A3: _	I: flexpwm0_FAULT[2] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
W20	GPIO	lin0 RXD	A0: siul_GPIO[19] A1: _ A2: i2c0_data A3: sscm_DEBUG[3]	I: lin0_RXD I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
W21	GPIO	dramc ADD[14]	A0: siul_GPIO[172] A1: dramc_ADD[14] A2: ebi_AD6 A3: ebi_ADD22	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
W22	GPIO	dramc ADD[7]	A0: siul_GPIO[165] A1: dramc_ADD[7] A2: ebi_ADD15 A3: flexpwm1_B[2]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
W23	GPIO	dramc ADD[4]	A0: siul_GPIO[162] A1: dramc_ADD[4] A2: ebi_ADD12 A3: ebi_ALE	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
Y3	GPIO	dspi0 CS2	A0: siul_GPIO[54] A1: dspi0_CS2 A2: i2c2_data A3: _	I: flexpwm0_FAULT[1] I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y5	GPIO	flexpwm1 X[0]	A0: siul_GPIO[116] A1: flexpwm1_X[0] A2: etimer2_ETC[0] A3: dspi0_CS1	I: ctu0_EXT_IN I: ctu1_EXT_IN I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y6	ANA	adc3 AN[0]	—	siul_GPI[229]	AN: adc3_AN[0]	—	Analog	VDD_HV_ADR23
Y7	ANA	adc2_adc3 AN[11]	—	siul_GPI[225]	AN: adc2_adc3_AN[11]	—	Analog Shared	VDD_HV_ADR23
Y8	ANA	adc2_adc3 AN[14]	—	siul_GPI[228]	AN: adc2_adc3_AN[14]	—	Analog Shared	VDD_HV_ADR23
Y9	GPIO	etimer1 ETC[1]	A0: siul_GPIO[45] A1: etimer1_ETC[1] A2: _ A3: _	I: ctu0_EXT_IN I: flexpwm0_EXT_SYNC I: ctu1_EXT_IN	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y10	GPIO	etimer1 ETC[2]	A0: siul_GPIO[46] A1: etimer1_ETC[2] A2: ctu0_EXT_TGR A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y11	GPIO	etimer1 ETC[3]	A0: siul_GPIO[92] A1: etimer1_ETC[3] A2: _ A3: _	I: ctu1_EXT_IN I: mc_rgm_FAB I: siul_EIRQ[30]	—	pulldown	GP Slow/ Medium	VDD_HV_IO
Y14	ANA	adc0_adc1 AN[11]	—	siul_GPI[25]	AN: adc0_adc1_AN[11]	—	Analog Shared	VDD_HV_ADR0

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
Y15	GPIO	etimer1 ETC[5]	A0: siul_GPIO[78] A1: etimer1_ETC[5] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[26]	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y16	GPIO	etimer1 ETC[4]	A0: siul_GPIO[93] A1: etimer1_ETC[4] A2: ctu1_EXT_TGR A3: _	I: _ I: _ I: siul_EIRQ[31]	—	disabled	GP Slow/ Medium	VDD_HV_IO
Y17	ANA	adc1 AN[8]	—	siul_GPI[74]	AN: adc1_AN[8]	—	Analog	VDD_HV_ADR1
Y18	ANA	adc1 AN[6]	—	siul_GPI[76]	AN: adc1_AN[6]	—	Analog	VDD_HV_ADR1
Y21	GPIO	dramc ADD[15]	A0: siul_GPIO[173] A1: dramc_ADD[15] A2: ebi_AD7 A3: ebi_ADD23	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
Y22	GPIO	dramc ADD[11]	A0: siul_GPIO[169] A1: dramc_ADD[11] A2: ebi_AD3 A3: ebi_ADD19	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
Y23	GPIO	dramc ADD[5]	A0: siul_GPIO[163] A1: dramc_ADD[5] A2: ebi_ADD13 A3: flexpwm1_B[1]	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
AA4	GPIO	dspi1 CS3	A0: siul_GPIO[55] A1: dspi1_CS3 A2: lin2_TXD A3: dspi0_CS4	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AA5	GPIO	flexpwm1 X[1]	A0: siul_GPIO[119] A1: flexpwm1_X[1] A2: etimer2_ETC[1] A3: dspi0_CS4	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
AA6	ANA	adc3 AN[1]	—	siul_GPI[230]	AN: adc3_AN[1]	—	Analog	VDD_HV_ADR23
AA7	ANA	adc2_adc3 AN[12]	—	siul_GPI[226]	AN: adc2_adc3_AN[12]	—	Analog Shared	VDD_HV_ADR23
AA8	ANA	adc2 AN[0]	—	siul_GPI[221]	AN: adc2_AN[0]	—	Analog	VDD_HV_ADR23
AA11	ANA	adc0 AN[2]	—	siul_GPI[33]	AN: adc0_AN[2]	—	Analog	VDD_HV_ADR0
AA12	ANA	adc0 AN[5]	—	siul_GPI[66]	AN: adc0_AN[5]	—	Analog	VDD_HV_ADR0
AA13	ANA	adc0 AN[8]	—	siul_GPI[69]	AN: adc0_AN[8]	—	Analog	VDD_HV_ADR0
AA14	ANA	adc0_adc1 AN[12]	—	siul_GPI[26]	AN: adc0_adc1_AN[12]	—	Analog Shared	VDD_HV_ADR0
AA15	ANA	adc1 AN[0]	—	siul_GPI[29] lin1_RXD	AN: adc1_AN[0]	—	Analog	VDD_HV_ADR1
AA16	ANA	adc1 AN[2]	—	siul_GPI[31] siul_EIRQ[20]	AN: adc1_AN[2]	—	Analog	VDD_HV_ADR1
AA17	ANA	adc1 AN[5]	—	siul_GPI[64]	AN: adc1_AN[5]	—	Analog	VDD_HV_ADR1

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
AA18	ANA	adc1 AN[7]	—	siul_GPI[73]	AN: adc1_AN[7]	—	Analog	VDD_HV_ADR1
AA19	GPIO	TDI	A0: siul_GPIO[21] A1: _ A2: _ A3: _	I: jtagc_TDI I: _ I: _	—	pullup	GP Slow/ Medium	VDD_HV_IO
AA20	GPIO	etimer1 ETC[0]	A0: siul_GPIO[4] A1: etimer1_ETC[0] A2: _ A3: _	I: _ I: _ I: siul_EIRQ[4]	—	disabled	GP Slow/ Medium	VDD_HV_IO
AA22	GPIO	lin1 TXD	A0: siul_GPIO[94] A1: lin1_TXD A2: i2c1_clock A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AA23	GPIO	dramc ADD[10]	A0: siul_GPIO[168] A1: dramc_ADD[10] A2: ebi_AD2 A3: ebi_ADD18	I: _ I: _ I: _	—	disabled	DRAM ACC	VDD_HV_DRAM
AB3	GPIO	dspi2 SOUT	A0: siul_GPIO[12] A1: dspi2_SOUT A2: _ A3: _	I: _ I: _ I: siul_EIRQ[11]	—	disabled	GP Slow/ Medium	VDD_HV_IO
AB4	GPIO	flexpwm1 X[2]	A0: siul_GPIO[122] A1: flexpwm1_X[2] A2: etimer2_ETC[2] A3: dspi0_CS5	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AB5	GPIO	flexpwm1 X[3]	A0: siul_GPIO[125] A1: flexpwm1_X[3] A2: etimer2_ETC[3] A3: dspi0_CS6	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AB6	ANA	adc3 AN[2]	—	siul_GPI[231]	AN: adc3_AN[2]	—	Analog	VDD_HV_ADR23

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
AB7	ANA	adc2_adc3 AN[13]	—	siul_GPI[227]	AN: adc2_adc3_AN[13]	—	Analog Shared	VDD_HV_ADR23
AB8	ANA	adc2 AN[1]	—	siul_GPI[222]	AN: adc2_AN[1]	—	Analog	VDD_HV_ADR23
AB9	ANA	adc2 AN[2]	—	siul_GPI[223]	AN: adc2_AN[2]	—	Analog	VDD_HV_ADR23
AB10	ANA	adc0 AN[0]	—	siul_GPI[23] lin0_RXD	AN: adc0_AN[0]	—	Analog	VDD_HV_ADR0
AB11	ANA	adc0 AN[4]	—	siul_GPI[70]	AN: adc0_AN[4]	—	Analog	VDD_HV_ADR0
AB12	ANA	adc0 AN[6]	—	siul_GPI[71]	AN: adc0_AN[6]	—	Analog	VDD_HV_ADR0
AB13	ANA	adc0 AN[7]	—	siul_GPI[68]	AN: adc0_AN[7]	—	Analog	VDD_HV_ADR0
AB14	ANA	adc0_adc1 AN[13]	—	siul_GPI[27]	AN: adc0_adc1_AN[13]	—	Analog Shared	VDD_HV_ADR0
AB15	ANA	adc1 AN[1]	—	siul_GPI[30] etimer0_ETC[4] siul_EIRQ[19]	AN: adc1_AN[1]	—	Analog	VDD_HV_ADR1
AB16	ANA	adc1 AN[3]	—	siul_GPI[32]	AN: adc1_AN[3]	—	Analog	VDD_HV_ADR1

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
AB17	ANA	adc1 AN[4]	—	siul_GPI[75]	AN: adc1_AN[4]	—	Analog	VDD_HV_ADR1
AB18	GPIO	TDO	A0: siul_GPIO[20] A1: jtagc_TDO A2: _ A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Fast	VDD_HV_IO
AB21	GPIO	lin1 RXD	A0: siul_GPIO[95] A1: _ A2: i2c1_data A3: _	I: lin1_RXD I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AC3	GPIO	dspi2 SIN	A0: siul_GPIO[13] A1: _ A2: _ A3: _	I: dspi2_SIN I: flexpwm0_FAULT[0] I: siul_EIRQ[12]	—	disabled	GP Slow/ Medium	VDD_HV_IO
AC4	GPIO	flexpwm1 A[3]	A0: siul_GPIO[126] A1: flexpwm1_A[3] A2: etimer2_ETC[4] A3: dspi0_CS7	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AC5	GPIO	flexpwm1 B[3]	A0: siul_GPIO[127] A1: flexpwm1_B[3] A2: etimer2_ETC[5] A3: _	I: _ I: _ I: _	—	disabled	GP Slow/ Medium	VDD_HV_IO
AC6	ANA	adc3 AN[3]	—	siul_GPI[232]	AN: adc3_AN[3]	—	GP Slow/ Medium	VDD_HV_ADR23
AC9	ANA	adc2 AN[3]	—	siul_GPI[224]	AN: adc2_AN[3]	—	Analog	VDD_HV_ADR23
AC10	ANA	adc0 AN[1]	—	siul_GPI[24] etimer0_ETC[5]	AN: adc0_AN[1]	—	Analog	VDD_HV_ADR0

Table 3-2. 473 MAPBGA pin multiplexing (continued)

Ball number	Ball type	Ball name	Alternate I/O	Additional Inputs	Analog Inputs	Weak pull during reset	Pad type	Power domain
AC11	ANA	adc0 AN[3]	—	siul_GPI[34]	AN: adc0_AN[3]	—	Analog	VDD_HV_ADR0
AC14	ANA	adc0_adc1 AN[14]	—	siul_GPI[28]	AN: adc0_adc1_AN[14]	—	Analog Shared	VDD_HV_ADR0
END OF 473 MAPBGA PIN MULTIPLEXING TABLE								

1 Do not connect pin directly to a power supply or ground.

3.2.2 Power supply and reference voltage pins

Table 3-3 and Table 3-5 show the supply pins for the MPC5675K microcontrollers in the 257 MAPBGA and 473 MAPBGA packages, respectively.

Table 3-4 and Table 3-6 list the pin positions that are not populated on the MPC5675K microcontrollers for the 257 MAPBGA and 473 MAPBGA packages, respectively.

Table 3-3. 257 MAPBGA supply pins

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
V_{DD}					
A3	VDD_HV_IO	VDD_HV	F9	VDD_LV_COR	VDD_LV
A9	VDD_HV_IO	VDD_HV	F10	VDD_LV_COR	VDD_LV
B16	VDD_HV_IO	VDD_HV	F11	VDD_LV_COR	VDD_LV
C1	VDD_HV_IO	VDD_HV	F12	VDD_LV_COR	VDD_LV
G2	VDD_HV_IO	VDD_HV	G6	VDD_LV_COR	VDD_LV
M2	VDD_HV_IO	VDD_HV	G12	VDD_LV_COR	VDD_LV
P10	VDD_HV_IO	VDD_HV	H6	VDD_LV_COR	VDD_LV
P14	VDD_HV_IO	VDD_HV	H12	VDD_LV_COR	VDD_LV
T2	VDD_HV_IO	VDD_HV	J6	VDD_LV_COR	VDD_LV
T16	VDD_HV_IO	VDD_HV	J12	VDD_LV_COR	VDD_LV
L14	VDD_HV_DRAM_VREF	VDD_HV	K6	VDD_LV_COR	VDD_LV
D8	VDD_HV_FLA	VDD_HV	K12	VDD_LV_COR	VDD_LV
M1	VDD_HV_OSC	VDD_HV	L6	VDD_LV_COR	VDD_LV
D14	VDD_HV_PDI	VDD_HV	L12	VDD_LV_COR	VDD_LV
H16	VDD_HV_PDI	VDD_HV	M6	VDD_LV_COR	VDD_LV
U14	VDD_HV_PMU	VDD_HV	M7	VDD_LV_COR	VDD_LV
R7	VDD_HV_ADR_13	VDD_HV_A	M8	VDD_LV_COR	VDD_LV
R9	VDD_HV_ADR_02	VDD_HV_A	M9	VDD_LV_COR	VDD_LV
U9	VDD_HV_ADV	VDD_HV_A	M10	VDD_LV_COR	VDD_LV
F6	VDD_LV_COR	VDD_LV	M11	VDD_LV_COR	VDD_LV
F7	VDD_LV_COR	VDD_LV	M12	VDD_LV_COR	VDD_LV
F8	VDD_LV_COR	VDD_LV	P4	VDD_LV_PLL	VDD_LV

Table 3-3. 257 MAPBGA supply pins (continued)

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
V_{SS}					
A1	VSS_HV_IO	VSS_HV	G7	VSS_LV_COR	VSS_LV
A2	VSS_HV_IO	VSS_HV	G8	VSS_LV_COR	VSS_LV
A16	VSS_HV_IO	VSS_HV	G9	VSS_LV_COR	VSS_LV
A17	VSS_HV_IO	VSS_HV	G10	VSS_LV_COR	VSS_LV
B1	VSS_HV_IO	VSS_HV	G11	VSS_LV_COR	VSS_LV
B2	VSS_HV_IO	VSS_HV	H7	VSS_LV_COR	VSS_LV
B9	VSS_HV_IO	VSS_HV	H8	VSS_LV_COR	VSS_LV
B17	VSS_HV_IO	VSS_HV	H9	VSS_LV_COR	VSS_LV
C3	VSS_HV_IO	VSS_HV	H10	VSS_LV_COR	VSS_LV
D15	VSS_HV_IO	VSS_HV	H11	VSS_LV_COR	VSS_LV
H2	VSS_HV_IO	VSS_HV	J7	VSS_LV_COR	VSS_LV
N2	VSS_HV_IO	VSS_HV	J8	VSS_LV_COR	VSS_LV
P9	VSS_HV_IO	VSS_HV	J9	VSS_LV_COR	VSS_LV
R3	VSS_HV_IO	VSS_HV	J10	VSS_LV_COR	VSS_LV
R15	VSS_HV_IO	VSS_HV	J11	VSS_LV_COR	VSS_LV
T1	VSS_HV_IO	VSS_HV	K7	VSS_LV_COR	VSS_LV
T17	VSS_HV_IO	VSS_HV	K8	VSS_LV_COR	VSS_LV
U1	VSS_HV_IO	VSS_HV	K9	VSS_LV_COR	VSS_LV
U2	VSS_HV_IO	VSS_HV	K10	VSS_LV_COR	VSS_LV
U16	VSS_HV_IO	VSS_HV	K11	VSS_LV_COR	VSS_LV
U17	VSS_HV_IO	VSS_HV	L7	VSS_LV_COR	VSS_LV
D9	VSS_HV_FLA	VSS_HV	L8	VSS_LV_COR	VSS_LV
P1	VSS_HV_OSC	VSS_HV	L9	VSS_LV_COR	VSS_LV
C15	VSS_HV_PDI	VSS_HV	L10	VSS_LV_COR	VSS_LV
J16	VSS_HV_PDI	VSS_HV	L11	VSS_LV_COR	VSS_LV
T9	VSS_HV_ADR_02	VSS_HV_A	N4	VSS_LV_PLL	VSS_LV
T7	VSS_HV_ADR_13	VSS_HV_A	U15	VSS_HV_PMU	VSS_LV
U10	VSS_HV_ADV	VSS_HV_A			

Table 3-4. 257 MAPBGA pins not populated on package

E5	E6	E7	E8	E9	E10	E11	E12
E13	F5	F13	G5	G13	H5	H13	J5
J13	K5	K13	L5	L13	M5	M13	N5
N6	N7	N8	N9	N10	N11	N12	N13

Table 3-5. 473 MAPBGA supply pins

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
V_{DD}					
A3	VDD_HV_IO	VDD_HV	F15	VDD_LV_COR	VDD_LV
A14	VDD_HV_IO	VDD_HV	F16	VDD_LV_COR	VDD_LV
B22	VDD_HV_IO	VDD_HV	F17	VDD_LV_COR	VDD_LV
C1	VDD_HV_IO	VDD_HV	F18	VDD_LV_COR	VDD_LV
D8	VDD_HV_IO	VDD_HV	G6	VDD_LV_COR	VDD_LV
G2	VDD_HV_IO	VDD_HV	G18	VDD_LV_COR	VDD_LV
L20	VDD_HV_IO	VDD_HV	H6	VDD_LV_COR	VDD_LV
M2	VDD_HV_IO	VDD_HV	H18	VDD_LV_COR	VDD_LV
M4	VDD_HV_IO	VDD_HV	J6	VDD_LV_COR	VDD_LV
T4	VDD_HV_IO	VDD_HV	J18	VDD_LV_COR	VDD_LV
V2	VDD_HV_IO	VDD_HV	K6	VDD_LV_COR	VDD_LV
Y13	VDD_HV_IO	VDD_HV	K18	VDD_LV_COR	VDD_LV
Y20	VDD_HV_IO	VDD_HV	L6	VDD_LV_COR	VDD_LV
AB2	VDD_HV_IO	VDD_HV	L18	VDD_LV_COR	VDD_LV
AB22	VDD_HV_IO	VDD_HV	M6	VDD_LV_COR	VDD_LV
AC12	VDD_HV_ADR_0	VDD_HV_A	M18	VDD_LV_COR	VDD_LV
AC15	VDD_HV_ADR_1	VDD_HV_A	N6	VDD_LV_COR	VDD_LV
AC7	VDD_HV_ADR_23	VDD_HV_A	N18	VDD_LV_COR	VDD_LV
AA9	VDD_HV_ADV	VDD_HV_A	P6	VDD_LV_COR	VDD_LV
H22	VDD_HV_DRAM	VDD_HV	P18	VDD_LV_COR	VDD_LV
L23	VDD_HV_DRAM	VDD_HV	R6	VDD_LV_COR	VDD_LV
P23	VDD_HV_DRAM	VDD_HV	R18	VDD_LV_COR	VDD_LV
U22	VDD_HV_DRAM	VDD_HV	T6	VDD_LV_COR	VDD_LV
R20	VDD_HV_DRAM_VREF	VDD_HV	T18	VDD_LV_COR	VDD_LV
H21	VDD_HV_DRAM_VTT	VDD_HV	U6	VDD_LV_COR	VDD_LV

Table 3-5. 473 MAPBGA supply pins (continued)

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
L21	VDD_HV_DRAM_VTT	VDD_HV	U18	VDD_LV_COR	VDD_LV
D13	VDD_HV_FLA	VDD_HV	V6	VDD_LV_COR	VDD_LV
V1	VDD_HV_OSC	VDD_HV	V7	VDD_LV_COR	VDD_LV
D16	VDD_HV_PDI	VDD_HV	V8	VDD_LV_COR	VDD_LV
D20	VDD_HV_PDI	VDD_HV	V9	VDD_LV_COR	VDD_LV
AC17	VDD_HV_PMU	VDD_HV	V10	VDD_LV_COR	VDD_LV
F6	VDD_LV_COR	VDD_LV	V11	VDD_LV_COR	VDD_LV
F7	VDD_LV_COR	VDD_LV	V12	VDD_LV_COR	VDD_LV
F8	VDD_LV_COR	VDD_LV	V13	VDD_LV_COR	VDD_LV
F9	VDD_LV_COR	VDD_LV	V14	VDD_LV_COR	VDD_LV
F10	VDD_LV_COR	VDD_LV	V15	VDD_LV_COR	VDD_LV
F11	VDD_LV_COR	VDD_LV	V16	VDD_LV_COR	VDD_LV
F12	VDD_LV_COR	VDD_LV	V17	VDD_LV_COR	VDD_LV
F13	VDD_LV_COR	VDD_LV	V18	VDD_LV_COR	VDD_LV
F14	VDD_LV_COR	VDD_LV	Y4	VDD_LV_PLL	VDD_LV
V_{SS}					
A2	VSS_HV_IO	VSS_HV	L7	VSS_LV_COR	VSS_LV
A22	VSS_HV_IO	VSS_HV	L8	VSS_LV_COR	VSS_LV
A23	VSS_HV_IO	VSS_HV	L9	VSS_LV_COR	VSS_LV
B1	VSS_HV_IO	VSS_HV	L10	VSS_LV_COR	VSS_LV
B2	VSS_HV_IO	VSS_HV	L11	VSS_LV_COR	VSS_LV
B14	VSS_HV_IO	VSS_HV	L12	VSS_LV_COR	VSS_LV
B23	VSS_HV_IO	VSS_HV	L13	VSS_LV_COR	VSS_LV
C3	VSS_HV_IO	VSS_HV	L14	VSS_LV_COR	VSS_LV
D9	VSS_HV_IO	VSS_HV	L15	VSS_LV_COR	VSS_LV
D11	VSS_HV_IO	VSS_HV	L16	VSS_LV_COR	VSS_LV
H2	VSS_HV_IO	VSS_HV	L17	VSS_LV_COR	VSS_LV
K20	VSS_HV_IO	VSS_HV	M7	VSS_LV_COR	VSS_LV
L4	VSS_HV_IO	VSS_HV	M8	VSS_LV_COR	VSS_LV
N2	VSS_HV_IO	VSS_HV	M9	VSS_LV_COR	VSS_LV
A1	VSS_HV_IO	VSS_HV	M10	VSS_LV_COR	VSS_LV
R4	VSS_HV_IO	VSS_HV	M11	VSS_LV_COR	VSS_LV

Table 3-5. 473 MAPBGA supply pins (continued)

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
W2	VSS_HV_IO	VSS_HV	M12	VSS_LV_COR	VSS_LV
Y12	VSS_HV_IO	VSS_HV	M13	VSS_LV_COR	VSS_LV
AA3	VSS_HV_IO	VSS_HV	M14	VSS_LV_COR	VSS_LV
AA21	VSS_HV_IO	VSS_HV	M15	VSS_LV_COR	VSS_LV
AB1	VSS_HV_IO	VSS_HV	M16	VSS_LV_COR	VSS_LV
AB23	VSS_HV_IO	VSS_HV	M17	VSS_LV_COR	VSS_LV
AC1	VSS_HV_IO	VSS_HV	N7	VSS_LV_COR	VSS_LV
AC2	VSS_HV_IO	VSS_HV	N8	VSS_LV_COR	VSS_LV
AC22	VSS_HV_IO	VSS_HV	N9	VSS_LV_COR	VSS_LV
AC23	VSS_HV_IO	VSS_HV	N10	VSS_LV_COR	VSS_LV
AC13	VSS_HV_ADR_0	VSS_HV_A	N11	VSS_LV_COR	VSS_LV
AC16	VSS_HV_ADR_1	VSS_HV_A	N12	VSS_LV_COR	VSS_LV
AC8	VSS_HV_ADR_23	VSS_HV_A	N13	VSS_LV_COR	VSS_LV
AA10	VSS_HV_ADV	VSS_HV_A	N14	VSS_LV_COR	VSS_LV
H23	VSS_HV_DRAM	VSS_HV	N15	VSS_LV_COR	VSS_LV
L22	VSS_HV_DRAM	VSS_HV	N16	VSS_LV_COR	VSS_LV
P22	VSS_HV_DRAM	VSS_HV	N17	VSS_LV_COR	VSS_LV
V22	VSS_HV_DRAM	VSS_HV	P7	VSS_LV_COR	VSS_LV
D12	VSS_HV_FLA	VSS_HV	P8	VSS_LV_COR	VSS_LV
Y1	VSS_HV_OSC	VSS_HV	P9	VSS_LV_COR	VSS_LV
C21	VSS_HV_PDI	VSS_HV	P10	VSS_LV_COR	VSS_LV
D17	VSS_HV_PDI	VSS_HV	P11	VSS_LV_COR	VSS_LV
G7	VSS_LV_COR	VSS_LV	P12	VSS_LV_COR	VSS_LV
G8	VSS_LV_COR	VSS_LV	P13	VSS_LV_COR	VSS_LV
G9	VSS_LV_COR	VSS_LV	P14	VSS_LV_COR	VSS_LV
G10	VSS_LV_COR	VSS_LV	P15	VSS_LV_COR	VSS_LV
G11	VSS_LV_COR	VSS_LV	P16	VSS_LV_COR	VSS_LV
G12	VSS_LV_COR	VSS_LV	P17	VSS_LV_COR	VSS_LV
G13	VSS_LV_COR	VSS_LV	R7	VSS_LV_COR	VSS_LV
G14	VSS_LV_COR	VSS_LV	R8	VSS_LV_COR	VSS_LV
G15	VSS_LV_COR	VSS_LV	R9	VSS_LV_COR	VSS_LV
G16	VSS_LV_COR	VSS_LV	R10	VSS_LV_COR	VSS_LV

Table 3-5. 473 MAPBGA supply pins (continued)

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
G17	VSS_LV_COR	VSS_LV	R11	VSS_LV_COR	VSS_LV
H7	VSS_LV_COR	VSS_LV	R12	VSS_LV_COR	VSS_LV
H8	VSS_LV_COR	VSS_LV	R13	VSS_LV_COR	VSS_LV
H9	VSS_LV_COR	VSS_LV	R14	VSS_LV_COR	VSS_LV
H10	VSS_LV_COR	VSS_LV	R15	VSS_LV_COR	VSS_LV
H11	VSS_LV_COR	VSS_LV	R16	VSS_LV_COR	VSS_LV
H12	VSS_LV_COR	VSS_LV	R17	VSS_LV_COR	VSS_LV
H13	VSS_LV_COR	VSS_LV	T7	VSS_LV_COR	VSS_LV
H14	VSS_LV_COR	VSS_LV	T8	VSS_LV_COR	VSS_LV
H15	VSS_LV_COR	VSS_LV	T9	VSS_LV_COR	VSS_LV
H16	VSS_LV_COR	VSS_LV	T10	VSS_LV_COR	VSS_LV
H17	VSS_LV_COR	VSS_LV	T11	VSS_LV_COR	VSS_LV
J7	VSS_LV_COR	VSS_LV	T12	VSS_LV_COR	VSS_LV
J8	VSS_LV_COR	VSS_LV	T13	VSS_LV_COR	VSS_LV
J9	VSS_LV_COR	VSS_LV	T14	VSS_LV_COR	VSS_LV
J10	VSS_LV_COR	VSS_LV	T15	VSS_LV_COR	VSS_LV
J11	VSS_LV_COR	VSS_LV	T16	VSS_LV_COR	VSS_LV
J12	VSS_LV_COR	VSS_LV	T17	VSS_LV_COR	VSS_LV
J13	VSS_LV_COR	VSS_LV	U7	VSS_LV_COR	VSS_LV
J14	VSS_LV_COR	VSS_LV	U8	VSS_LV_COR	VSS_LV
J15	VSS_LV_COR	VSS_LV	U9	VSS_LV_COR	VSS_LV
J16	VSS_LV_COR	VSS_LV	U10	VSS_LV_COR	VSS_LV
J17	VSS_LV_COR	VSS_LV	U11	VSS_LV_COR	VSS_LV
K7	VSS_LV_COR	VSS_LV	U12	VSS_LV_COR	VSS_LV
K8	VSS_LV_COR	VSS_LV	U13	VSS_LV_COR	VSS_LV
K9	VSS_LV_COR	VSS_LV	U14	VSS_LV_COR	VSS_LV
K10	VSS_LV_COR	VSS_LV	U15	VSS_LV_COR	VSS_LV
K11	VSS_LV_COR	VSS_LV	U16	VSS_LV_COR	VSS_LV
K12	VSS_LV_COR	VSS_LV	U17	VSS_LV_COR	VSS_LV
K13	VSS_LV_COR	VSS_LV	W4	VSS_LV_PLL	VSS_LV
K14	VSS_LV_COR	VSS_LV	AC19	VSS_HV_PMU	VSS_LV
K15	VSS_LV_COR	VSS_LV	D5	RESERVED	VSS_HV

Table 3-5. 473 MAPBGA supply pins (continued)

Ball number	Ball name	Pad type	Ball number	Ball name	Pad type
K16	VSS_LV_COR	VSS_LV	AB20	RESERVED	VSS_HV
K17	VSS_LV_COR	VSS_LV			

Table 3-6. 473 MAPBGA pins not populated on package

E5	E6	E7	E8	E9	E10	E11	E12
E13	E14	E15	E16	E17	E18	E19	F5
F19	G5	G19	H5	H19	J5	J19	K5
K19	L5	L19	M5	M19	N5	N19	P5
P19	R5	R19	T5	T19	U5	U19	V5
V19	W5	W6	W7	W8	W9	W10	W11
W12	W13	W14	W15	W16	W17	W18	W19

3.2.3 System pins

Table 3-7 and Table 3-8 show the system pins for the MPC5675K microcontrollers in the 257 MAPBGA and the 473 MAPBGA packages, respectively.

Table 3-7. 257 MAPBGA system pins

Ball number	Ball name	Weak pull during reset	Safe mode default condition	Pad type	Power domain
C4	FCCU_F[1]	disabled	not available	GP Slow/Medium	VDD_HV_IO
C10	JCOMP	pulldown	not available	GP Slow	VDD_HV_IO
E1	Nexus MDO[0][1]	—	not available	GP Slow/Fast	VDD_HV_IO
E4	NMI	pullup	not available	GP Slow	VDD_HV_IO
L15	TCK[2]	pullup	not available	GP Slow	VDD_HV_IO
M16	TMS	pullup	not available	GP Slow	VDD_HV_IO
N1	XTALIN	—	not available	Analog Feedthrough	VDD_HV_IO
P2	$\overline{\text{RESET}}$	pulldown	not available	Reset	VDD_HV_IO
R1	XTALOUT	—	not available	Analog Feedthrough	VDD_HV_IO
R2	FCCU_F[0]	disabled	not available	GP Slow/Medium	VDD_HV_IO
R13	VREG_CTRL	—	—	Analog Feedthrough	VDD_REG
U12	VREG_INT_ENABLE	—	—	Analog Feedthrough	VDD_HV_IO
U13	$\overline{\text{RESET\_SUP}}$	pulldown	—	Analog Feedthrough	VDD_HV_IO

1 Do not connect pin directly to a power supply or ground.

2 If LBIST is enabled, an external pull between 1K and 100K ohm must be connected from TCK to either power or ground to avoid LBIST failures.

Table 3-8. 473 MAPBGA system pins

Ball number	Ball name	Weak pull during reset	Safe mode default condition	Pad type	Power domain
C4	FCCU_F[1]	disabled	not available	GP Slow/Medium	VDD_HV_IO
D10	JCOMP	pulldown	not available	GP Slow	VDD_HV_IO
E1	Nexus MDO[0][1]	—	not available	GP Slow/Fast	VDD_HV_IO
E4	NMI	pullup	not available	GP Slow	VDD_HV_IO
R23[2]	dramc CLKB	—	—	DRAM CLK	VDD_HV_DRAM
T23[2]	dramc CLK	disabled	—	DRAM CLK	VDD_HV_DRAM
W1	XTALIN	—	not available	Analog Feedthrough	VDD_HV_IO
Y2	$\overline{\text{RESET}}$	pulldown	not available	Reset	VDD_HV_IO
Y19	TCK[3]	pullup	not available	GP Slow	VDD_HV_IO
AA1	XTALOUT	—	not available	Analog Feedthrough	VDD_HV_IO
AA2	FCCU_F[0]	disabled	not available	GP Slow/Medium	VDD_HV_IO
AB19	TMS	pullup	not available	GP Slow	VDD_HV_IO
AC18	VREG_CTRL	—	—	Analog Feedthrough	VDD_REG
AC20	$\overline{\text{RESET\_SUP}}$	pulldown	—	Analog Feedthrough	VDD_HV_IO
AC21	VREG_INT_ENABLE	—	—	Analog Feedthrough	VDD_HV_IO

1 Do not connect pin directly to a power supply or ground.

2 PCR234 can be used to control the slew rate of DRAM CLK and DRAM CLKB. See the “System Integration Unit Lite” chapter of the MPC5675K reference manual.

3 If LBIST is enabled, an external pull between 1K and 100K ohm must be connected from TCK to either power or ground to avoid LBIST failures.

This page is intentionally left blank.

Chapter 4
Functional Safety

4.1 Overview

The MPC5675K microcontroller offers a set of features to support using it for applications that require high integrity to fulfill functional safety requirements. For more information, please see the MPC5675K Safety Manual (SM).

4.2 Redundancy

The main approach used to achieve functional safety is redundancy. Redundancy is applied in different ways for different modules of this microcontroller:

- Processing cores: When used for a safety critical application, the two redundant cores are used preferably in Lock Step Mode (LSM). Only within this mode any difference between the outputs of the cores indicates a fault and triggers the corresponding reaction to prevent propagation of the fault and to put the microcontroller into a Fail-Safe mode.
- Replicated peripherals, if safety critical for the application, preferably are used in a redundant way by the application software.
- Non-replicated input peripherals, if safety critical for the application and not self-tested, have to be supported by system level safety measures, for example, by the application software reading them multiple times (time redundancy).
- Non-replicated output peripherals, if safety critical for the application and not read-back, have to be supported by system level safety measures, for example, by the application software writing them multiple times (time redundancy).

4.3 Sphere of Replication (SoR)

Figure 4-1 shows the Sphere of Replication (SoR) and its building blocks on different hierarchy levels.

The two parts of the SoR are referred to as Sphere Part 0 and Sphere Part 1. Each module instantiated in Sphere Part 0 is replicated in Sphere Part 1. In other words, Sphere Part 1 is the replica of Sphere Part 0.

Respective to ISO 26262 the Sphere Part 0 and Sphere Part 1 are redundant elements. In the light of IEC 61508 each Sphere Part is a channel. A channel is an element or group of elements that perform(s) a function. The two parts of the SoR can be seen as two independent processing channels.

IP instantiations that are part of Sphere Part 0 are referenced with the suffix “_0”. IP instantiations that are part of Sphere Part 1 are referenced with the suffix “_1”.

All IP modules from Sphere Part 0 except for its RCCUs are part of Lake 0. All IP modules from Sphere Part 1 except for its RCCUs are part of Lake 1. IP modules in Lake 0 must physically be separated from the IP modules in Lake 1. The RCCUs check qualified outputs of Lake 0 and Lake 1 in each SoR clock cycle.

See Section 4.3.2, Sphere of Replication (SoR), for a list of IP modules belonging to the SoR.

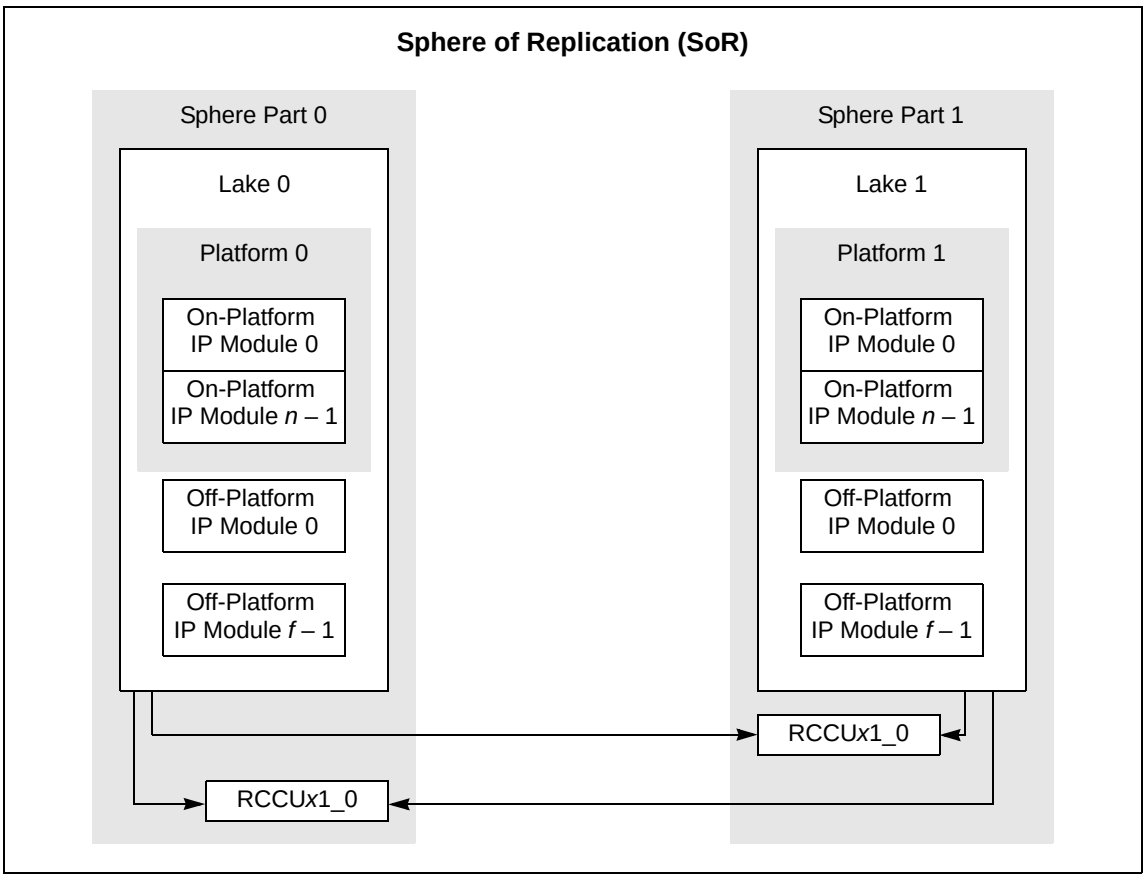


Figure 4-1. SoR Definition

4.3.1 Input synchronization

As the RC checkers are checking the outputs of both parts of the SoR for equality on every clock cycle, it is important that the input signals provided to each part of the SoR are identical at every clock cycle of the SoR. This is especially important if clock domain synchronization is required between signal source(s) outside of the SoR and the SoR inputs. Some examples here are all interrupt signals routed to INTC_0 and INTC_1 potentially coming from a different clock domain to the one of the INTC_0 and INTC_1. SWT_0 and SWT_1 or all motor control related peripherals behind the IPSYNC bridge are just two examples.

Figure 4-2 shows the SoR input signal synchronization required for all signals originating from a clock source asynchronous to the clock of the SoR.

Asynchronous interrupts coming from peripherals need to be synchronized in a lockstep-compatible way outside of the SoR.

Special attention needs to be paid to this requirement as the DPM requires some of the signals entering the two parts of the SoR to be controlled separately (one example is the reset input for both cores in DPM), which is not the case in LSM. The reset for both parts of the SoR is one example.

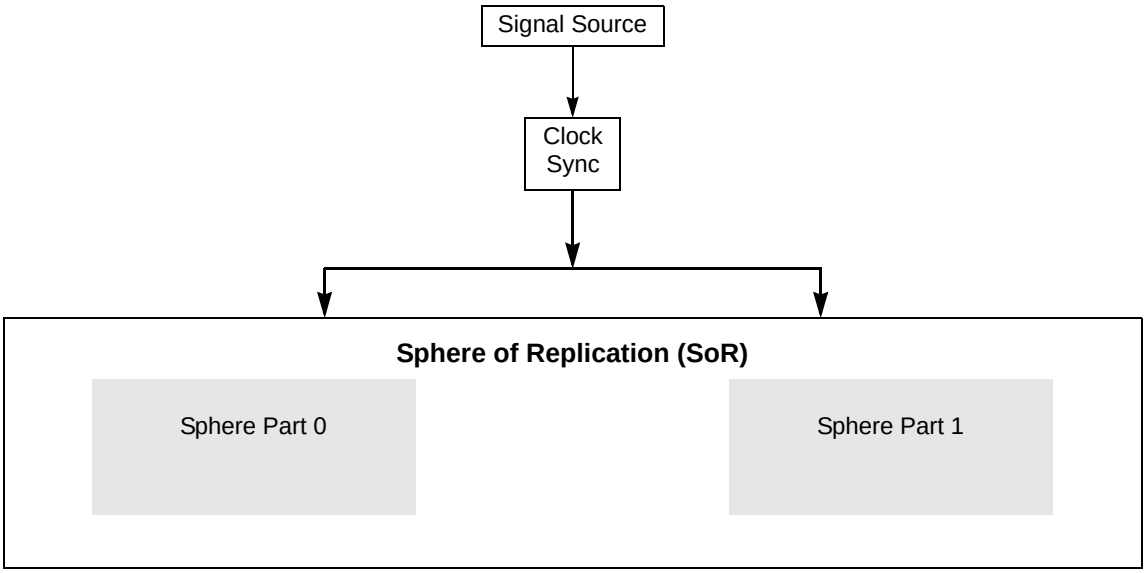


Figure 4-2. SoR input synchronization

4.3.2 Sphere of Replication (SoR)

In LSM, the Sphere of Replication represents one channel. However, the two Lakes of the SoR are physically separated on the die to minimize undetectable common cause faults inside the SoR.

In LSM, the two replicas are configured in a symmetrical way to ensure lockstep operation. In DPM, the two replicas are configured in a slightly different way to allow independent operation. More details are described in the subsequent sections.

Each replica (each part of the SoR, Sphere Part 0 and Sphere Part 1) must keep separated ports for common signals like clock, mode control, reset, and debug. These signals are routed separately to each SoR.

4.4 Built-In Self-Test (BIST)

BIST is a mechanism that permits a microcontroller to test itself. On the MPC5675K, BIST enables the reliability requirements necessary to achieve SIL3 and ISO26262.

4.4.1 BIST during boot

A device BIST is performed every time the microcontroller is reset desctructively (poweron reset). The BIST is performed transparently for the application while the microcontroller is still under reset. In case the BIST fails and the optional Stay in Reset condition is programmed into shadow flash, the microcontroller will be kept under reset. Application software can only start to run when BIST finishes successfully without detecting a fault in this configuration. The boot time BIST comprises:

- Memory BIST for all RAMs and ROM
- Scan-based Logic BIST for digital logic, partitioned with a separate BIST for each Sphere of Replication (SoR), and a partition for the remaining non-replicated peripherals

4.4.2 Software-triggered BIST during operation

For some modules of this microcontroller it is required to run BIST during operation, because testing only during destructive reset (poweron reset) is not sufficient for the targeted safety integrity level. These BIST runs need to be triggered by application software once within each process safety time.

Modules that require BIST runs during application are:

- ADC, in which software triggers several self-tests implemented in hardware
- Flash memory, for ECC logic test

4.4.3 Software-triggered self-tests after boot

In order to ensure integrity of flash memory for a safety application, hardware-based flash self-test needs to be triggered by software every time the microcontroller is destructively reset (poweron reset).

4.5 Memory error detection and correction

RAMs are protected against soft errors by an 8-bit/64 SECDED (single error correction, double error detection) ECC HSIAO code computed over address and data at each memory access. Detected faults are reported to the FCCU.

NVM flash memory is protected by an 8-bit/64 SECDED ECC HSIAO code.

4.6 Monitoring

All monitoring features react within 10 ms or faster.

Core voltage is equipped with a low voltage detector and a high voltage detector to indicate if voltage is out of the specified range. I/O voltage is monitored by a low voltage detector. The voltage detectors themselves have a hardware based self-checking feature.

This microcontroller is equipped with one temperature sensor.

Three clocks are monitored:

- Internal oscillator clock
- PLL-generated system clock
- FlexRay clock

The MPU prevents incorrect memory accesses based on 16 possibly overlapping physical memory regions.

4.7 Software measures

Several software measures are recommended to achieve functional safety integrity for this microcontroller. Software has to trigger these test features at least once within the process safety time or fault tolerant time interval. These are simple checks. No software-based core self-test routine library is mandatory for this microcontroller.

The following shows some examples for software checks:

Example 4-1. Modules that require CRC checks during operation

- SIUL: System configuration registers checked
- ADC: ADC configuration registers checked
- eTimer: Timer configuration registers checked

Example 4-2. Checks that are required by the CTU

- Have all triggers been generated and served (supported by hardware, faults to be handled in software)?
- Do trigger times match expected behavior?
- Is there a trigger buffer overrun (supported by hardware, faults to be handled in software)?
- Does the channel number sequence match expected behavior?
- Are the issued commands valid (supported by hardware, faults to be handled in software)?

4.8 Fault reaction

All faults detected by hardware measures like the redundancy checkers, self-test features, ECC, voltage and clock monitors are reported to the central Fault Collection and Control Unit (FCCU). Depending on the particular fault, the FCCU puts the microcontroller into the corresponding configured fail-safe state. This prevents fault propagation to system level. The following states are considered as fail-safe:

- Shut-down state
- Safety critical digital output in high impedance state (tri-state)
- Reset state
- Indication of internal faults on “Error Out” signals to the surrounding system

4.9 External measures

This microcontroller requires several external measures to support safe operation:

- External power supply and monitor
- External timeout circuitry (for example, watchdog)

This page is intentionally left blank.

Chapter 5
Boot and Operating Modes

5.1 Boot modes

The MPC5675K supports the following boot modes:

- Single Chip (SC) mode—the microcontroller boots from the first bootable section of the flash memory main array.
- Serial Boot Loader (SBL) mode—the microcontroller downloads boot code from either the LINFlexD or FlexCAN interface and then executes it. Serial boot can be performed with or without autobaud.

If booting is not possible with the selected configuration (for example, if no Boot ID is found in the selected boot location) then the microcontroller enters Static mode (see Section 5.4.5, Static mode).

5.2 Hardware configuration

The microcontroller detects the boot mode based on external pins and device status. The following sequence applies:

- To boot from LINFlexD or FlexCAN, the microcontroller must be forced into Alternate Boot Loader mode via the FAB (Force Alternate Boot mode) pin, which must be asserted before initiating the reset sequence. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see Table 5-1). For details of the serial boot modes, refer to Chapter 12, Boot Assist Module (BAM).
- If the microcontroller identifies a flash memory sector with a valid boot sector, it boots from the lowest sector (see Figure 5-1).
- If none of the flash memory sectors contains a valid boot signature, the microcontroller goes into Safe mode.

Table 5-1. Hardware configuration

FAB	ABS2	ABS0	Boot ID	Boot mode
1	0	0	—	Serial boot LINFlexD
1	0	1	—	Serial boot FlexCAN
1	1	0	—	Serial boot via LINFlexD or FlexCAN in autobaud
0	—	—	valid	SC (single chip)
0	—	—	not found	Safe mode

5.3 Boot-sector search

5.3.1 Potential boot sectors

As shown in Figure 5-1, in Single Chip mode the microcontroller searches several locations for a valid boot ID. The lowest sector that starts with a valid boot ID is used to boot the microcontroller.

The flash memory locations listed in Table 5-2 are searched.

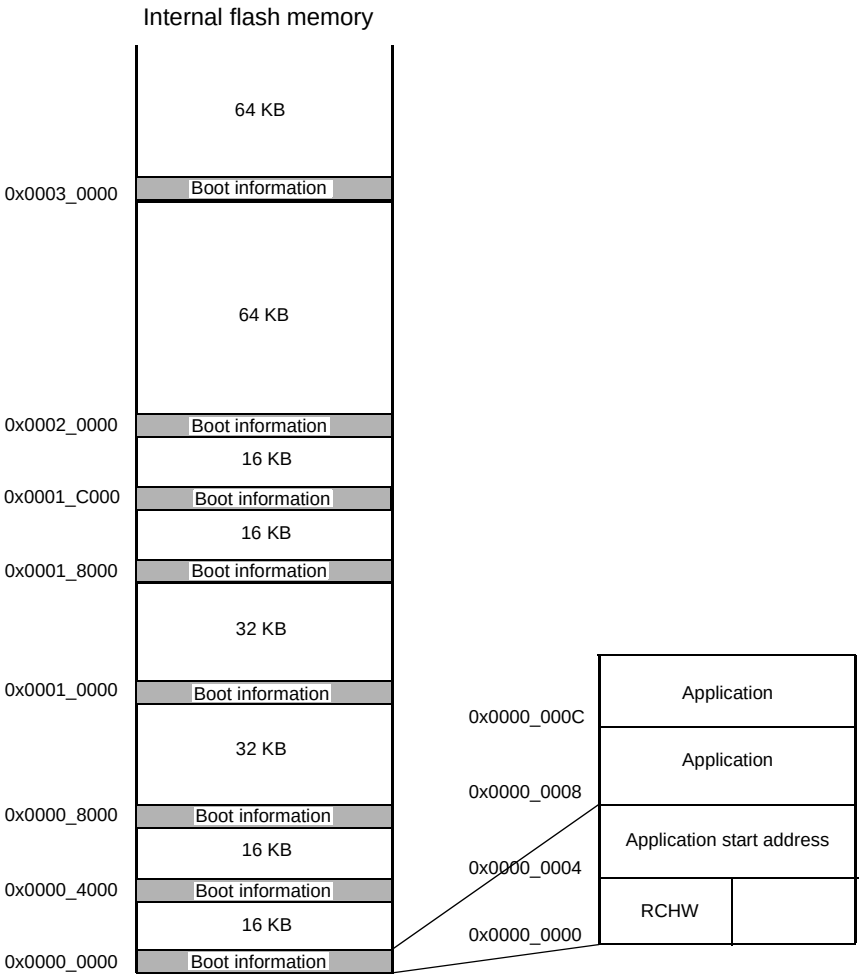


Figure 5-1. Flash memory partitioning and RCHW search

Table 5-2. Flash boot sector locations

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0000_8000
3	0x0001_0000
4	0x0001_8000
5	0x0001_C000
6	0x0002_0000
7	0x0003_0000

5.3.2 Reset configuration halfword (RCHW)

Each boot sector in the flash memory contains the Reset configuration halfword (RCHW) at offset 0x00. If the RCHW field BOOT_ID holds the value 0x5A, then the sector is considered bootable. In addition, the VLE flag indicates that the code is VLE code. All other bits are reserved.

No BAM code execution occurs in this case, and BAM code is not visible unless the microcontroller is in Serial Boot Loader (SBL) mode.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	VLE	BOOT_ID							
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-2. Reset configuration halfword (RCHW)

Once the microcontroller detects that it needs to boot from flash memory, and finds a valid BOOT_ID, it boots from the application start address at offset 0x04 within the boot sector.

5.3.3 Boot and alternate boot

Some applications require an alternate boot sector so that the main boot can be erased and reprogrammed in the field.

When an alternate boot is needed, the user can create two bootable sectors. The lowest sector is the main boot sector, and the highest is the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This scheme ensures that there is always one active boot sector by erasing one of the boot sectors only:

- Sector is activated (that is, program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector is deactivated writing to 0 some of the bits BOOT-ID bit field (bit 1 and/or bit 3, and/or bit 4, and/or bit 6).

5.4 Device behavior by boot mode

The following describes the security related device features that are available in the device boot modes.

5.4.1 Single-chip mode—unsecured

In normal Single-Chip mode, the system boots from the flash memory main array. The microcontroller boots from the first sector that is marked bootable. The system may be configured to enable the external bus:

- Nexus/JTAG available
- Boot from flash memory main array
  - — Shadow block available

5.4.2 Single-chip mode—secured

In secured Single-Chip mode, the system boots from the flash memory main array. The microcontroller boots from the first sector that is marked bootable:

- No Nexus/JTAG apart from JTAG information for device and mask ID (major and minor) and commands to temporarily unlock the microcontroller with a valid flash memory password.
- Boot from flash memory main array.
  - Shadow block accessible

5.4.3 Serial boot loader mode—public password enabled

In Serial Boot Loader (SBL) mode, if public serial access is allowed, the MCU executes BAM code that checks for a valid public password on the chosen interface. The interface is selected via FAB and ABS pins:

- Nexus/JTAG available
- Boot from BAM
- Disable flash memory main array and shadow block access

5.4.4 Serial boot loader mode—flash memory password enabled

It is possible to boot in SBL mode when “serial access with password” is selected. The MCU executes BAM code that checks for a valid flash memory password on the chosen interface. If the password is known, full access to the device is enabled:

- Nexus/JTAG available
- Boot from BAM

5.4.5 Static mode

Static mode means the device enters the low power Safe mode and the processor executes a wait instruction. This is needed if the device is unable to boot in the selected mode. Access to Nexus and flash memory is as defined by the boot mode and the security status. A power-on reset is required to bring the device out of Static mode.

5.5 Operating modes

MPC5675K microcontrollers can operate in two modes of operation:

- Lock Step mode (LSM)
- Decoupled Parallel mode (DPM)

One of the two modes is statically selected at startup. The selected mode may be changed only going through a full power-on reset.

Each of these modes has several specific submodes that the microcontroller can enter. These modes differ, for example, in the list of enabled modules, pin configurations, reset phase, and safety status. These are covered in Section 35.1, Introduction.

5.5.1 Lock Step Mode (LSM)

This mode takes its name from the execution of the same commands by both cores in synchronicity (lock step). It has been implemented to support high levels of functional safety integrity (for example, SIL3 or ASIL D) with minimal software overhead.

In this mode, the Sphere of Replication (SoR) plays a major role. It contains all hardware elements that have been replicated for safety reasons, resulting in the sphere being a collection of pairs. Each member of such a pair executes the same operations or transactions as its partner, resulting in lock step behavior. The compliance with this behavior expectation is checked only on the boundary of the SoR, minimizing checker effort.

This boundary check is based on a modified version of the fault isolation concept. Fault isolation requires that a fault must not cause failures outside a marked area, in this case the SoR. A failure in the SoR, as long as it does not propagate to the outside of the SoR and potentially cause a fault there, does not influence the effective operability of the periphery (and so the ECU). Thus it cannot cause a hazard.

For example, an error in the ALU can cause wrong calculation results but as long as these results only influence core register values, they are not a hazard to the operation of the system. Also, propagation inside the SoR is of no immediate consequence. For example, if the wrong register value is written to the INTC, this — in itself — does not change the overall behavior of the system. But once the registers are written somewhere external or used as addresses, or once the badly changed interrupt triggers, this “safeness” changes because the failure now propagates to the outside of the SoR.

The Redundancy Control Checker Units (RCCU) at the outputs of the SoR to the periphery bus, to the flash memory subsystem, and to the SRAM subsystem detect such propagating failures due to data on the external busses being inconsistent between both processing units. Thus the RCCUs implement the modified fault isolation in that they detect but not prevent the propagation of a non-common cause failure at the point where the two redundant channels are merged into a single actuator or recipient. Isolation of the overall system is then achieved by the Fault Collection and Control Unit (FCCU) signaling an error, thereby allowing the microcontroller or application to react appropriately.

5.5.2 Decoupled Parallel mode (DPM)

In this mode, each CPU core runs independently. This mode of operation puts the chip into a symmetric multi-core processor mode. When in this mode, the redundancy checkers are not enabled, and the replicated peripherals are available at a different set of addresses. The SRAM is split in half and relocated in the memory map (See Chapter 2, Memory Map, for details). In DMP mode, the hardware Semaphore module becomes available. Reciprocal comparison of data caclulated independently on both CPUs may be required by software to achieve respective functional safety integrity.

The Decoupled Parallel mode (DPM) increased performances can be estimated in a first approximation as about 1.6× the performance of the Lock Step mode (LSM) at the same frequency.

The selection between the Lock Step mode and the Decoupled Parallel mode is done via a user option programmed into the shadow block of the flash. See Chapter 8, Flash Memory, for more details.

MPC5675K microcontrollers support only static configuration at power-on (either LSM or DPM).

5.6 Selecting LSM or DPM

The operating mode (LSM or DPM) on MPC5675K is determined by the LSM_DPM user option bit in the shadow sector of the flash memory. This user option bit is accessed using the UOPS[UOPT] field in the SSCM (see Section 49.3.1.8, User Option Status (SSCM_UOPS) Register).

NOTE

The external reset sequence should always be configured to start from PHASE1 by setting RGM_FESS[SS_EXR] = 0. See Section 46.3.1.6, Functional Event Short Sequence Register (RGM_FESS).

The following steps summarize the MPC5675K behavior in DPM following an external reset:

1. External reset is triggered.
2. Both Core_0 and Core_1 undergo reset.
3. Core_0 starts fetching its first instruction, while Core_1 stays under reset until step 4.
4. After some basic configuration, Core_0 wakes Core_1 up through the SSCM module.
5. Both Core_0 and Core_1 work independently.

If the external reset sequence starts from PHASE3 (RGM_FESS[SS_EXR] = 1), Core_0 and Core_1 go under reset at the same time, but Core_1 starts to run immediately without waiting for any signal from Core_0. This could cause unexpected results to occur if the Core_1 startup code does not take short reset into account. Core_1 code can check the last reset source using RGM flags. If it finds that the reset was due to short reset, then it can poll for a Core_0 signal to wait before executing further code.

5.6.1 Entering DPM

By default, MPC5675K is configured to start in DPM (LSM_DPM = 0).

5.6.2 Entering LSM

5.6.2.1 Dual-core boot concepts

Entering DPM implies a dual-core boot. The key concept to a dual-core boot is that it is nothing more than a typical single-core boot, except that it starts another single-core boot. The initialization of interrupts, stack, and other parameters needs to be performed on each core. In other words, it is a single-core boot performed twice.

Figure 5-3 shows a simplification of this boot process.

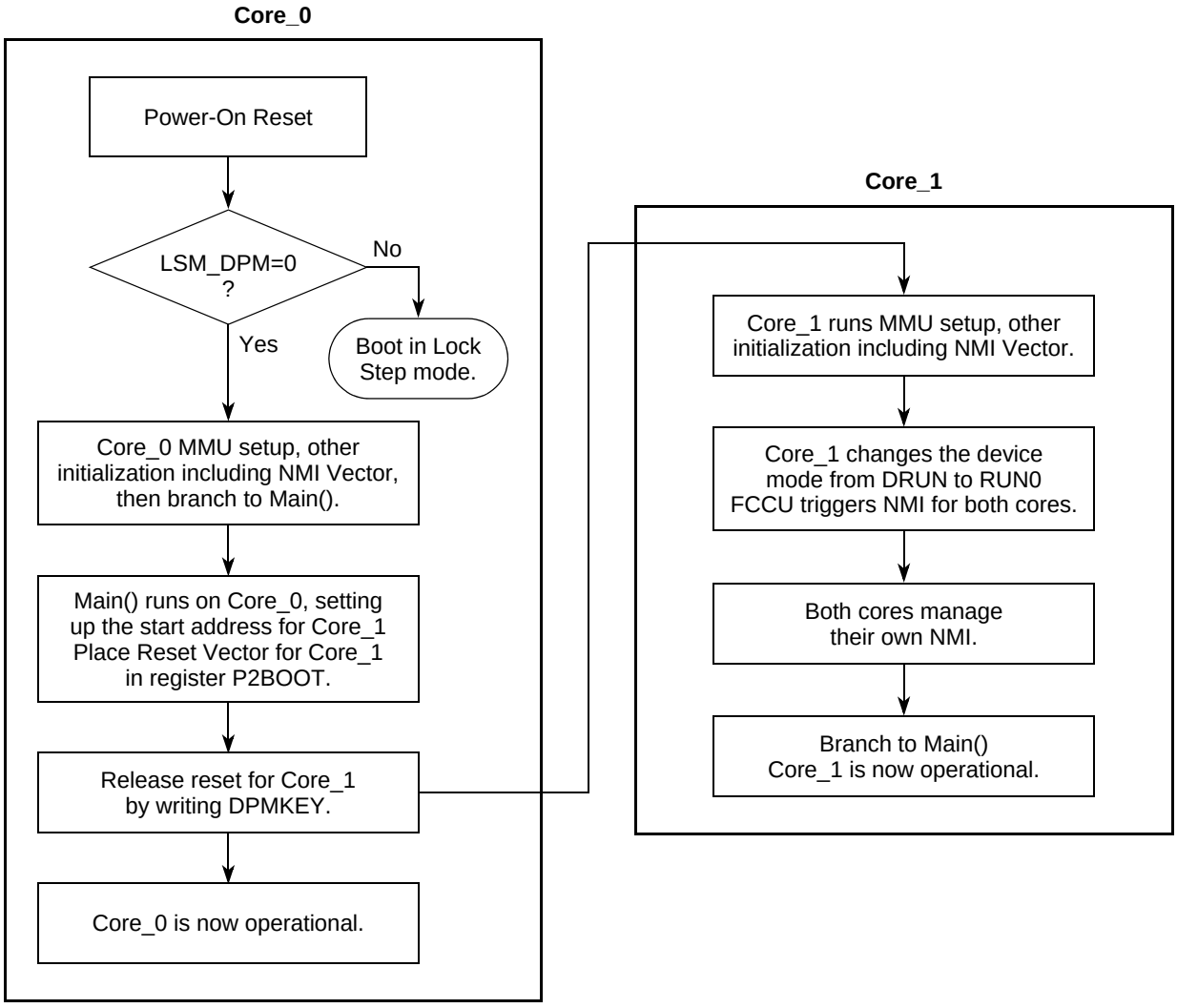


Figure 5-3. Simplified boot process in DPM

At power-on reset (POR), Core_0 begins operation while Core_1 remains held in reset. At this time, Core_0 must initialize its set of peripherals, set up its environment (including the NMI routine), then branch to main. At this point, Core_0 is essentially fully operational. Now Core_0 provides the reset vector and writes the DMPKEY, thus releasing Core_1 from reset.

Core_1 begins its execution. The first thing that it must do is initialize its set of peripherals and set up its environment, including its NMI routine. Core_1 then moves the chip from DRUN mode to RUN0 mode. The system generates an NMI to both cores when the chip moves from DRUN to RUN0. Each core must service its own NMI routines.

5.6.2.2 Software setup

During the boot sequence, this dual core architecture is set up with one core being the master and the other core designated as slave. That is to say, the primary core, Core_0, is run from reset and executes code, which then sets up and releases reset to Core_1. At that time, the system then begins operating in DPM.

To make the second core operational, you must configure the following two registers in the SSCM:

- DPMBOOT (see Section 49.3.1.6, DPM Boot Register (SSCM_DPMBOOT))
- DPMKEY (see Section 49.3.1.7, DPM Boot Key Register (SSCM_DPMKEY))

Follow this sequence to enable DPM:

1. Write the reset vector into DPMBOOT[P2BOOT].
2. Set DPMBOOT[DVLE] to indicate that the second core will be executing in VLE mode. (Otherwise, the core will operate in Book E mode.)
3. Write 0x5AF0 to DPMKEY[KEY].
4. Write 0xA50F to DPMKEY[KEY].

After the second write to DPMKEY[KEY], Core_1 jumps to its reset vector.

Chapter 6
e200z7 Core

6.1 e200z7 Overview

The e200z7 is a dual-issue, 32-bit Power Architecture Book E-compliant design with 64-bit general purpose registers (GPRs). Power Architecture Book E floating-point instructions are not supported by e200z7 in hardware, but are trapped and may be emulated by software. For an overview of this architecture and the instruction set, see *EREF: A Programmer's Reference Manual for Freescale Embedded Processors (EREFRM)*.

An Embedded Floating-point (EFPU2) APU is provided to support real-time single-precision embedded numerics operations using the general-purpose registers.

A Signal Processing Extension (SPE2.1) APU is provided to support real-time SIMD fixed point and single-precision, embedded numerics operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). The GPRs have been extended to 64-bits in order to support vector instructions defined by the SPE2.1 APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results. For more information, see *Signal Processing Engine (SPE) Programming Environments Manual: A Supplement to the EREF (SPEPEM)*.

In addition to the base Power Architecture Book E instruction set support, the e200z7 core also implements the VLE (variable-length encoding) technology, providing improved code density. The VLE technology is further documented in *Variable-Length Encoding (VLE) Programming Environments Manual: A Supplement to the EREF (VLEPEM)*, a separate document.

The e200z7 processor integrates a pair of integer execution units, a branch control unit, instruction fetch unit and load/store unit, and a multi-ported register file capable of sustaining six read and three write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in many cases.

The e200z7 contains a 16 KB instruction cache, a 16 KB data cache, as well as a memory management unit (MMU). A Nexus Class 3+ module is also integrated. See Chapter 39, Nexus Development Interface (NDI).

Figure 6-1 shows a high-level block diagram of the e200z7 core.

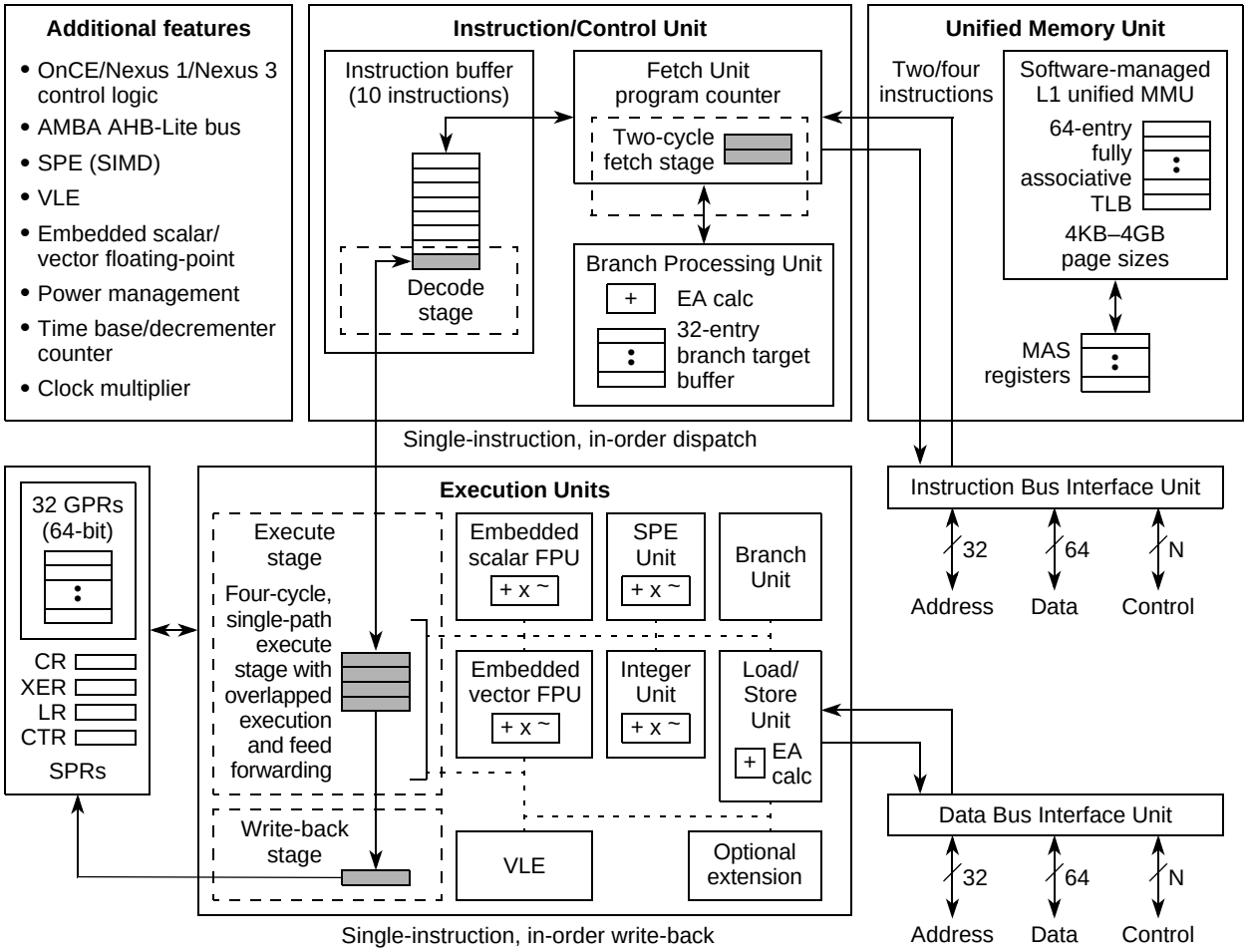


Figure 6-1. e200z7 block diagram

6.1.1 Features

The following is a list of some of the key features of the e200z7:

- Dual issue, 32-bit Power Architecture Book E-compliant CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - — Dedicated branch address calculation adder
  - — Branch target prefetching using BTB
  - — Return address stack
- Load/store unit (LSU)
  - — Three-cycle load latency

  - Fully pipelined
  - Big- and little-endian support on a per-page basis
  - Misaligned access support
  - AMBA (advanced microcontroller bus architecture) AHB-Lite (advanced high-performance bus) 64-bit system bus
- Memory management unit (MMU) with 64-entry, fully associative TLB and multiple page-size support
- 16 KB, 4-way set-associative Harvard instruction and data caches
- SPE unit supporting SIMD fixed-point and single-precision floating-point operations, using the 64-bit GPR file
- Embedded floating-point unit (EFPU) supporting scalar single-precision floating-point operations
- Performance management unit (PMU) supporting execution profiling
- Nexus Class 3+ real-time development unit
- Power management
  - Low-power design—extensive clock gating
  - Power-saving modes: doze, nap, sleep, and wait
  - Dynamic power management of execution units, caches, and MMUs
- e200z7-specific debug interrupt
- Testability
  - Synthesizeable, full MuxD scan design
  - Built-in parallel signature unit

6.1.2 Microarchitecture summary

The e200z7 processor utilizes a ten-stage instruction pipeline, with four stages for execution. These stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions:

- Instruction fetch 0
- Instruction fetch 1
- Instruction fetch 2
- Instruction decode 0
- Instruction decode 1
- Register file read / EA calc
- Execute 0 / memory access 0
- Execute 1 / memory access 1
- Execute 2 / memory access 2
- Execute 3
- Register writeback

The integer execution units each consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register Manipulation Unit (CRU), a

Count-Leading-Zeros Unit (CLZ), a 32 × 32 hardware ultiplier array, and result feed-forward hardware. Integer EU1 also supports hardware division.

Most arithmetic and logical operations are executed in a single cycle with the exception of multiply, which is implemented with a pipelined hardware array, and the divide instructions. A Count-Leading-Zeros Unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and dedicated Branch Address adders to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer.

Branch target addresses are calculated in parallel with branch instruction decode, resulting in an execution time of four clocks for correctly predicted branches. Conditional branches that are not taken execute in a single clock. Branches with successful BTB target prefetching have an effective execution time of one clock if correctly predicted.

Memory load and store operations are provided for byte, halfword, word (32-bit), and doubleword data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized.

The CRU supports the Condition Register (CR) and condition register operations defined by the Power Architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The SPE2.1 APU supports vector instructions operating on 8-, 16- and 32-bit fixed-point data types. The EFPU2 APU supports 32-bit IEEE-754 single-precision floating-point formats, and supports scalar and vector single-precision floating-point operations in a pipelined fashion. The 64-bit general purpose register file is used for source and destination operands, and there is a unified storage model for scalar single-precision floating-point data types of 32-bits and the normal integer type. Low latency fixed-point and floating-point add, subtract, mixed add/subtract, sum, diff, min, max, multiply, multiply-add, multiply-sub, divide, square root, compare, and conversion operations are provided, and most operations can be pipelined.

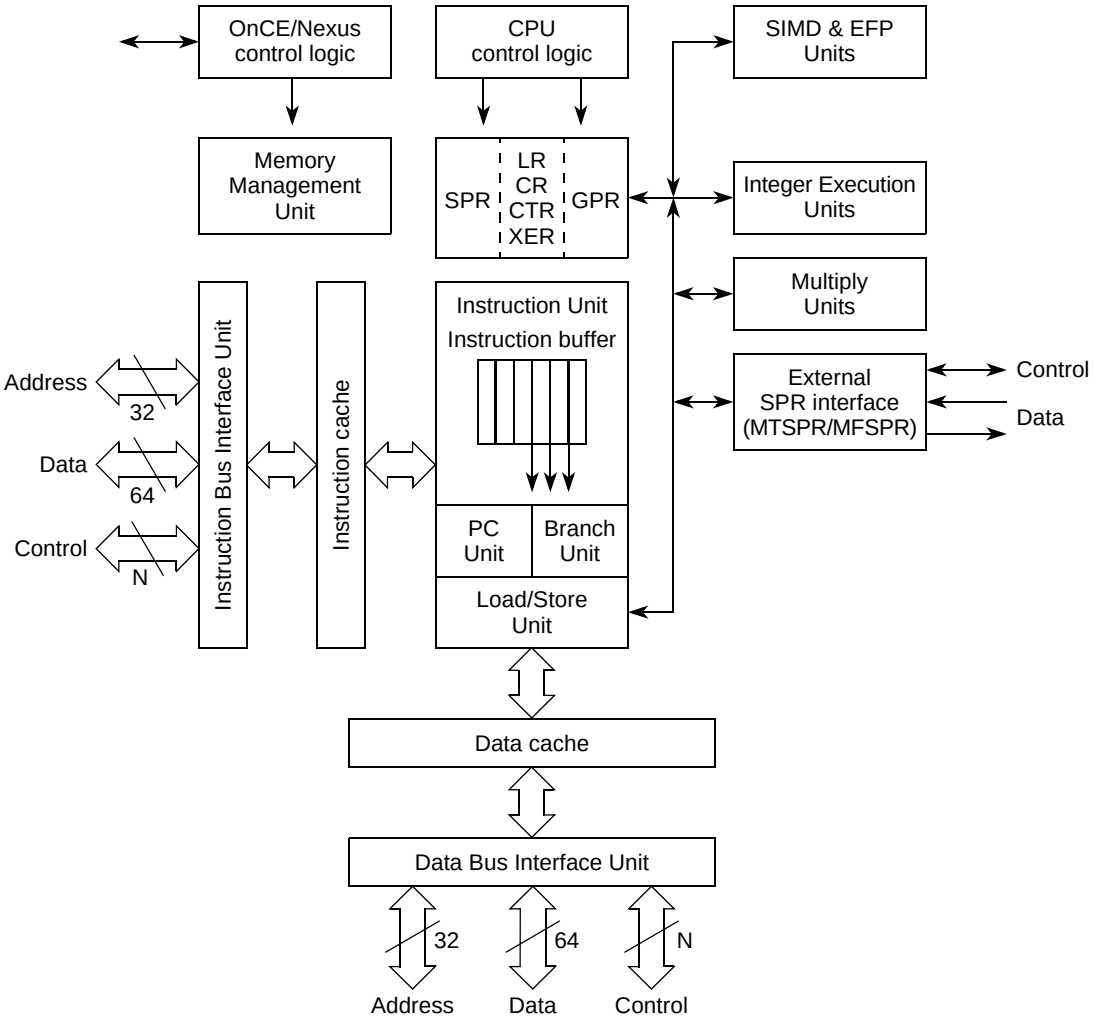


Figure 6-2. e200z7 block diagram

6.1.2.1 Instruction unit features

The features of the e200z7 instruction unit are:

- 64-bit path to cache supports fetching of two 32-bit instructions per clock
- Instruction buffer holds up to 10 32-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder, and branch lookahead logic (BTB) supporting single cycle execution of successfully predicted branches

6.1.2.2 Integer unit features

The e200z7 integer units support single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations

- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 4-15 clocks with minimized execution timing (EU1 only)
- Pipelined 32 × 32 hardware multiplier array supports 32 × 32 → 32 multiply with 3 clock latency, 1 clock throughput

6.1.2.3 Load/store unit features

The e200z7 load/store unit supports load, store, and the load multiple/store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring of up to two registers per cycle for load multiple and store multiple word instructions

6.1.2.4 Cache features

The features of the cache are as follows:

- Separate 16 KB, 4-way set-associative instruction and data caches (Harvard)
- Copyback and writethrough support
- 8-entry store buffer
- Push buffer
- Linefill buffer
- 32-bit address bus plus attributes and control
- Separate uni-directional 64-bit read data bus and 64-bit write data bus
- Cache line locking
- Way allocation
- Write allocation policies
- Tag and data parity
- Multi-bit EDC for the ICache
- Parity protection for the DCache
- Correction/auto-invalidation capability for the I and D caches
- Hardware cache coherency support for the data cache

6.1.2.5 MMU features

The features of the MMU are as follows:

- Virtual memory support
- 32-bit virtual and physical addresses

- 8-bit process identifier
- 64-entry fully associative TLB
- Multiple page size support from 1 KB to 4 GB
- Entry flush protection

6.1.2.6 e200z7 system bus features

The features of the e200z7 system bus interface are as follows:

- Independent instruction and data interfaces
- AMBA AHB2.v6 protocol
- 32-bit address bus, 64-bit data bus, plus attributes and control
- Data interface provides separate uni-directional 64-bit read and write data buses
- Support for HCLK running at a slower rate than CPU clock

6.2 Programming model

This section describes the register model, instruction model, and the interrupt model as they are defined by the Power ISA, Freescale EIS, and the e200z7 implementation.

6.2.1 Register set

The following figures show the complete e200z7 register set, including the sets of the registers that are accessible in supervisor mode and the set of registers that are accessible in user mode The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register. For example, the integer exception register (XER) is SPR 1.

Figure 6-3 and Figure 6-4 show the registers that can be accessed by supervisor-level software. User-level software can access only those registers listed in Figure 6-5 and Figure 6-6.

Figure 6-3 shows the supervisor mode SPRs and GPRs.

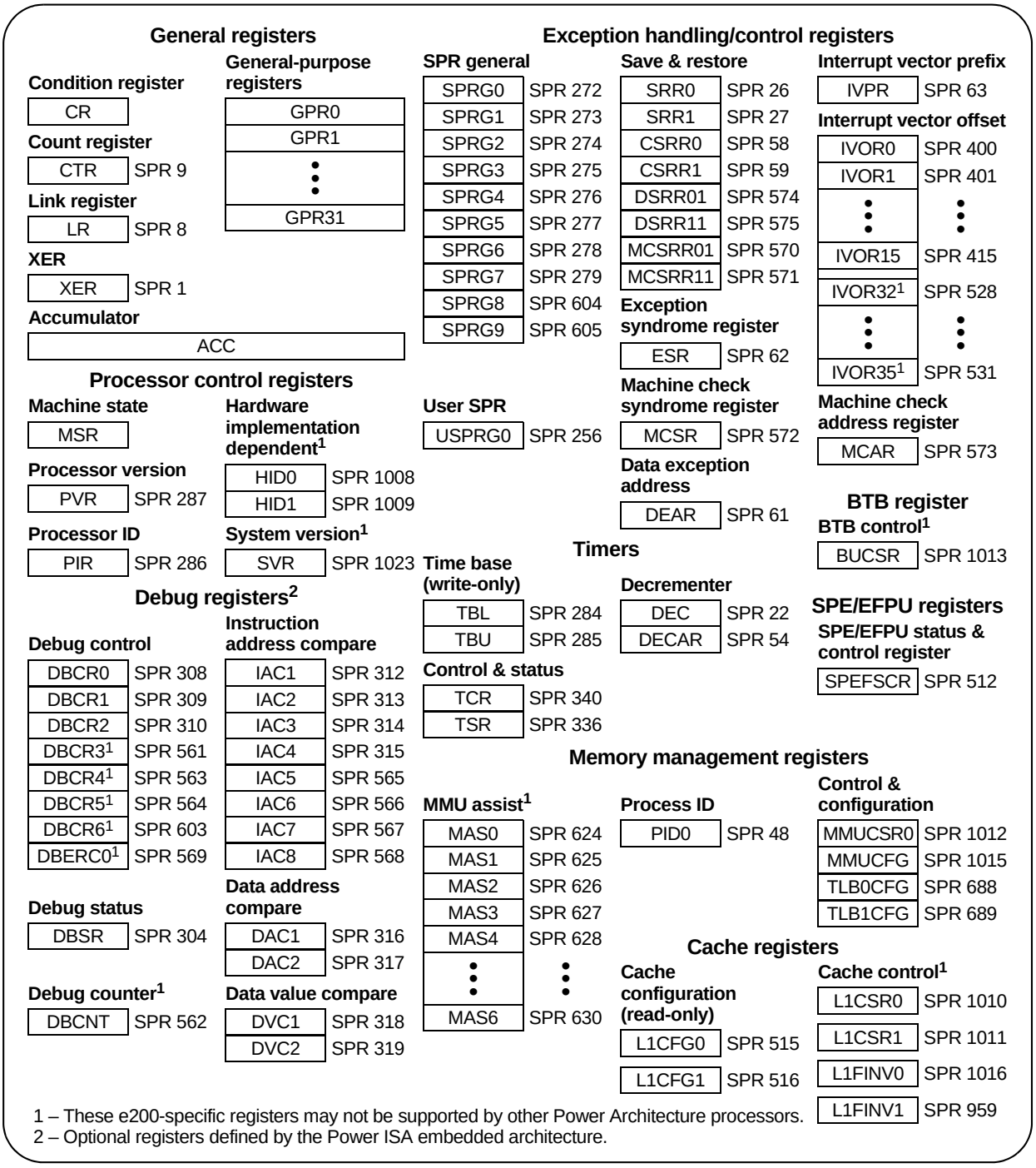


Figure 6-3. e200z760 supervisor mode programmer's model

Figure 6-4 shows the supervisor mode programmer's model DCRs and PMRs.

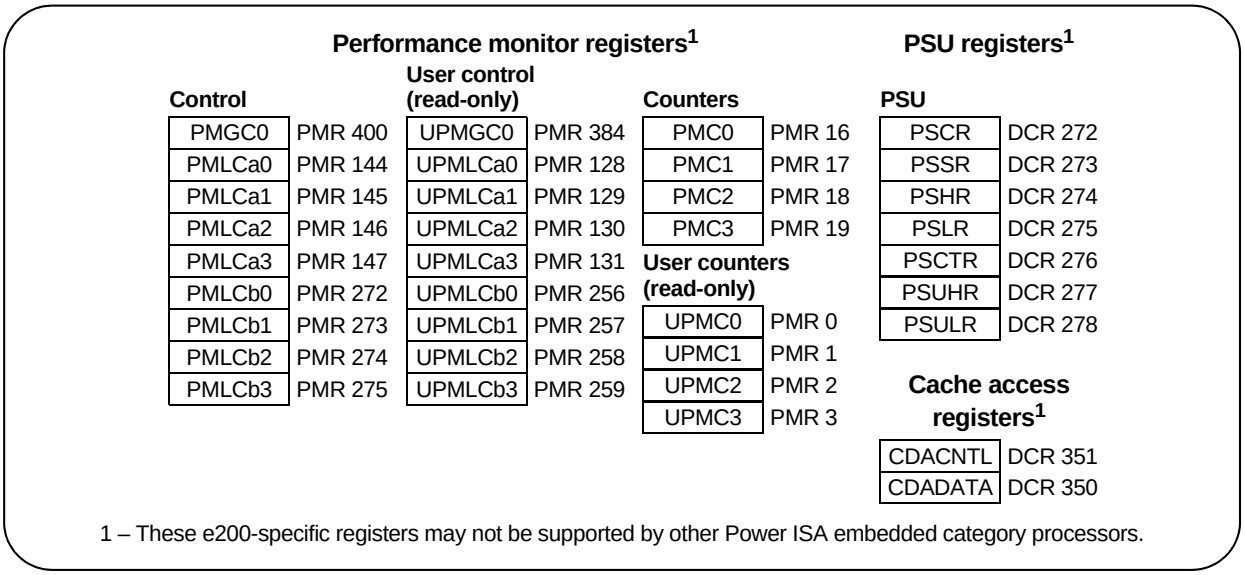


Figure 6-4. e200z760 supervisor mode programmer's model DCRs and PMRs

Figure 6-5 shows the user mode programmer's model SPRs and GPRs.

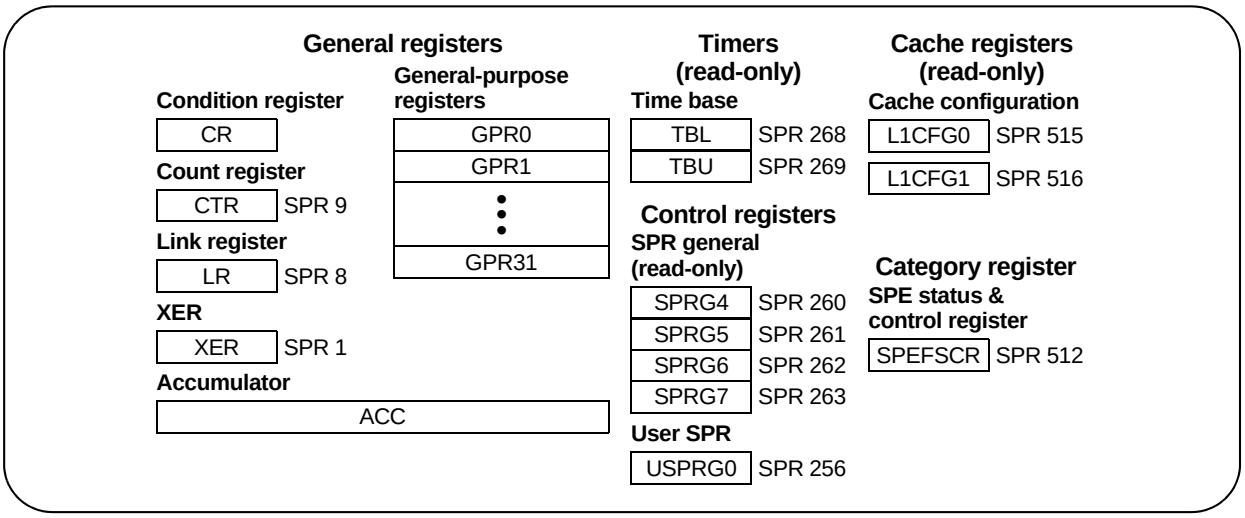


Figure 6-5. e200z7 user mode programmer's model

Figure 6-6 shows the user mode programmer's model PMRs.

Performance monitor registers[1]

User control (read-only)		User counters (read-only)	
UPMGC0	PMR 384	UPMC0	PMR 0
UPMLCa0	PMR 128	UPMC1	PMR 1
UPMLCa1	PMR 129	UPMC2	PMR 2
UPMLCa2	PMR 130	UPMC3	PMR 3
UPMLCa3	PMR 131		
UPMLCb0	PMR 256		
UPMLCb1	PMR 257		
UPMLCb2	PMR 258		
UPMLCb3	PMR 259		

1 – These e200-specific registers may not be supported by other Power ISA embedded category processors.

Figure 6-6. e200z760 user mode programmer's model PMRs

The GPRs are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as the Move To Special Purpose Register (**mtspr**) and Move From Special Purpose Register (**mfspr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

6.2.2 Instruction set

The e200z7 supports the following architectural extensions: VLE, ISEL, debug, machine check, wait, SPE, cache line locking, and enhanced reservations.

The e200z7 implements the following instructions:

- The Power ISA instruction set for 32-bit embedded implementations. This is composed primarily of the user-level instructions defined by the user instruction set architecture (UISA). The e200z7 does not include the Power ISA floating-point, load string, or store string instructions.
- The e200z7 supports the following EIS-defined instructions:
  - Integer select category. This category consists of the Integer Select instruction (**isel**), which functions as an if-then-else statement that selects between two source registers by comparison to a CR bit. This instruction eliminates conditional branches, takes fewer clock cycles than the equivalent coding, and reduces the code footprint.
  - Cache line lock and unlock category. The cache block lock and unlock category consists of the instructions described in Table 6-1, which defines a set of instructions for locking and clearing cache lines.

Table 6-1. Cache block lock and unlock instructions

Name	Mnemonic	Syntax
Data Cache Block Lock Clear	**dcblc**	CT,**r**A,**r**B
Data Cache Block Touch and Lock Set	**dcbtls**	CT,**r**A,**r**B

Table 6-1. Cache block lock and unlock instructions (continued)

Name	Mnemonic	Syntax
Data Cache Block Touch for Store and Lock Set	**dcbtstls**	CT,rA,rB
Instruction Cache Block Lock Clear	**icblc**	CT,rA,rB
Instruction Cache Block Touch and Lock Set	**icbtls**	CT,rA,rB

— Debug category. This category defines the Return from Debug Interrupt instruction (**rfdi**), which defines a separate set of interrupt save and restore registers to provide greater responsiveness for debug interrupts.

— SPE vector category. New vector instructions are defined that view the 64-bit GPRs as being composed of a vector of two 32-bit elements (some of the instructions also read or write 16-bit elements). Some scalar instructions are defined for DSP that produce a 64-bit scalar result.

— The embedded floating-point categories provide single-precision scalar and vector floating-point instructions. Scalar floating-point instructions use only the lower 32 bits of the GPRs for single-precision floating-point calculations. Table 6-2 lists embedded floating-point instructions.

— Wait category. This category consists of the **wait** instruction that allows software to cease all synchronous activity and wait for an asynchronous interrupt to occur.

— Machine check category. This feature set adds two new instructions (**rfmci**, **se_rfmci**) and four new registers (MCSRRO, MCSRR1, MCSR, MCAR)

— Volatile Context Save/Restore category supports the capability to quickly save and restore volatile register context on entry into an interrupt handler.

Table 6-2. Scalar and vector embedded floating-point instructions

Instruction	Mnemonic		Syntax
	Scalar	Vector	
Convert Floating-Point from Signed Fraction	**efscfsf**	**evfscfsf**	rD,rB
Convert Floating-Point from Signed Integer	**efscfsi**	**evfscfsi**	rD,rB
Convert Floating-Point from Unsigned Fraction	**efscfuf**	**evfscfuf**	rD,rB
Convert Floating-Point from Unsigned Integer	**efscfui**	**evfscfui**	rD,rB
Convert Floating-Point to Signed Fraction	**efsctsf**	**evfsctsf**	rD,rB
Convert Floating-Point to Signed Integer	**efsctsi**	**evfsctsi**	rD,rB
Convert Floating-Point to Signed Integer with Round Toward Zero	**efsctsiz**	**evfsctsiz**	rD,rB
Convert Floating-Point to Unsigned Fraction	**efsctuf**	**evfsctuf**	rD,rB
Convert Floating-Point to Unsigned Integer	**efsctui**	**evfsctui**	rD,rB
Convert Floating-Point to Unsigned Integer with Round Toward Zero	**efsctuiz**	**evfsctuiz**	rD,rB
Floating-Point Absolute Value	**efsabs**	**evfsabs**	rD,rA
Floating-Point Add	**efsadd**	**evfsadd**	rD,rA,rB

Table 6-2. Scalar and vector embedded floating-point instructions (continued)

Instruction	Mnemonic		Syntax
	Scalar	Vector	
Floating-Point Compare Equal	**efscmpeq**	**evfscmpeq**	**cr**D,**r**A,**r**B
Floating-Point Compare Greater Than	**efscmpgt**	**evfscmpgt**	**cr**D,**r**A,**r**B
Floating-Point Compare Less Than	**efscmplt**	**evfscmplt**	**cr**D,**r**A,**r**B
Floating-Point Divide	**efsdiv**	**evfsdiv**	**r**D,**r**A,**r**B
Floating-Point Multiply	**efsmul**	**evfsmul**	**r**D,**r**A,**r**B
Floating-Point Negate	**efsneg**	**evfsneg**	**r**D,**r**A
Floating-Point Negative Absolute Value	**efsnabs**	**evfsnabs**	**r**D,**r**A
Floating-Point Subtract	**efssub**	**evfssub**	**r**D,**r**A,**r**B
Floating-Point Test Equal	**efststeq**	**evfststeq**	**cr**D,**r**A,**r**B
Floating-Point Test Greater Than	**efststgt**	**evfststgt**	**cr**D,**r**A,**r**B
Floating-Point Test Less Than	**efststlt**	**evfststlt**	**cr**D,**r**A,**r**B
Floating-Point Single-Precision Multiply-Add	**efsmadd**	**evfsmadd**	**r**D,**r**A,**r**B
Floating-Point Single-Precision Negative Multiply-Add	**efsnmadd**	**evfsnmadd**	**r**D,**r**A,**r**B
Floating-Point Single-Precision Multiply-Subtract	**efsmsub**	**evfsmsub**	**r**D,**r**A,**r**B
Floating-Point Single-Precision Negative Multiply-Subtract	**efsnmsub**	**evfsnmsub**	**r**D,**r**A,**r**B

6.2.2.1 VLE category

This section describes the extensions to the architecture to support VLE:

- **rfci**, **rfdi**, **rfi** do not mask bit 62 of CSRR0, DSRR0, or SRR0. The destination address is [D,C]SRR0[32–62] || 0b0.
- **bclr**, **bclrl**, **bcctr**, **bcctrl** do not mask bit 62 of the LR or CTR. The destination address is [LR, CTR][32–62] || 0b0.

6.2.3 Interrupts and exception handling

The core supports an extended exception handling model, with nested interrupt capability and extensive interrupt vector programmability. The following sections define the interrupt model, including an overview of interrupt handling as implemented on the e200z7 core, a brief description of the interrupt classes, and an overview of the registers involved in the processes.

6.2.3.1 Interrupt handling

In general, interrupt processing begins with an exception that occurs due to external conditions, errors, or program execution problems. When an exception occurs, the processor checks whether interrupt processing is enabled for that particular exception. If enabled, the interrupt causes the state of the processor

to be saved in the appropriate registers and prepares to begin execution of the handler located at the associated vector address for that particular exception.

Once the handler is executing, the implementation may need to check bits in the exception syndrome register (ESR), the machine check syndrome register (MCSR), or the signal processing and embedded floating-point status and control register (SPEFSCR), depending on the exception type, to verify the specific cause of the exception and take appropriate action.

The core complex supports the interrupts described in Section 6.2.3.4, Interrupt registers.

6.2.3.2 Interrupt classes

All interrupts may be categorized as asynchronous/synchronous and critical/noncritical.

- Asynchronous interrupts (such as machine check, critical input, and external interrupts) are caused by events that are independent of instruction execution. For asynchronous interrupts, the address reported in a save/restore register is the address of the instruction that would have executed next had the asynchronous interrupt not occurred.
- Synchronous interrupts are those that are caused directly by the execution or attempted execution of instructions. Synchronous inputs are further divided into precise and imprecise types.
  - Synchronous precise interrupts are those that precisely indicate the address of the instruction causing the exception that generated the interrupt or, in some cases, the address of the immediately following instruction. The interrupt type and status bits allow determination of which of the two instructions has been addressed in the appropriate save/restore register.
  - Synchronous imprecise interrupts are those that may indicate the address of the instruction causing the exception that generated the interrupt, or some instruction after the instruction causing the interrupt. If the interrupt was caused by either the context synchronizing mechanism or the execution synchronizing mechanism, the address in the appropriate save/restore register is the address of the interrupt-forcing instruction. If the interrupt was not caused by either of those mechanisms, the address in the save/restore register is the last instruction to start execution and may not have completed. No instruction following the instruction in the save/restore register has executed.

6.2.3.3 Interrupt types

The e200z7 core processes all interrupts as either debug, machine check, critical, or noncritical types. Separate control and status register sets are provided for each type of interrupt. Table 6-3 describes the interrupt types.

Table 6-3. Interrupt types

Category	Description	Programming Resources
Noncritical interrupts	First-level interrupts that let the processor change program flow to handle conditions generated by external signals, errors, or unusual conditions arising from program execution or from programmable timer-related events. These interrupts are largely identical to those defined by the OEA.	SRR0/SRR1 SPRs and **rfi** instruction. Asynchronous noncritical interrupts can be masked by the external interrupt enable bit, MSR[EE].
Critical interrupts	Critical input, watchdog timer, and debug interrupts. These interrupts can be taken during a noncritical interrupt or during regular program flow. The critical input and watchdog timer interrupts are treated as critical interrupts. If the debug interrupt is not enabled, it is also treated as a critical interrupt.	Critical save and restore SPRs (CSRR0/CSRR1) and **rfci**. Critical input and watchdog timer critical interrupts can be masked by the critical enable bit, MSR[CE]. Debug events can be masked by the debug enable bit MSR[DE].
Machine check interrupt	Provides a separate set of resources for the machine check interrupt.	Machine check save and restore SPRs (MCSRR0/MCSRR1) and **rfmci**. Maskable with the machine check enable bit, MSR[ME]. Includes the machine check syndrome register (MCSR).
Debug interrupt	Provides a separate set of resources for the debug interrupt.	Debug save and restore SPRs (DSRR0/DSRR1) and **rfdi**. Can be masked by the machine check enable bit, MSR[DE]. Includes the debug syndrome register (DBSR).

Because save/restore register pairs are serially reusable, care must be taken to preserve program state that may be lost when an unordered interrupt is taken.

6.2.3.4 Interrupt registers

The registers associated with interrupt handling are described in Table 6-4.

Table 6-4. Interrupt registers

Register	Description
Noncritical interrupt registers	
SRR0	Save/restore register 0—Stores the address of the instruction causing the exception or the address of the instruction that will execute after the **rfi** instruction.
SRR1	Save/restore register 1—Saves machine state on noncritical interrupts and restores machine state after an **rfi** instruction is executed.
Critical interrupt registers	

Table 6-4. Interrupt registers (continued)

Register	Description
CSRR0	Critical save/restore register 0—On critical interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfci**.
CSRR1	Critical save/restore register 1—Saves machine state on critical interrupts and restores machine state after an **rfci** instruction is executed.
Debug interrupt registers	
DSRR0	Debug save/restore register 0—Used to store the address of the instruction that will execute after an **rfdi** instruction is executed.
DSRR1	Debug save/restore register 1—Stores machine state on debug interrupts and restores machine state after an **rfdi** instruction is executed.
Machine check interrupts	
MCSRR0	Machine check save/restore register 0—On machine check interrupts, stores either the address of the instruction causing the exception or the address of the instruction that executes after the **rfmci** instruction.
MCSRR1	Machine check save/restore register 1—Saves machine state on machine check interrupts and restores those values when an **rfmci** instruction is executed
Syndrome registers	
MCSR	Machine check syndrome register—Saves machine check syndrome information on machine check interrupts.
ESR	Exception syndrome register—Provides a syndrome to differentiate among the different kinds of exceptions that generate the same interrupt type. Upon generation of a specific exception type, the associated bits are set and all other bits are cleared.
SPE interrupt registers	
SPEFSCR	Signal processing and embedded floating-point status and control register—Provides interrupt control and status as well as various condition bits associated with the operations performed by the SPE.
Other interrupt registers	
DEAR	Data exception address register—Contains the address that was referenced by a load, store, or cache management instruction that caused an alignment, data TLB miss, or data storage interrupt.
IVPR IVORs	Together, IVPR[32–47] || IVOR*n* [48–59] || 0b0000 define the address of an interrupt-processing routine.
MSR	Machine state register—Defines the state of the processor. When an interrupt occurs, it is updated to preclude unrecoverable interrupts from occurring during the initial portion of the interrupt handler

Each interrupt has an associated interrupt vector address, obtained by concatenating IVPR[32–47] with the address index in the associated IVOR (that is, IVPR[32–47] || IVOR*n*[48–59] || 0b0000). The resulting address is that of the instruction to be executed when that interrupt occurs. IVPR and IVOR values are

indeterminate on reset and must be initialized by the system software using **mtspr**. Table 6-5 lists IVOR registers implemented on the e200z7 core and the associated interrupts.

Table 6-5. Exceptions and conditions

IVOR*n*	Interrupt type		IVOR*n*	Interrupt type
None[1]	System reset (not an interrupt)		10	Decrementer
0[2]	Critical input		11	Fixed-interval timer
1	Machine check		12	Watchdog timer
2	Data storage		13	Data TLB error
3	Instruction storage		14	Instruction TLB error
4[2]	External input		15	Debug
5	Alignment		16–31	Reserved
6	Program		32	SPE unavailable
7	Floating-point unavailable		33	SPE data exception
8	System call		34	SPE round exception
9	APU unavailable (not used by this core)			

1 Vector to [*p_rstbase[0:29]*] || 0xFFC.

2 Autovectored external and critical input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

NOTE

After power-on, the IVOR*n* registers are not initialized to a known value. The user software must initialize the IVOR*n* registers before any other register after power-on.

6.3 Microarchitecture summary

The e200z7 processor has a ten-stage pipeline with four stages for instruction execution. These stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions:

1. Instruction fetch 0
2. Instruction fetch 1
3. Instruction fetch 2
4. Instruction decode 0
5. Instruction decode 1/register file read/effective address calculation
6. Execute 0/memory access 0
7. Execute 1/memory access 1
8. Execute 2/memory access 2
9. Execute 3
10. Register writeback

The integer execution unit consists of a 32-bit AU, a LU, a 32-bit Shifter, a MIU, a CRU, a CRZ, a 32 × 32 hardware multiplier array, result feed-forward hardware, and support hardware for division.

Most arithmetic and logical operations are executed in a single cycle with the exception of multiply, which is implemented with a pipelined hardware array, and the divide instructions. A count-leading-zeros unit operates in a single clock cycle.

The instruction unit contains a program counter incrementer and a dedicated branch address adder to minimize delays during change-of-flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching is performed to accelerate taken branches. Prefetched instructions are placed into an instruction buffer capable of holding six instructions.

Branch target addresses are calculated in parallel with branch instruction decode, resulting in execution time of four clocks for correctly predicted branches. Conditional branches that are not taken execute in a single clock. Branches with successful BTB target prefetching have an effective execution time of one clock if correctly predicted.

Memory load and store operations are provided for byte, halfword, word (32-bit), and doubleword data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single-cycle throughput. Load and store multiple word instructions allow low-overhead context save and restore operations. The load/store unit (LSU) contains a dedicated effective address adder to optimize effective address generation.

The condition register unit supports the condition register (CR) and condition register operations defined by the architecture. The CR consists of eight 4-bit fields that reflect the results of certain operations generated by instructions such as move, integer and floating-point compare, arithmetic, and logical instructions. The CR also provides a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

The SPE category supports vector instructions operating on 8-, 16- and 32-bit integer and fractional data types. The vector and scalar floating-point instructions operate on 32-bit IEEE Std 754™ single-precision floating-point formats, and support single-precision floating-point operations in a pipelined fashion.

The 64-bit GPRs are used for source and destination operands for all vector instructions, and there is a unified storage model for single-precision floating-point data types of 32 bits and the normal integer type. The following low latency fixed-point and floating-point operations are provided:

- Add
- Subtract
- Mixed add/subtract
- Sum
- Diff
- Min
- Max
- Multiply
- Multiply-add

- Multiply-sub
- Divide
- Square root
- Compare
- Conversion

Most operations can be pipelined.

6.3.1 Instruction unit features

The e200z7 instruction unit implements the following:

- 64-bit fetch path that supports fetching of two 32-bit or up to four 16-bit VLE instructions per clock
- Instruction buffer holds up to ten 32-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch processing unit with dedicated branch address adder and branch target buffer (BTB) supporting single-cycle execution of successfully predicted branches

6.3.2 Integer unit features

The integer unit supports single-cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count-leading-zeros function
- 32-bit single-cycle barrel shifter for static shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 4 to 15 clocks with minimized execution timing
- Pipelined 32 × 32 hardware multiplier array that supports 32 × 32 → 32 multiply with 3-clock latency, 1-clock throughput

6.3.3 Load/Store Unit (LSU) features

The e200z7 LSU supports load, store, and load multiple/store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- Dedicated 64-bit interface to memory supports saving and restoring of up to 2 registers per cycle for load multiple and store multiple word instructions

6.3.4 L1 cache features

The features of the cache are as follows:

- Separate 16 KB, 4-way set-associative instruction and data caches

- Copyback and writethrough support
- 8-entry store buffer
- Push buffer
- Line-fill buffers with critical doubleword forwarding for both data loads and instruction fetches
- 32-bit address bus plus attributes and control
- Separate unidirectional 64-bit read and 64-bit write data buses
- Cache line locking
- Data cache locking control instructions
  - Data Cache Block Touch and Lock Set (**dcbtls**)
  - Data Cache Block Touch for Store and Lock Set (**dcbtstls**)
  - Data Cache Block Lock Clear (**dcblc**)
- Instruction cache locking control instructions
  - Instruction Cache Block Touch and Lock Set (**icbtls**)
  - Instruction Cache Block Lock Clear (**icblc**)
- Way allocation
- Write allocation policies
- Tag and data parity
- Hardware cache coherency support for the data cache
- Supports multibit EDC for the instruction cache
- Supports parity for the data cache
- Correction/auto-invalidation capability for the instruction and data caches
- e200z7-specific L1 cache flush and invalidate registers (L1FINV0 and L1FINV1) support software-based flush and invalidation control on a set and way basis

6.3.5 Memory Management Unit (MMU) features

The MMU is an implementation of the embedded MMU category of the Power ISA, with the following feature set:

- 32-bit effective-to-real address translation
- 8-bit process identifier (PID)
- 64-entry, fully associative TLB
- Support for multiple page sizes (1 KB to 4 GB)
- Software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions
- Entry flush protection
- Byte ordering (endianness) configurable on a per-page basis

6.3.6 System bus (core complex interface) features

The features of the core complex interface are as follows:

- Independent instruction and data buses
- Advanced microcontroller bus architecture (AMBA) and advanced high-performance bus (AHB2.v6)-Lite protocol
- 32-bit address bus plus attributes and control on each bus
- Instruction interface supports read transfers of 16, 32, and 64 bits
- Data interface has separate unidirectional 64-bit read data bus and 64-bit write data bus
- Both the instruction and data interface buses support misaligned transfers, true big- and little-endian operating modes, and operates in a pipelined fashion

6.3.7 Nexus 3+ module features

The Nexus 3+ module provides real-time development capabilities for e200z7 processors in compliance with the IEEE-ISTO 5001™-2008 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility. The ‘3+’ suffix indicates that some Nexus Class 4 features are implemented.

A portion of the pin interface (the JTAG port) is shared with the OnCE/Nexus 1 unit. The IEEE-ISTO 5001-2008 standard defines an extensible auxiliary port, which is used in conjunction with the JTAG port in e200z7 processors.

Chapter 7
General-Purpose Static RAM (SRAM)

7.1 Introduction

Microcontrollers in the MPC5675K family provide as much as 512 KB general-purpose SRAM with the following features:

- SRAM can be read/written from any bus master
- Byte, halfword, word, and doubleword addressable
- Single-bit correction and double-bit error detection

7.2 SRAM Controller (SRAMC)

7.2.1 Overview

- Duplicated SRAM controller (including ECC logic) to enable high diagnostic coverage
- Configurable support for 0 WS (wait states) and 1 WS for single read and burst read accesses
- Selective insertion of WS based on XBAR logical master ID in Decoupled Parallel Mode (DPM)
- Support for 32-byte bursts with a read burst support pattern of
  - 2-1-1-1 @ 90 MHz at 1 WS
- ECC protection extended to SRAM array address lines
- One bit error correction and two bit error detection on a 64-bit boundary
- 64-bit wide (net data width) interface to the SRAM array
- Support for the total SRAM size in LSM mode (max address)

7.2.2 ECC on SRAM array address lines

The SRAMC calculates an 8-bit ECC over the data and address lines required to address the implemented SRAM memory space.

7.2.3 System integration

The two instantiations of the SRAM controller are referred to as SRAMC_0 and SRAMC_1. Both instantiations are identical but they differ slightly in Lock Step Mode (LSM) or Decoupled Parallel Mode (DPM).

Both SRAM arrays shown in Figure 7-1 and Figure 7-2 are built out of several smaller SRAM cuts.

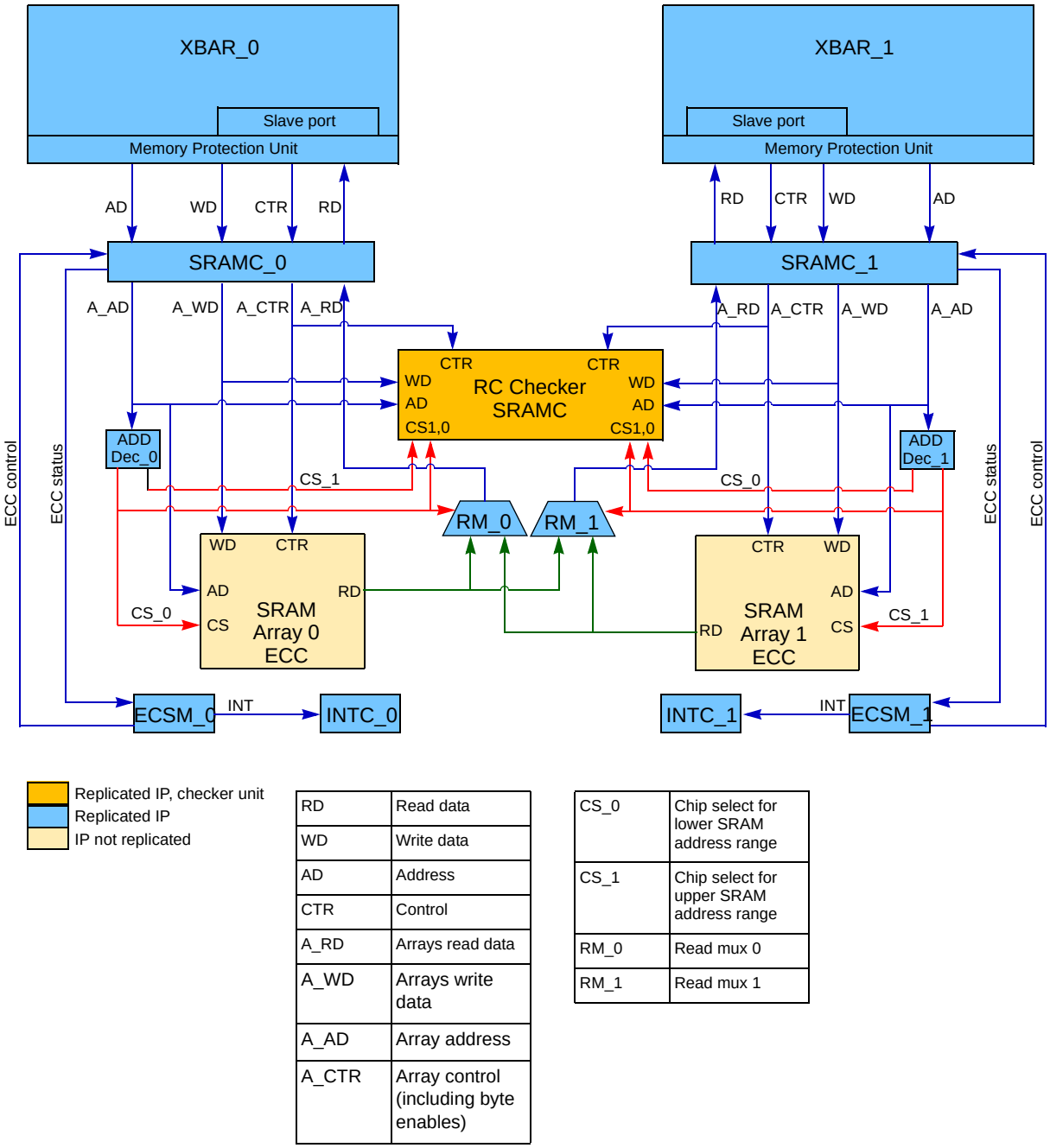
7.2.3.1 Lock Step Mode (LSM)

Figure 7-1 shows the SRAM array integration in LSM. The two SRAM arrays shown in Figure 7-1 are memory mapped sequentially without gaps. Depending on the SRAM size implemented in each array,

ADD Dec_0 and ADD Dec_1 need to decode different address lines to achieve a contiguous mapping of both SRAM arrays in the MPC5675K memory map.

In LSM, SRAM accesses are received by both SRAM controllers simultaneously. Both SRAMC need to be configured to accept accesses to the whole device SRAM range (max address set to total SRAM size). SRAMC_0 forwards the access to ADD Dec_0 and SRAM Array 0. SRAMC_1 forwards the access to ADD Dec_1 and SRAM Array 1. Depending on the address (access to the lower half of the implemented total SRAM on a device or to the upper half), either SRAM Array 0 is activated by ADD Dec_0 or SRAM Array 1 by ADD Dec_1. SRAM Array 0 and SRAM Array 1 shall never be activated (by asserting CS_0 and CS_1) at the same time. The activation signals (CS_0 and CS_1) generated from both ADD decoders also need to be sent to the RC checker for comparison.

Both ADD decoders also need to generate the mux select signals for the two read mux RM_0 and RM_1 to ensure proper read data routing back to both SRAMC.



RD	Read data
WD	Write data
AD	Address
CTR	Control
A_RD	Arrays read data
A_WD	Arrays write data
A_AD	Array address
A_CTR	Array control (including byte enables)

CS_0	Chip select for lower SRAM address range
CS_1	Chip select for upper SRAM address range
RM_0	Read mux 0
RM_1	Read mux 1

Figure 7-1. SRAM Integration in LSM

Table 7-1, Table 7-2, and Table 7-3 define the address ranges that both instantiations of the address decoder (ADD Dec_0 and ADD Dec_1) need to decode for the different MPC5675K microcontrollers. The tables as well as Figure 7-1 assume active high logic, however that shall not limit the selection of the active signal levels chosen for the implemented logic.

NOTE

The described SRAM architecture does NOT implement any automated RAM content duplication distributed to the two arrays. Accesses to the lower half of the implemented total SRAM on the device are routed to SRAM Array 0 and accesses to the upper half of the implemented total SRAM on the device are routed to the SRAM Array 1.

Table 7-1. Address decoder details for MPC5673K

Start address (offset)	Stop address (offset)	Size [KB]	ADD Dec_0 ADD Dec_1	
			CS_0	CS_1
0x0_0000	0x1_FFFF	128	1	0
0x2_0000	0x3_FFFF	128	0	1

Table 7-2. Address decoder details for MPC5674K

Start address (offset)	Stop address (offset)	Size [KB]	ADD Dec_0 ADD Dec_1	
			CS_0	CS_1
0x0_0000	0x2_FFFF	192	1	0
0x3_0000	0x5_FFFF	192	0	1

Table 7-3. Address decoder details for MPC5675K

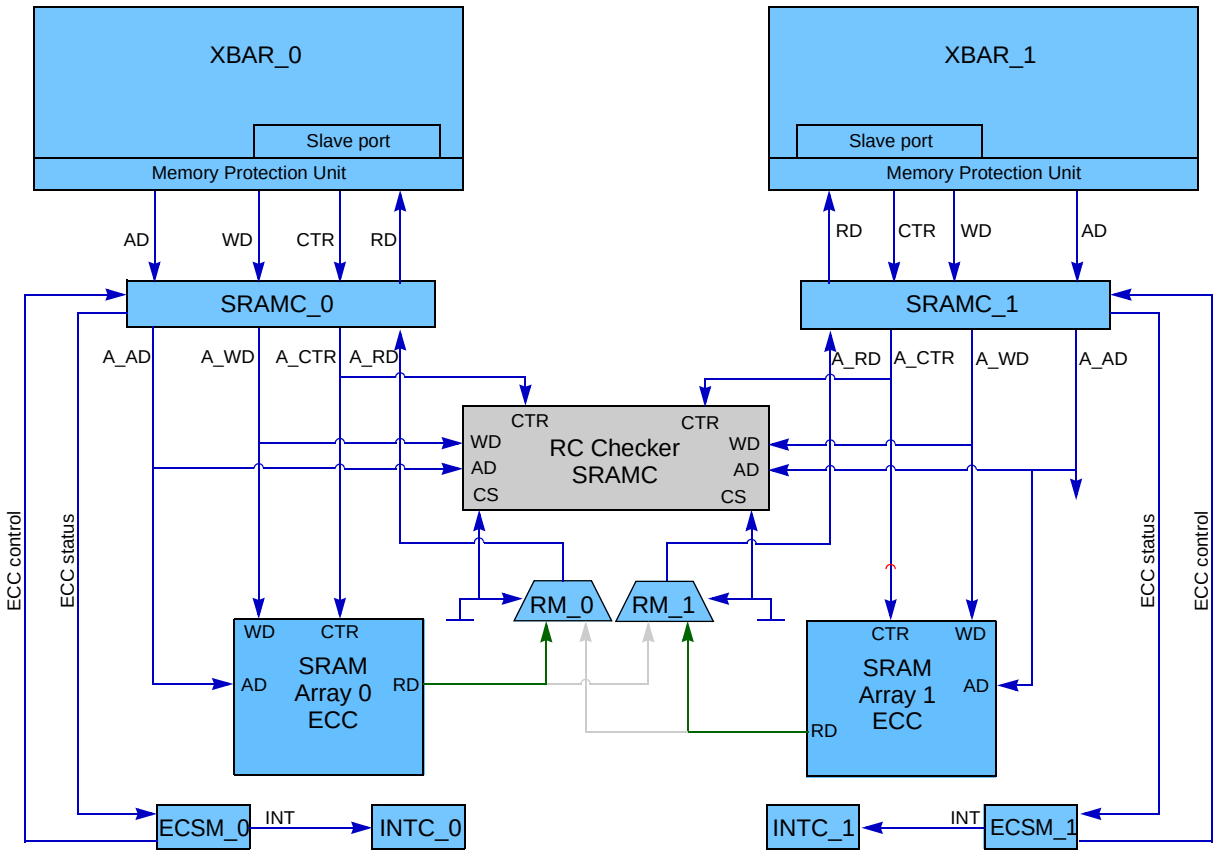
Start address (offset)	Stop address (offset)	Size [KB]	ADD Dec_0 ADD Dec_1	
			CS_0	CS_1
0x0_0000	0x3_FFFF	256	1	0
0x4_0000	0x7_FFFF	256	0	1

7.2.3.2 Decoupled Parallel Mode (DPM)

Figure 7-2 shows the SRAM array integration in DPM. The two SRAM arrays shown in Figure 7-2 are completely decoupled and are mapped at different (noncontiguous) address locations within the memory map. For more details please see Chapter 2, Memory Map.

In contrast to LSM, shown in Figure 7-1, in DPM no additional address decoders are required. In DPM, the ADD Dec_0 and ADD Dec_1 are bypassed such that the SRAMC directly controls the SRAM array. The select input of the read mux RM_0 and RM_1 is fixed to a static state to only forward the read data from Array 0 to SRAMC_0 and the read data from SRAM Array 1 to SRAMC_1. The RCCU SRAMC is disabled in DPM.

Both SRAMC need to be configured to accept accesses to half of the whole device SRAM range (max address set to half of the total SRAM size).



RD	Read data
WD	Write data
AD	Address
CTR	Control
A_RD	Arrays read data
A_WD	Arrays write data
A_AD	Array address

A_CTR	Array control (including byte enables)
CS_0	Chip select for lower SRAM address range
CS_1	Chip select for upper SRAM address range
RM_0	Read mux 0
RM_1	Read mux 1

Figure 7-2. SRAM integration in DPM

7.3 SRAM operating mode

The SRAM has only one operating mode, the Normal mode, which allows reads and writes of the SRAM. No Standby mode is available.

7.4 Registers

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM.

7.5 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 72-bit reads (64-bit data bus plus the 8-bit ECC) that return all 0s or all 1s, asserts an error indicator on the bus cycle, and sets the error flag

The inclusion of address bits in the ECC algorithm offers greater coverage, specifically in the detection three or more bit flips.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)
- 8 bytes or 2 words (0:63 bits)

On any write, the most significant 29 address bits are calculated as part of the ECC.

If the entire 64 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 64-bit data bus. The 8-bit combination of address and data ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 64-bit data width (1-, 2-, or 4-byte segment), the following occurs:

1. The ECC mechanism checks the entire 64-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1-, 2-, or 4-byte segment) are merged with the corrected 64 bits on the data bus.
3. The ECC is then calculated on the resulting 64 bits formed in the previous step and combined with the address ECC.
4. The 8-bit ECC result is appended to the 64 bits from the data bus, and the 72-bit value is then written to SRAM.

7.5.1 Access timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. Table 7-4 lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

Current operation — Lists the type of SRAM operation executing currently

Previous operation — Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)

Wait states — Lists the number of wait states (bus clocks) the operation requires, which depends on the combination of the current and previous operation

Table 7-4. Number of wait states required for SRAM operations

	Current operation	Previous operation	Number of wait states required
Read operation	Read	Idle	1
		Pipelined read	
		8-, 16-, or 32-bit write	0 (read from the same address)
			1 (read from a different address)
	Pipelined read	Read	0
Write operation	8- or 16-bit write	Idle	1
		Read	
		Pipelined 8- or 16-bit write	2
		32-bit write	
		8- or 16-bit write	0 (write to the same address)
	Pipelined 8-, 16-, or 32-bit write	8-, 16-, or 32-bit write	0
	32-bit write	Idle	1
		32-bit write	
		Read	
	64-bit write	Idle	0
		64-bit write	
		Read	

7.5.2 Reset effects on SRAM accesses

Asynchronous reset can possibly corrupt RAM if it is asserted during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed. In case of no access ongoing when reset occurs, the RAM corruption does not happen.

If an initialization procedure is needed that does not require RAM initialization, synchronous reset (software reset) should be used.

7.6 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 64-bit write operations prior any read accesses. This is also true for implicit read accesses caused by any write accesses of less than 64-bit as discussed in Section 7.5, SRAM ECC mechanism.

7.7 Initialization and application information

To use the SRAM after power on, all the bits of the SRAM must be initialized using a 64-bit write. If a non-64-bit write access is done after power on, it generates an ECC error because such an access implicitly does an ECC check on uninitialized locations.

NOTE

SRAM *must* be initialized, even if the application does not use ECC reporting.

Chapter 8
Flash Memory

8.1 Introduction

A flash memory module is a nonvolatile memory device consisting of blocks (also called sectors) of storage elements, an electrical means for selectively adding (programming) and removing (erasing) charge from these elements, and a means of selectively sensing (reading) the charge stored in these elements. The primary function of the flash memory module is to serve as electrically programmable and erasable nonvolatile memory (NVM). Nonvolatile memory can be used for instruction and/or data storage.

The MPC5675K implements three flash memory modules: two code flash memory modules (CFM0 and CFM1) and one data flash memory module (DFM0). The code flash memory modules are used primarily for code storage, while the data flash memory module is intended for storage of nonvolatile data or emulation. Read-While-Write (RWW) is supported between different flash memory modules, but not within a flash memory module.

Each flash memory module is arranged as two functional units: the flash memory core and the memory interface.

The flash memory core is composed of arrayed nonvolatile storage elements, sense amplifiers, row decoders, column decoders, and charge pumps. The arrayed storage elements in the flash memory core are sub-divided into physically separate units referred to as blocks (or sectors).

The flash memory core is organized including Error Correction Code (ECC) circuitry. ECC circuitry provides correction of single-bit faults and is used to achieve automotive reliability targets. Some units will experience single-bit corrections throughout the life of the product with no impact to product reliability.

The memory interface contains the registers and logic that control the operation of the flash memory core. The memory interface is also the interface between the flash memory module and the flash memory controller, also called the Bus Interface Unit (BIU).

The flash memory controller connects the flash memory modules to the system bus via the XBAR, and contains all system level customization required for the device.

Each code flash memory module supports memory sector sizes ranging from 16 KB to 128 KB of user memory, plus 16 KB of test memory. A portion of test memory is One-Time Programmable (OTP) by the user. An extra 16 KB sector is available as shadow space.

The data flash memory module supports memory sizes of 4 × 16 KB of user memory, plus 8 KB of test memory. The data flash memory module supports just one user address space: Low Address Space (LAS). Only one block size is available to the user in the data flash memory core: 16 KB. 8 KB is reserved for test flash memory.

The flash memory modules are divided into blocks to implement independent program/erase protection. A software mechanism is provided to independently lock/unlock each block in low, mid, and high address space against program and erase.

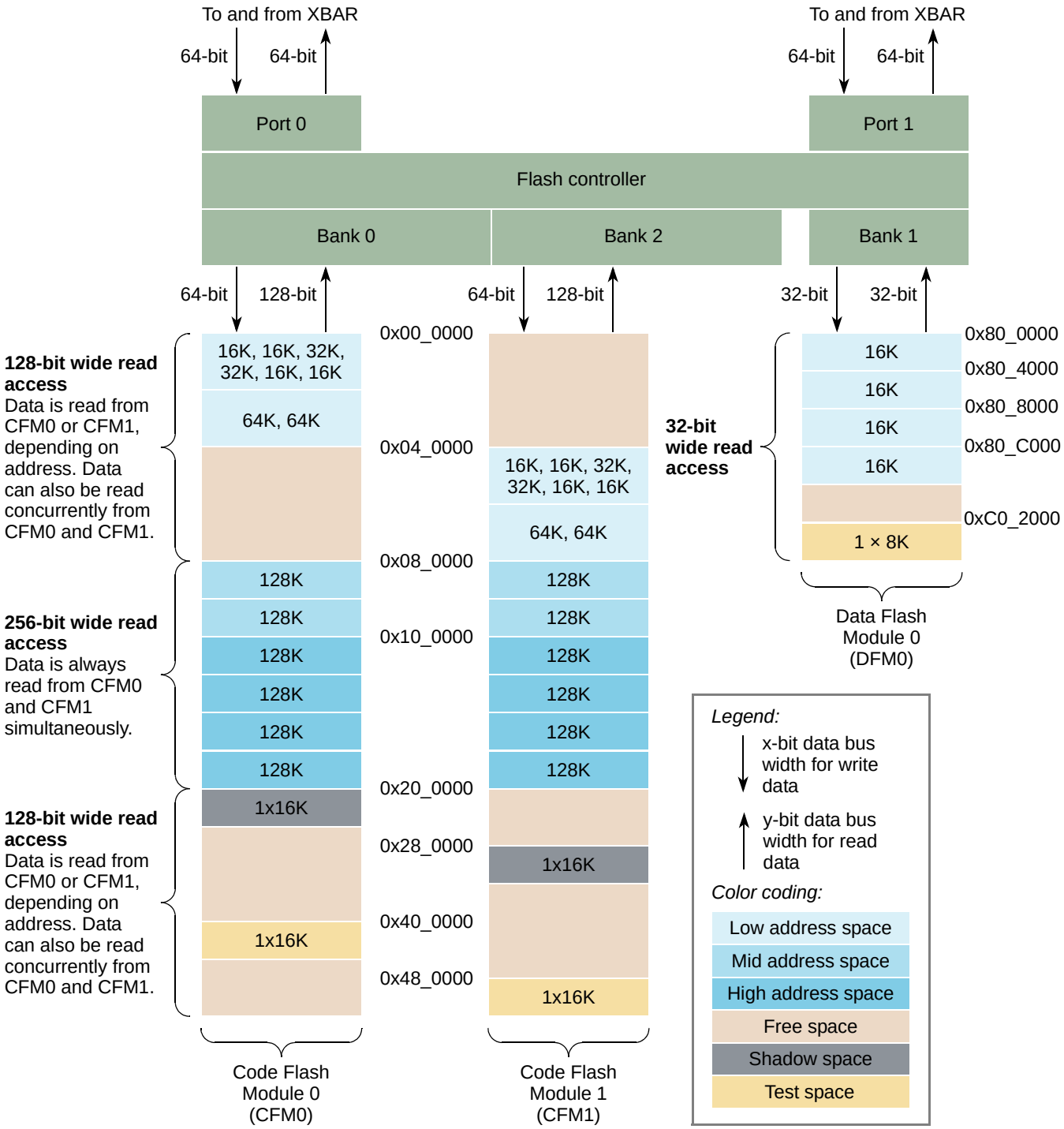


Figure 8-1. Flash memory subsystem architecture

8.1.1 Features

The flash memory subsystem has these major features:

- Features common to code flash memory and data flash memory:

  - Support for a 32-bit data bus for CPU loads and DMA access (byte, halfword, word, and doubleword reads are supported; only aligned word and doubleword writes are supported)
  - Configurable read buffering and line prefetch support; prefetch controller is used to support single-cycle read responses for hits in the buffers
  - Erase of selected block(s)
  - Erase suspend supported
  - ECC with single-bit correction, single-bit detection, and double-bit detection
  - Embedded hardware program and erase algorithm
  - Configurable wait states allow use in a wide range of system frequencies
  - Shadow information stored in nonvolatile shadow sector
  - Independent program/erase of the shadow block
  - Software programmable block program/erase restriction control for low, mid, and high address spaces
  - Hardware and software configurable read and write access protections on a per-master basis
  - Memory-mapped test sector for direct access to device-specific data and calibration information
- Code flash memory
  - Support for a 64-bit data bus for instruction fetch
  - Read page size of 128 bits (low-address space) and 256 bits (for mid/high-address space)
  - Read-while-write between different banks
  - Memory-mapped test flash memory sector for direct access to device-specific data and calibration information
- Data flash memory
  - Multiple-mapping support and mapping-based block access timing (0–31 additional cycles) allowing use for emulation of other memory types

8.1.2 Modes of operation

8.1.2.1 Flash memory user mode

User mode is the default operating mode of the flash memory modules. In this mode, it is possible to read, write, program, and erase memory within the flash memory module.

8.1.2.2 User test mode

User Test mode is a procedure to check the integrity of the flash memory module. For more information, see Section 8.3.2, User test mode.

8.2 Memory map and registers

8.2.1 Module memory map

Table 8-1 lists the base addresses of the flash memory modules. For the modules listed in Table 8-1, the base address is independent of the device mode (LSM or DPM). The two flash memory controllers shown in Figure 8-36 are configured via three registers of the Code Flash Module 0 (CFM0).

Table 8-1. Flash memory related modules base addresses

Mode	Module	FLASH_BASE address
Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM)	Code Flash Module 0 (CFM0)	0xC3F8_8000
	Data Flash Module 0 (DFM0)	0xC3F8_C000
	Code Flash Module 1 (CFM1)	0xC3FB_0000

Table 8-2 shows the sectorization of the flash memory modules.

Table 8-2. Flash memory multi module sectorization

FLASH_SEC_BASE addresses	Flash memory usage	Address space range	Flash memory		Block size	Data bus width for reads
			Module	Sector		
0x0000_0000	Code Flash	Low Address Space	CFM0[1]	B0F0	16 KB	128
0x0000_4000				B0F1	16 KB	128
0x0000_8000				B0F2	32 KB	128
0x0001_0000				B0F3	32 KB	128
0x0001_8000				B0F4	16 KB	128
0x0001_C000				B0F5	16 KB	128
0x0002_0000				B0F6	64 KB	128
0x0003_0000				B0F7	64 KB	128
0x0004_0000			CFM1[2]	B0F0	16 KB	128
0x0004_4000				B0F1	16 KB	128
0x0004_8000				B0F2	32 KB	128
0x0005_0000				B0F3	32 KB	128
0x0005_8000				B0F4	16 KB	128
0x0005_C000				B0F5	16 KB	128
0x0006_0000				B0F6	64 KB	128
0x0007_0000				B0F7	64 KB	128
0x0008_0000		Mid Address Space[3]	CFM0[2] and CFM1[3]	2 × B0F8	2 × 128 KB	256
0x000C_0000				2 × B0F9	2 × 128 KB	256
0x0010_0000		High Address Space[3]	CFM0[2] and CFM1[3]	2 × B0FA	2 × 128 KB	256
0x0014_0000				2 × B0FB	2 × 128 KB	256
0x0018_0000				2 × B0FC	2 × 128 KB	256
0x001C_0000				2 × B0FD	2 × 128 KB	256
0x0020_0000		Shadow	CFM0[2]	B0SH	16 KB	128
0x0020_4000		Reserved	—	—	496 KB	—
0x0028_0000		Shadow	CFM1[3]	B1SH	16 KB	128
0x0028_4000		Reserved	—	—	1520 KB	—
0x0040_0000		Test	CFM0[2]	B0TF	16 KB	128
0x0040_4000		Reserved	—	—	496 KB	—
0x0048_0000		Test	CFM1[3]	B1TF	16 KB	128
0x0048_4000		Reserved	—	—	3568 KB	—

Table 8-2. Flash memory multi module sectorization (continued)

FLASH_SEC_BASE addresses	Flash memory usage	Address space range	Flash memory		Block size	Data bus width for reads
			Module	Sector		
0x0080_0000	Data Flash	Low Address Space	DFM0[4]	B0F0	16 KB	32
0x0080_4000				B0F1	16 KB	32
0x0080_8000				B0F2	16 KB	32
0x0080_C000				B0F3	16 KB	32
0x0081_0000		Reserved		—	4032 KB	—
0x00C0_0000		Reserved		—	8 KB	—
0x00C0_2000		Test	DFM0[4]	B0TF	8 KB	32

1 Code Flash Module 0.

2 Code Flash Module 1.

3 The first (lower addresses) 16 bytes (128 bits) of every 32-byte aligned range are stored in CFM0 and the last 16 bytes (128 bits) are stored in CFM1.

4 Data Flash Module 0.

8.2.2 Register overview

The flash memory user register mapping is shown in Table 8-3. An “X” in the code flash memory or data flash memory columns (CFM0, CFM1, and DFM0, respectively) indicates that the register is available in that module. A dash indicates that the corresponding register is not available in the corresponding flash memory module.

Table 8-3. Flash memory user registers

Offset from FLASH_BASE[1]	Register	CFM0[2]	CFM1[3]	DFM0[4]	Location
0x0000	Module Configuration Register (MCR)	X	X	X	on page 168
0x0004	Low/Mid Address Space Block Locking Register (LML)	X	X	X	on page 174
0x0008	High Address Space Block Locking Register (HBL)	X	X	—[5]	on page 176
0x000C	Secondary Low/Mid Address Space Block Lock Register (SLL)	X	X	X	on page 177
0x0010	Low/Mid Address Space Block Select Register (LMS)	X	X	X	on page 179
0x0014	High Address Space Block Select Register (HBS)	X	X	—[5]	on page 180
0x0018	Address Register (ADR)	X	X	X	on page 181
0x001C	Platform Flash Configuration Register 0 (PFCR0)	X[6]	—[6]	—[6]	on page 183
0x0020	Platform Flash Configuration Register 1 (PFCR1)	X[6]	—[6]	—[6]	on page 187
0x0024	Platform Flash Access Protection Register (PFAPR)	X[6]	—[6]	—[6]	on page 190
0x0028–0x0038	Reserved				
0x003C	User Test Register 0 (UT0)	X	X	X	on page 192

Table 8-3. Flash memory user registers (continued)

Offset from FLASH_BASE[1]	Register	CFM0[2]	CFM1[3]	DFM0[4]	Location
0x0040	User Test Register 1 (UT1)	X	X	X	on page 195
0x0044	User Test Register 2 (UT2)	X	X	—[6]	on page 196
0x0048	User Multiple Input Signature Register 0 (UMISR0)	X	X	X	on page 196
0x004C	User Multiple Input Signature Register 1 (UMISR1)	X	X	X	on page 197
0x0050	User Multiple Input Signature Register 2 (UMISR2)	X	X	—[6]	on page 199
0x0054	User Multiple Input Signature Register 3 (UMISR3)	X	X	—[6]	on page 199
0x0058	User Multiple Input Signature Register 4 (UMISR4)	X	X	—[6]	on page 200
0x005C–0x3FFF	Reserved				

1 See Table 8-1 for FLASH_BASE value.
2 Code Flash Module 0.
3 Code Flash Module 1.
4 Data Flash Module 0.
5 This address is reserved.
6 The configuration of this register is always applied to both flash memory controllers in order to allow them to work in a lock-step configuration independent of whether the device is operating in LSM or DPM.

Whenever MCR[DONE] or UT0[AID] are low, the flash memory user registers are not accessible. Reads return indeterminate data, and writes have no effect.

In the following, some nonvolatile registers are described. Such entities are not flip-flops, but locations of test flash memory sector with a special meaning.

During the flash memory initialization phase, the Flash Program/Erase Controller (FPEC) reads these nonvolatile registers and updates their related volatile registers. When the FPEC detects ECC double-bit errors in these special locations, it behaves in the following way:

- In case of failing system locations (configurations, redundancy, etc.), the initialization phase is interrupted and a fatal error is flagged.
- In case of failing user locations (protections, etc.), the volatile registers are filled with all ‘1’s and the flash memory initialization ends, setting the MCR[PEG] bit low.

8.2.3 Register descriptions

The flash memory user registers represent the communication interface between the host CPU and the Flash Program/Erase Controller (FPEC). Some register bits (command bits) are read/write for the CPU and read-only for the FPEC. Some other register bits (status bits) are read/write for the FPEC and read-only for the CPU.

8.2.3.1 Module Configuration Register (MCR)

The Module Configuration Register (MCR) enables and monitors all the modify operations for each flash memory module. The structure of the MCR is similar for code and data flash memory, but has different reset values to reflect the different sizes of the memory space. The MCR is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE			0	LAS			0	MAS		
W	w1c															
Reset	0	0	0	0	0	0	1	1	0	1	1	1	0	—[1]	—[1]	—[1]

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 8-2. Module Configuration Register (MCR)

[1] Varies depending on flash memory module. See MAS field description.

Table 8-4. MCR field descriptions

Field	Description
EDC	ECC Data Correction The EDC bit provides information on previous reads. When an ECC single-bit error detection and correction occurs, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. In the event of an ECC double-bit error detection, this bit is not set. If EDC is not set or remains 0, it indicates that all previous reads (from the last reset or clearing of EDC) were not corrected through ECC. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 has no effect. This bit cannot be set by the user. EDC bit functionality is enabled or disabled by the UT0[SBCE] bit. 0 Reads are occurring normally. 1 An ECC single-bit error occurred and was corrected during a previous read. **Note:** An ECC event does not mean device failure; the chip still is functional. Parts with single-bit ECC corrections are fully functional.

Table 8-4. MCR field descriptions (continued)

Field	Description
SIZE	Array Space Size The value of the SIZE field depends on the size of the flash memory module. 000 128 KB 001 256 KB 010 512 KB 011 1056 KB (the value for the MPC5675K devices in the code flash memory modules) 100 1536 KB 101 2048 KB 110 64 KB (the value for the MPC5675K devices in the data flash memory modules) 111 Reserved
LAS	Low Address Space The value of the LAS field corresponds to the configuration of the Low Address Space. 000 0 KB 001 2 × 128 KB (256 KB) 010 32 KB + (2 × 16 KB) + (2 × 32 KB) + 128 KB (256 KB) 011 Reserved 100 Reserved 101 Reserved 110 4 × 16 KB (64 KB, the value for the MPC5675K device in the data flash memory module) 111 (2 × 16 KB) + (2 × 32 KB) + (2 x 16 KB) + (2 × 64 KB) (256 KB, the value for the MPC5675K device in the code flash memory modules)
MAS	Mid Address Space The value of the MAS field corresponds to the configuration of the Mid Address Space. 000 2 × 128 KB (256 KB, the value for the MPC5675K device in the code flash memory modules) 111 MID block not present (the value for the MPC5675K device in the data flash memory module) All other values are reserved.
EER	ECC Event Error The EER bit provides information on previous reads. If an ECC double-bit error detection occurs, the EER bit is set to 1. This bit must then be cleared or a reset must occur before this bit can return to a 0 state. In the event of an ECC single-bit error detection and correction, this bit is not set. If EER is not set or remains 0, it indicates that all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. Writing 0 has no effect. This bit cannot be set by the user. 0 Reads are occurring normally. 1 An ECC double-bit error occurred during a previous read.

Table 8-4. MCR field descriptions (continued)

Field	Description
RWE	Read-While-Write Event Error The RWE bit provides information on previous reads when a modify operation is ongoing. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the flash memory module has occurred while the FPEC was performing a program or erase operation or an array integrity check. This bit must then be cleared or a reset must occur before this bit can return to a 0 state. If RWE is not set, or remains 0, it indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 has no effect. This bit cannot be set by the user. 0 Reads are occurring normally. 1 A RWW Error occurred during a previous read.
PEAS	Program/Erase Access Space The PEAS bit indicates which space is valid for program and erase operations: main array space or shadow/test space. PEAS = 0 indicates that the main address space is active for all flash memory module program and erase operations. PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes). 0 Shadow/Test address space is disabled for program/erase and main address space enabled. 1 Shadow/Test address space is enabled for program/erase and main address space disabled.
DONE	Modify operation done The DONE bit indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory module reset. DONE is set to 1 in the following cases: • At the end of program and erase high voltage sequences • After a 1-to-0 transition of EHV within t_{PABT} or t_{EABT}, equal to P/E abort latency, which terminates a high voltage program/erase operation. • After a 0-to-1 transition of ESUS within t_{ESUS}, time equal to erase suspend latency, which suspends an erase operation. DONE is cleared to 0 just after a 0-to-1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation. 0 Flash memory is executing a high voltage operation. 1 Flash memory is not executing a high voltage operation.

Table 8-4. MCR field descriptions (continued)

Field	Description
PEG	Program/erase good The PEG bit indicates the completion status of the last flash memory program, erase, array integrity check, or MM sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program, erase, array integrity check, or MM high voltage operations. Aborting a program/erase/array integrity check/MM high voltage operation causes PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the flash memory module is reset, unless a flash memory initialization error has been detected. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a program/erase/array integrity check/MM operation. PEG is valid until PGM/ERS makes a 1-to-0 transition or EHV makes a 0-to-1 transition. The value in PEG is not valid after a 0-to-1 transition of DONE caused by ESUS being set to logic 1. If Program or erase is attempted on blocks that are locked, the response is PEG = 1, indicating that the operation was successful, and the contents of the block were properly protected from the Program or erase operation. If a Program operation tries to program to ‘1’ bits that are at ‘0’, the program operation is correctly executed on the new bits to be programmed at ‘0’, but PEG is cleared, indicating that the requested operation has failed. In array integrity check or MM, PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0–1. 0 Program or erase, operation failed or aborted. 1 Program or erase operation successful. 0 Array integrity check or MM aborted. 1 Array integrity check or MM operation successfully concluded, with or without checksum errors.
PGM	Program The PGM bit sets up the flash memory module for a program operation. A 0-to-1 transition of PGM initiates a program sequence. A 1-to-0 transition of PGM ends the program sequence. PGM can be set only under User mode Read (ERS is low and UT0[AIE] is low). PGM can be cleared by the user only when EHV is low and DONE is high. 0 Flash memory is not executing a program sequence. 1 Flash memory is executing a program sequence.

Table 8-4. MCR field descriptions (continued)

Field	Description
PSUS	Program Suspend PSUS indicates whether the flash memory module is in program suspend or in the process of entering a suspended state. The module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set only when PGM and EHV are set. A 0-to-1 transition of PSUS starts the sequence that sets DONE, and places the flash memory module in program suspend. The module enters suspend within t_{PSUS} of this transition. PSUS can be cleared only when DONE and EHV are set. A 1-to-0 transition of PSUS with EHV = 1 starts the sequence that clears DONE and returns the flash memory module to program. The module cannot exit program suspend and clear DONE while EHV is low. 0 Program sequence is not suspended. 1 Program sequence is suspended.
ERS	Erase ERS sets up the flash memory module for an erase operation. A 0-to-1 transition of ERS initiates an erase sequence. A 1-to-0 transition of ERS ends the erase sequence. ERS can be set only under User mode Read (PGM is low and UT0[AIE] is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. 0 Flash memory is not executing an erase sequence. 1 Flash memory is executing an erase sequence.
ESUS	Erase Suspend The ESUS bit indicates that the flash memory module is in erase suspend or in the process of entering a Suspend state. The flash memory module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS = 1 and EHV = 1, and PGM = 0. A 0-to-1 transition of ESUS starts the sequence that sets DONE and places the flash memory in erase suspend. The flash memory module enters Suspend within t_{ESUS} of this transition. ESUS can be cleared only when DONE = 1 and EHV = 1, and PGM = 0. A 1-to-0 transition of ESUS with EHV = 1 starts the sequence that clears DONE and returns the module to erase. The flash memory module cannot exit erase suspend and clear DONE while EHV is low. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.

Table 8-4. MCR field descriptions (continued)

Field	Description
EHV	Enable High Voltage The EHV bit enables the flash memory module for a high voltage program/erase operation. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set under one of the following conditions: • Erase (MCR[ERS] = 1, MCR[ESUS] = 0, UT0[AIE] = 0) • Program (MCR[ERS] = 0, MCR[ESUS] = 0, MCR[PGM] = 1, UT0[AIE] = 0) MCR[DONE] is cleared Immediately after a 0-to-1 transition, which starts high voltage P/E operation. In normal operation, a 1-to-0 transition of EHV with MCR[DONE] high and MCR[ESUS] low terminates the current program/erase high voltage operation. When an operation is terminated, there is a 1-to-0 transition of EHV with MCR[DONE] low and the eventual Suspend bit low. A termination causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the terminated operation contain indeterminate data after a termination. A suspended operation cannot be terminated. Terminating a high voltage operation leaves the flash memory module addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks. EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after MCR[ESUS] is set and before MCR[DONE] transitions high. EHV may not be cleared after MCR[ESUS] is cleared and before MCR[DONE] transitions low. 0 Flash memory is not enabled to perform an high voltage operation. 1 Flash memory is enabled to perform an high voltage operation.

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit-by-bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously. The flash memory module does not allow the user to write bits simultaneously, which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in Table 8-5.

Table 8-5. MCR bits set/clear priority levels

Priority Level	MCR Bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more MCR bits simultaneously, only the bit with the lowest priority level is written.

8.2.3.2 Low/Mid Address Space Block Locking Register (LML)

The Low/Mid Address Space Block Locking Register (LML) provides a means to protect blocks from being modified. These bits, along with bits in the SLL register, determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status. LML is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 8-3. Low/Mid Address Space Block Locking Register (LML)—Code flash memory modules

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	LLK3	LLK2	LLK1	LLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 8-4. Low/Mid Address Space Block Locking Register (LML)—Data flash memory module

Table 8-6. LML field descriptions

Field	Description
LME	Low/Mid Address Space Block Enable The LME bit enables the Lock registers (TSLK and LLK[3:0]) to be set or cleared by register writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the LML register. 0 Low Address Locks are disabled. TSLK and LLK[3:0] cannot be written. 1 Low Address Locks are enabled. TSLK and LLK[3:0] can be written.

Table 8-6. LML field descriptions (continued)

Field	Description
TSLK	Test Address Space Block Lock The TSLK bit locks the block of Test Address Space from program and erase (erase is any case forbidden for Test block). A value of 1 in the TSLK register signifies that the Test block is locked for program and erase. A value of 0 in the TSLK register signifies that the Test block is available to receive program and erase pulses. The TSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended or if a margin mode is ongoing. TSLK is not writable unless LME is high. 0 Test Address Space Block is unlocked and can be modified (if also SLL[STSLK] = 0). 1 Test Address Space Block is locked and cannot be modified.
MLK[1:0]	Mid Address Space Block Lock 1–0 These bits are used to lock the blocks of Mid Address Space from program and erase. MLK1–0 are related to sectors B0F9–8, respectively. A value of 1 in a bit of the MLK register signifies that the corresponding block is locked for program and erase. A value of 0 in a bit of the MLK register signifies that the corresponding block is available to receive program and erase pulses. The MLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK register is not writable if a high voltage operation is suspended. MLK is not writable unless LME is high. 0 Mid Address Space Block is unlocked and can be modified (if also SLL[SMLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.

Table 8-6. LML field descriptions (continued)

Field	Description
LLK[7:0]	Low Address Space Block Lock 3–0 These bits are used to lock the blocks of Low Address Space from program and erase. LLK[7:0] are related to sectors B0F[7:0], respectively. A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for program and erase. A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses. The LLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended or if a margin mode is ongoing. **Note:** LLK is not writable unless LME is high. 0 Low Address Space Block is unlocked and can be modified (if also SLL[SLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

8.2.3.3 High Address Space Block Locking Register (HBL)

The High Address Space Block Locking Register (HBL) provides a means to protect blocks from being modified.

NOTE

HBL is implemented in the code flash memory modules, but not in the data flash memory module.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	HLK 3	HLK 2	HLK 1	HLK 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 8-5. High Address Space Block Locking Register (HBL)—Code flash memory modules

Table 8-7. HBL field descriptions

Field	Description
HBE	High Address Space Block Enable This bit enables the Lock registers (HLK3-0) to be set or cleared by register writes. This bit is a status bit only. The method to set this bit is to write a password. If the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B2_2222 must be written to the HBL register. 0 High Address Locks are disabled: HLK3-0 cannot be written. 1 High Address Locks are enabled: HLK3-0 can be written.
HLK3-0	High Address Space Block Lock 3-0 These bits are used to lock the blocks of High Address Space (HAS) from program and erase. A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for program and erase. A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive program and erase pulses. The HLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended. HLK is not writable unless HBE is high. 0 HAS block is unlocked and can be modified. 1 HAS block is locked and cannot be modified.

8.2.3.4 Secondary Low/Mid Address Space Block Locking Register (SLL)

The Secondary Low/Mid Address Space Block Locking Register (SLL) provides an alternative means to protect blocks from being modified. These bits, along with bits in the LML register, determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status. SLL is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STS	0	0	SMK	SMK
W												LK			1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 8-6. Secondary Low/Mid Address Space Block Locking Register (SLL)—Code flash memory modules

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	SLK3	SLK2	SLK1	SLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 8-7. Secondary Low/Mid Address Space Block Locking Register (SLL)—Data flash memory module

Table 8-8. SLL field descriptions

Field	Description
SLE	SLE: Secondary Low/Mid Address Space Block Enable The SLE bit enables the Lock registers (STSLK and SLK3–0) to be set or cleared by register writes. This bit is a status bit only. The method to set this bit is to write a password. If the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLL register. 0 Secondary Low/Mid Address Locks are disabled. STSLK and SLK[3:0] cannot be written. 1 Secondary Low/Mid Address Locks are enabled. STSLK and SLK[3:0] can be written.
STSLK	STSLK: Secondary Test Address Space Block Lock The STSLK bit is used as an alternate means to lock the block of Test Address Space from program and erase. Erase is forbidden for the Test block. A value of 1 in the STSLK register signifies that the Test block is locked for program and erase. A value of 0 in the STSLK register signifies that the Test block is available to receive program and erase pulses. The STSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended or if a margin mode is ongoing. STSLK is not writable unless SLE is high. 0 Test Address Space Block is unlocked and can be modified (if also LML[TSLK] = 0). 1 Test Address Space Block is locked and cannot be modified.
SMK1–0	SMK1–0: Secondary Mid Address Space Block Lock 1-0 The SMK bits are used as an alternate means to lock the blocks of Mid Address Space from program and erase. SMK1–0 are related to sectors B0F9–8, respectively. A value of 1 in a bit of the SMK bitfield signifies that the corresponding block is locked for program and erase. A value of 0 in a bit of the SMK bitfield signifies that the corresponding block is available to receive program and erase pulses. The SMK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SMK bitfield is not writable if a high voltage operation is suspended. SMK is not writable unless SLE is high. 0 Mid Address Space Block is unlocked and can be modified (if also LML[MLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.

Table 8-8. SLL field descriptions (continued)

Field	Description
SLK[7:0]	Secondary Low Address Space Block Lock 7–0 These bits are used as an alternate means to lock the blocks of Low Address Space from program and erase. SLK[7:0] are related to sectors B0F[7:0], respectively. A value of 1 in a bit of the SLK bitfield signifies that the corresponding block is locked for program and erase. A value of 0 in a bit of the SLK bitfield signifies that the corresponding block is available to receive program and erase pulses. The SLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK bitfield is not writable if a high voltage operation is suspended or if a margin mode is ongoing. SLK is not writable unless SLE is high. 0 Low Address Space Block is unlocked and can be modified (if also LML[LLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

8.2.3.5 Low/Mid Address Space Block Select Register (LMS)

The Low/Mid Address Space Block Select Register (LMS) provides a means to select blocks to be operated on during erase. LMS is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL 1	MSL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-8. Low/Mid Address Space Block Select Register (LMS)—Code flash memory modules

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	LSL3	LSL2	LSL1	LSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-9. Low/Mid Address Space Block Select Register (LMS)—Data flash memory module

Table 8-9. LMS field descriptions

Field	Description
MSL[1:0]	Mid address space block select 1–0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. MSL[1:0] are related to sectors B0F[9:8], respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding MSL bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. 0 Mid Address Space Block is unselected for erase. 1 Mid Address Space Block is selected for erase. **Note:** The code flash memory modules implement MSL[1:0]. The data flash memory module does not implement the MSL bits. This difference is because of the different blocks and block sizes between the modules.
LSL[7:0]	Low address space block Select 3-0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. In the code flash memory modules, the LSL[7:0] bits are related to sectors B0F[7:0], respectively. In the data flash memory module, the LSL[3:0] bits are related to sectors B0F[3:0], respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended or if a margin mode is ongoing. In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. 0 Low Address Space Block is unselected for erase. 1 Low Address Space Block is selected for erase. **Note:** The code flash memory modules implement LSL[7:0]. The data flash memory module implements only LSL[3:0]. This difference is because of the different blocks and block sizes between the modules.

8.2.3.6 High Address Space Block Select Register (HBS)

The High Address Space Block Select Register (HBS) provides a means to select blocks to be operated on during erase.

NOTE

HBS is implemented in the code flash memory modules, but not in the data flash memory module, since no High Address Space is implemented in the data flash memory module.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	HSL 3	HSL 2	HSL 1	HSL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-10. High Address Space Block Select Register (HBS)—Code flash memory modules

Table 8-10. HBS field descriptions

Field	Description
HSL[3:0]	High Address Space (HAS) block Select 3–0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding HSL bits default to unselected, and are not writable. The reset value is always 0, and register writes have no effect. 0 HAS block is unselected for erase. 1 HAS block is selected for erase.

8.2.3.7 Address Register (ADR)

The Address Register (ADR) provides the first failing address in the event module failures (ECC, RWW, or FPEC) or the first address at which an ECC single-bit error correction occurs. ADR is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x0018 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SAD	TAD	0	0	0	0	0	0	0	0	0	AD20	AD19	AD18	AD17	AD16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-11. Address Register (ADR)—Code flash memory modules

Address: Base + 0x0018 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
W																
Reset	0	0	0	0	0	0	0	0	0	0[1]	0[1]	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0[1]	0	0

Figure 8-12. Address Register (ADR)—Data flash memory module

1 These bits are implemented in the data flash memory module, but not in the code flash memory modules.

Table 8-11. ADR field descriptions

Field	Description
SAD	Shadow Address When this read-only bit is high, the address indicated by AD[20:3] belongs to the shadow sector. **Note:** The SAD bit is implemented in the code flash memory modules, but not in the data flash memory module.
TAD	Test Address When this read-only bit is high, the address indicated by AD[20:3] belongs to the Test Sector. **Note:** The TAD bit is implemented in the code flash memory modules, but not in the data flash memory module.
AD[22:2]	Address 22–2 The AD bitfield provides the first failing address in the event of ECC error (MCR[EER] set) or the first failing address in the event of RWW error (MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (MCR[PEG] cleared). The AD bitfield also provides the first address at which an ECC single-bit error correction occurs (if MCR[EDC] is set), if the device is configured to show this feature in the UT0[SBCE] bitfield. The ECC double-bit error detection takes the highest priority, followed by the RWW error, the FPEC error and the ECC single-bit error correction. When accessed, the ADR provides the address related to the first event occurred with the highest priority. The priorities between these 4 possible events is summarized in the following table. **Priority level** / **Error flag** / **ADR content** 1 / MCR[EER] = 1 / Address of first ECC double-bit error detection 2 / MCR[RWE] = 1 / Address of first RWW Error 3 / MCR[PEG] = 0 / Address of first FPEC Error 4 / MCR[EDC] = 1 / Address of first ECC Single-bit error correction In User mode, the ADR register is read-only. **Note:** Bits AD22, AD21, and AD2 are implemented in the data flash memory module, but not in the code flash memory modules.

8.2.3.8 Platform Flash Configuration Register 0 (PFCR0)

The Platform Flash Configuration Register 0 (PFCR0) defines the configuration associated with flash memory banks 0 and 2. Collectively, this corresponds to both code flash memory modules (CFM0 and CFM1) and the operating configuration defined by certain fields applies to both memory banks. Additionally, it includes fields that provide specific information for the two separate AHB ports (p0 and p1). The register is described in Figure 8-13 for LSM operation of the MPC5675K and in Figure 8-14 for DPM operation of the MPC5675K and in Table 8-12.

NOTE

PFCR0 is implemented in the code flash memory modules, but not in the data flash memory module.

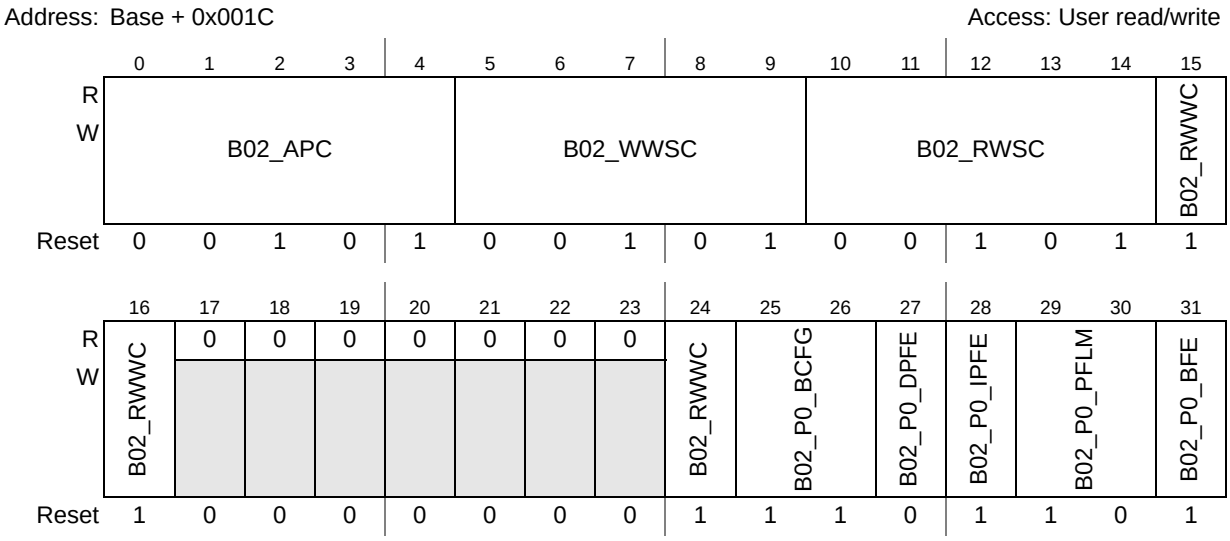


Figure 8-13. Platform Flash Configuration Register 0 (PFCR0)—LSM

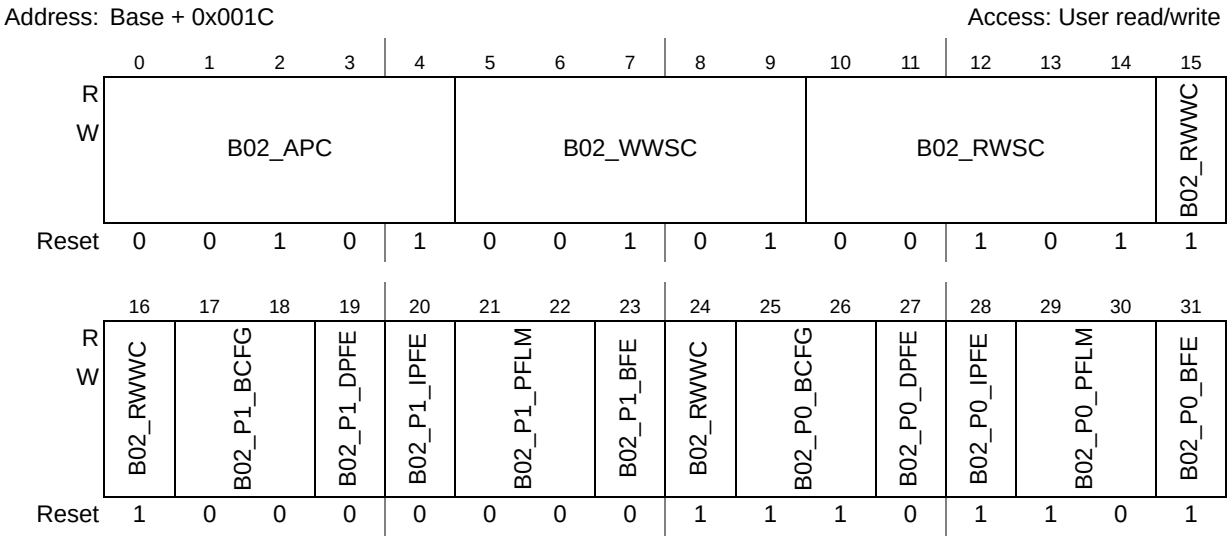


Figure 8-14. Platform Flash Configuration Register 0 (PFCR0)—DPM

Table 8-12. PFCR0 field descriptions

Field	Description
B02_APC	Bank0+2 Address Pipelining Control **Note:** The content of this bitfield must be set to the same value as B02_RWSC. This field controls the number of cycles between flash memory array access requests. This field must be set to a value appropriate to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. . . . 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.
B02_WWSC	Bank0+2 Write Wait State Control **Note:** Any number of wait states, including 0, are supported. This field controls the number of wait-states to be added to the flash memory array access time for writes. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. . . . 111111 31 additional wait-states are added.
B02_RWSC	Bank0+2 Read Wait State Control This field controls the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. Please refer to the device datasheet for supported wait state configurations depending on the device operating frequency. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 00000 No additional wait-states are added. 00001 1 additional wait-state is added. 00010 2 additional wait-states are added. . . . 111111 31 additional wait-states are added. **Note:** The content of this bitfield must be set to the same value as B02_APC.

Table 8-12. PFCR0 field descriptions (continued)

Field	Description
B02_RWWC	Bank0+2 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation. **Note:** The bits in this bitfield are not contiguous within the register. 0*xx* Reserved. This bit combination causes a malfunction. **Note:** Under certain circumstances, an expected AHB error on Read While Write (RWW) event is not handled properly. The resultant behavior is a failure to issue the AHB error. To a lesser extent, the bug can manifest as a system hang. The recommended workaround is to avoid configuring the flash memory controller to terminate RWW attempts with an error response. This restriction applies across all flash memory banks. The error cases are: — A 64-bit read access to optimized 32-bit data flash memory coinciding with the start of a program/erase operation may result in failure to terminate the read with error. — The start of a program/erase operation coinciding with the AHB data phase of a read request that has not yet been presented to the flash memory array may result in a system hang. The workaround is to avoid using PFCR0[B02_RWWC] = 0*xx*. 111 Generate a bus stall for a read-while-write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. 110 Generate a bus stall for a read-while-write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 101 Generate a bus stall for a read-while-write/erase, enable the operation abort, disable the abort notification interrupt. 100 Generate a bus stall for a read-while-write/erase, enable the operation abort and the abort notification interrupt.
B02_P1_BCFG	Bank0+2, Port 1 Page Buffer Configuration This field controls the configuration of the four page buffers in the PFLASH controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least recently used buffer within the group and the just-fetched entry then marked as most recently used. If the flash memory access is for the next-sequential line, the buffer is not marked as most recently used until the given address produces a buffer hit. 00 All four buffers are available for any flash memory access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0, 1, and 2 allocated for instruction fetches and buffer 3 for data accesses.
B02_P1_DPFE	Bank0+2, Port 1 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. 0 No prefetching is triggered by a data read access. 1 If page buffers are enabled (B02_P1_BFE = 1), prefetching is triggered by any data read access.

Table 8-12. PFCR0 field descriptions (continued)

Field	Description
B02_P1_IPFE	Bank0+2, Port 1 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. 0 No prefetching is triggered by an instruction fetch read access. 1 If page buffers are enabled (B02_P1_BFE = 1), prefetching is triggered by any instruction fetch read access.
B02_P1_PFLM	Bank0+2, Port 1 Prefetch Limit This field controls the prefetch algorithm used by the PFLASH controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. 00 No prefetching is performed. 01 The referenced line is prefetched on a buffer miss, that is, *prefetch on miss*. 1*x* The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, *prefetch on miss or hit*.
B02_P1_BFE	Bank0+2, Port 1 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.
B02_P0_BCFG	Bank0+2, Port 0 Page Buffer Configuration This field controls the configuration of the four page buffers in the PFLASH controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least recently used buffer within the group and the just-fetched entry then marked as most recently used. If the flash memory access is for the next-sequential line, the buffer is not marked as most recently used until the given address produces a buffer hit. 00 All four buffers are available for any flash memory access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0, 1, and 2 allocated for instruction fetches and buffer 3 for data accesses.
B02_P0_DPFE	Bank0+2, Port 0 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. 0 No prefetching is triggered by a data read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access.
B02_P0_IPFE	Bank0+2, Port 0 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. 0 No prefetching is triggered by an instruction fetch read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access.

Table 8-12. PFCR0 field descriptions (continued)

Field	Description
B02_P0_PFLM	Bank0+2, Port 0 Prefetch Limit This field controls the prefetch algorithm used by the PFLASH controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. 00 No prefetching is performed. 01 The referenced line is prefetched on a buffer miss, that is, *prefetch on miss*. 1*x* The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, *prefetch on miss or hit*.
B02_P0_BFE	Bank0+2, Port 0 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

NOTE

Under certain circumstances, an expected AHB error on Read While Write (RWW) event is not handled properly. The resultant behavior is a failure to issue the AHB error. To a lesser extent, the bug can manifest as a system hang. The recommended workaround is to avoid configuring the flash memory controller to terminate RWW attempts with an error response. This restriction applies across all flash memory banks. The error cases are:

— A 64-bit read access to optimized 32-bit data flash memory coinciding with the start of a program/erase operation may result in failure to terminate the read with error.

— The start of a program/erase operation coinciding with the AHB data phase of a read request that has not yet been presented to the flash memory array may result in a system hang.

The workaround is to avoid PFCR0[B02_RWWC] = 0*xx* should be avoided.

8.2.3.9 Platform Flash Configuration Register 1 (PFCR1)

The Platform Flash Configuration Register 1 (PFCR1) defines the configuration associated with flash memory bank 1. This corresponds to the data flash memory module 0. The register is described in Figure 8-15 for LSM operation of the MPC5675K and in Figure 8-16 for DPM operation of the MPC5675K and in Table 8-13.

NOTE

PFCR1 is implemented in the code flash memory modules, but not in the data flash memory module.

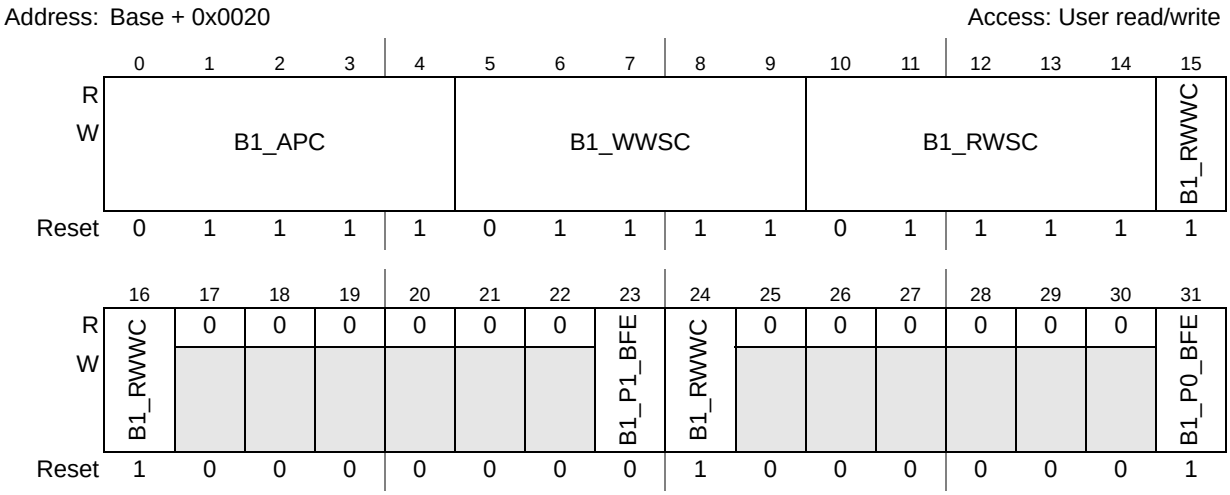


Figure 8-15. Platform Flash Configuration Register 1 (PFCR1)—LSM

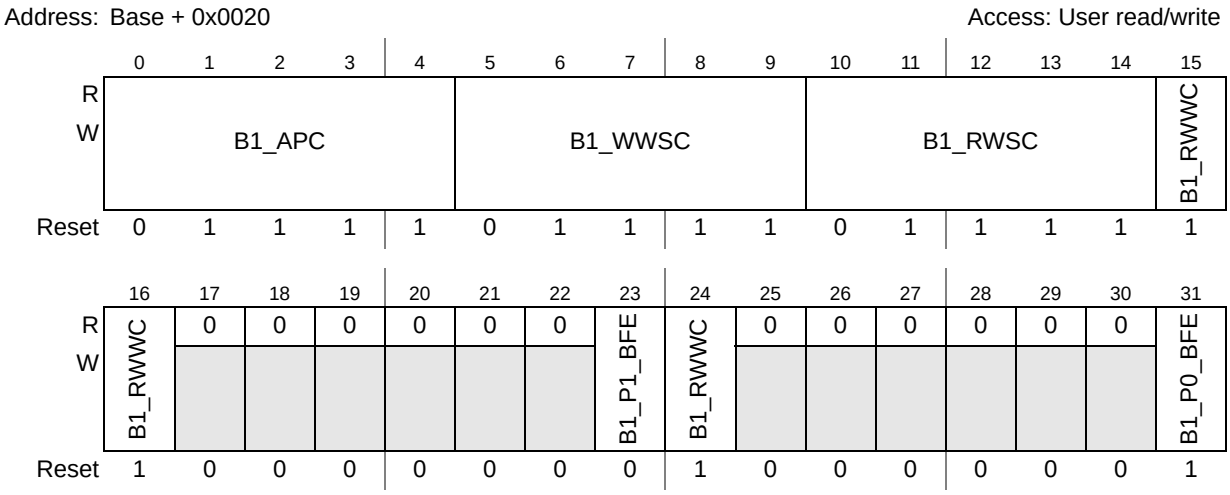


Figure 8-16. Platform Flash Configuration Register 1 (PFCR1)—DPM

Table 8-13. PFCR1 field descriptions

Field	Description
B1_APC	Bank1 Address Pipelining Control **Note:** The content of this bitfield must be set to the same value as B1_RWSC. This field controls the number of cycles between flash memory array access requests. This field must be set to a value appropriate to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. . . . 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.

Table 8-13. PFCR1 field descriptions (continued)

Field	Description
B1_WWSC	Bank1 Write Wait State Control **Note:** Any number of wait states, including 0, are supported. This field controls the number of wait-states to be added to the flash memory array access time for writes. 00000 No additional wait-states are added. 00001 1 additional wait-state is added. 00010 2 additional wait-states are added. . . . 111111 31 additional wait-states are added.
B1_RWSC	Bank1 Read Wait State Control This field controls the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. Please refer to the device datasheet for supported wait state configurations depending on the device operating frequency. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. 00000 No additional wait-states are added. 00001 1 additional wait-state is added. 00010 2 additional wait-states are added. . . . 111111 31 additional wait-states are added. **Note:** The content of this bitfield must be set to the same value as B1_APC.

Table 8-13. PFCR1 field descriptions (continued)

Field	Description
B1_RWWC	Bank1 Read-While-Write Control This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation. **Note:** The bits in this bitfield are not contiguous within the register. 0*xx* Reserved. This bit combination causes a malfunction. **Note:** Under certain circumstances, an expected AHB error on Read While Write (RWW) event is not handled properly. The resultant behavior is a failure to issue the AHB error. To a lesser extent, the bug can manifest as a system hang. The recommended workaround is to avoid configuring the flash memory controller to terminate RWW attempts with an error response. This restriction applies across all flash memory banks. The error cases are: — A 64-bit read access to optimized 32-bit data flash memory coinciding with the start of a program/erase operation may result in failure to terminate the read with error. — The start of a program/erase operation coinciding with the AHB data phase of a read request that has not yet been presented to the flash memory array may result in a system hang. The workaround is to avoid using PFC1[B1_RWWC] = 0*xx*. 111 Generate a bus stall for a read-while-write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. 110 Generate a bus stall for a read-while-write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 101 Generate a bus stall for a read-while-write/erase, enable the operation abort, disable the abort notification interrupt. 100 Generate a bus stall for a read-while-write/erase, enable the operation abort and the abort notification interrupt.
B1_P1_BFE	Bank1, Port 1 Buffer Enable This bit enables or disables read hits from the 32-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits.
B1_P0_BFE	Bank1, Port 0 Buffer Enable This bit enables or disables read hits from the 32-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits.

8.2.3.10 Platform Flash Access Protection Register (PFAPR)

The Platform Flash Access Protection Register (PFAPR) controls read and write accesses to the flash memory based on system master number. Prefetching capabilities are defined on a per-master basis. Please see Table 9-1 in Chapter 9, Multi-Layer AHB Crossbar Switch (XBAR). This register also defines the

arbitration mode between the 2 AHB ports for the PFLASH2P_LCA. The register is described in Figure 8-17 and Table 8-14.

NOTE

PFAPR is implemented in the code flash memory modules, but not in the data flash memory module.

NOTE: PFAPR Default Value

The contents of the PFAPR register are loaded from offset location 0x3E00 of the shadow region in CFM0 at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word location at address offset 0x3E00 of the shadow region in CFM0 must be programmed using the normal sequence of operations.

NOTE: PFAPR Core Access Rights to Flash Memory—System Boot

Because an improper reset value for PFAPR configured via the data in the shadow sector (see note above) can prevent the MPC5675K from booting, special attention must be paid to the proper configuration of this value in the shadow sector. It is important that this default value in the shadow sector grants read access to Core_0 from the boot flash memory location.

Address: Base + 0x024 Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	–	–	–	–	–	–	ARBM		M7PFD	M6PFD	M5PFD	M4PFD	M3PFD	M2PFD	M1PFD	M0PFD
W																
Reset[1]	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
W																
Reset[1]	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 8-17. Platform Flash Access Protection Register (PFAPR)

[1] The reset value depends on the content of the corresponding location (offset 0x3E00) in the shadow sector of CFM0.

Table 8-14. PFAPR field descriptions

Field	Description
ARBM	Arbitration mode This 2-bit field controls the arbitration for flash memory controllers supporting two AHB ports. The port arbitration mode is used only when accesses from the two AHB ports attempt to simultaneously reference the same flash memory bank. Simultaneous references to different memory banks are processed concurrently. 00 Fixed priority arbitration with AHB p0 > p1. 01 Fixed priority arbitration with AHB p1 > p0. 1*x* Round-robin arbitration. **Note:** These bits are configurable in DPM only. In LSM, they are hardwired to allow a proper lock step operation between Flash Controller 0 and Flash Controller 1. In LSM, the flash memory controllers appear as a single port controller.
M*x*PFD	Master x Prefetch Disable (*x* = 0,1,2,...,7) These bits control whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCRn[B02_Px_DPFE, B02_Px_IPFE, Bx_Py_BFE] bits. 0 Prefetching may be triggered by this master. 1 No prefetching may be triggered by this master.
M*x*AP	Master x Access Protection (x = 0,1,2,...,7) These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. 00 No accesses may be performed by this master. 01 Only read accesses may be performed by this master. 10 Only write accesses may be performed by this master. 11 Both read and write accesses may be performed by this master.

8.2.3.11 User Test 0 Register (UT0)

The User Test 0 Register (UT0) gives the user of the flash memory module the ability to perform test features on the flash memory. The User Test 0 Register allows control of the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE and DSI[7:0] of UT0 are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. UT0 is implemented on CFM0, CFM1, and DFM0.

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	SBCE	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W									RES							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 8-18. User Test 0 Register (UT0)—Code flash memory modules

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	0	0	0	0	0	0	0	0	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W									RES							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 8-19. User Test 0 Register (UT0)—Data flash memory module

Table 8-15. UT0 field descriptions

Field	Description
UTE	User Test Enable This status bit shows when User Test is enabled. This bit is not writable to a 1, but may be cleared. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. For UTE the password 0xF9F9_9999 must be written to the UT0 register. **Note:** For code flash memory modules: All bits in UT0–2 and UMISR0–4 are locked when this bit is 0. **Note:** For data flash memory module: All bits in UT0–1 and UMISR0–1 are locked when this bit is 0.
SBCE	Single Bit Correction Enable This bit, when high, enables to flag the informations about any eventual ECC single-bit correction in the Flash array (MCR[EDC] and MCR[ADR]). **Note:** This bit is implemented in the code flash modules, but not in the data flash module.

Table 8-15. UT0 field descriptions (continued)

Field	Description
DSI[7:0]	Data Syndrome Input 6–0 (Read/Write) These bits represents the input of Syndrome bits of ECC logic used in the ECC logic check. The DSI6–0 correspond to the 7 syndrome bits on a single word. These bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 The syndrome bit is forced at 0. 1 The syndrome bit is forced at 1. **Note:** The DSI7 bit is implemented in the code flash modules, but not in the data flash module.
RES	Reserved Reserved bit. Must always be programmed to 0.
X	X This bit can be written and its value can be read back, but no function is associated. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.
MRE	Margin Read Enable MRE enables margin reads to be done. This bit, combined with MRV, enables start of FPEC margin reads respect to erased or programmed value. Outputs of margin read are: checksum values in UMISR0–1. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Margin reads are not enabled, all reads are User mode reads. 1 Margin reads are enabled.
MRV	Margin Read Value If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV=1) or to a programmed level (MRV=0). This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Zero’s (programmed) margin reads are requested (if MRE = 1). 1 One’s (erased) margin reads are requested (if MRE = 1).
EIE	ECC Data Input Enable EIE enables the ECC logic check operation to be done. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. If this bit is high together with bit MRE, the Read Reset Operation is selected. 0 ECC logic check is not enabled. 1 ECC logic check is enabled.
AIS	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Array Integrity sequence is proprietary sequence. 1 Array Integrity sequence is sequential.

Table 8-15. UT0 field descriptions (continued)

Field	Description
AIE	Array Integrity Enable AIE set to 1 starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0–1) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR[ERS], MCR[PGM] and MCR[EHV] are all low. 0 Array Integrity Checks are not enabled. 1 Array Integrity Checks are enabled.
AID	Array Integrity Done AID is cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID is set to indicate that the Array Integrity Check is complete. At this time the MISR (UMISR0–1) can be checked. 0 Array Integrity Check is on-going. 1 Array Integrity Check is done.

8.2.3.12 User Test 1 Register (UT1)

The User Test 1 Register (UT1) allows the ability to enable the checks on the ECC logic related to a 32-bit word. The UT1 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-20. User Test 1 Register (UT1)

Table 8-16. UT1 field descriptions

Field	Description
DAI[31:00]	Data Array Input 31–0 0 The array bit is forced at 0. 1 The array bit is forced at 1. **Note:** For code flash memory modules: These bits represents the input of even words of ECC logic used in the ECC logic check. The DAI31–00 correspond to the 32 array bits **Note:** For data flash memory module: These bits represents the input of 32-bit words of ECC logic used in the ECC logic check. The DAI31–00 correspond to the 32 array bits of a word.

8.2.3.13 User Test 2 Register (UT2)

The User Test 2 Register allows to enable the checks on the ECC logic related to the 32 MSB of the double word. UT2 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

NOTE

UT2 is implemented in the code flash memory modules, but not in the data flash memory module.

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-21. User Test 2 Register (UT2)—Code flash memory modules

Table 8-17. UT2 field descriptions

Field	Description
DAI[63:32]	Data Array Input 63–32 These bits represents the input of odd words of ECC logic used in the ECC logic check. The DAI[63:32] correspond to the 32 array bits representing Word 1 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

8.2.3.14 User Multiple Input Signature Register 0 (UMISR0)

The User Multiple Input Signature Register 0 (UMISR0) provides a means to evaluate the array integrity. UMISR0 is implemented on CFM0, CFM1, and DFM0.

For the code flash memory modules, UMISR0 represents bits 31–0 of the whole 144-bit word (2 double words including ECC for each double word). For the data flash memory module, UMISR0 represents bits 31–0 of the word.

The UMISR0 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

Address: Base + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-22. User Multiple Input Signature Register 0 (UMISR0)—Code flash memory modules

Address: Base + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS31	MS30	MS29	MS28	MS27	MS26	MS25	MS24	MS23	MS22	MS21	MS20	MS19	MS18	MS17	MS16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS15	MS14	MS13	MS12	MS11	MS10	MS09	MS08	MS07	MS06	MS05	MS04	MS03	MS02	MS01	MS00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-23. User Multiple Input Signature Register 0 (UMISR0)—Data flash memory module

NOTE

The difference in the bit numbering reflects the 128-bit nature of the code flash memory modules vs. the 32-bit nature of the data flash memory module.

Table 8-18. UMISR0 field descriptions

Field	Description
MS[031:000] MS[31:00]	Multiple input Signature 031–000 (31–00). These bits represents the MISR value obtained accumulating the bits 31–0 of all the pages read from the flash memory. The MS can be seeded to any value by writing the UMISR0 register.

8.2.3.15 User Multiple Input Signature Register 1 (UMISR1)

UMISR1 provides a means to evaluate the array integrity. UMISR1 is implemented on CFM0, CFM1, and DFM0.

On the code flash memory modules, UMISR1 represents bits 63–32 of the whole 144-bit word (2 double words including ECC for each double word). On the data flash memory module, UMISR1 represents the ECC bits of the 32-bits word.

The MS can be seeded to any value by writing to UMISR1.

UMISR1 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

Address: Base + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-24. User Multiple Input Signature Register 1 (UMISR1)—Code flash memory modules

Address: Base + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	MS43	MS42	0	0	0	MS38	MS37	MS36	MS35	MS34	MS33	MS32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-25. User Multiple Input Signature Register 1 (UMISR1)—Data flash memory module

NOTE

The difference in the bit numbering reflects the 128-bit nature of the code flash memory modules vs. the 32-bit nature of the data flash memory module.

Table 8-19. UMISR1 field descriptions

Field	Description
MS[063:032]	**Note:** For code flash memory modules only: Multiple input Signature 063–032. These bits represents the MISR value obtained accumulating the bits 63–32 of all the pages read from the flash memory.
MS43	**Note:** For data flash memory module only: Multiple input Signature 43. These bits represents the MISR value obtained accumulating double ECC error detection.
MS42	**Note:** For data flash memory module only: Multiple input Signature 42. These bits represents the MISR value obtained accumulating single ECC error detection.

Table 8-19. UMISR1 field descriptions

Field	Description
MS[38:32]	**Note:** For data flash memory module only: Multiple input Signature 38–32. These bits represents the MISR value obtained accumulating 7 ECC bits for the word.

8.2.3.16 User Multiple Input Signature Register 2 (UMISR2)

UMISR2 provides a means to evaluate the array integrity. UMISR2 represents the bits 95–64 of the whole 144-bit word (2 double words including ECC for each double word).

The MS can be seeded to any value by writing the UMISR2 register.

UMISR2 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

NOTE

UMISR2 is implemented in the code flash memory modules, but not in the data flash memory module.

Address: Base + 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-26. User Multiple Input Signature Register 2 (UMISR2)—Code flash memory modules

Table 8-20. UMISR2 field descriptions

Field	Description
MS[095:064]	Multiple input Signature 095–064. These bits represents the MISR value obtained accumulating the bits 95–64 of all the pages read from the code flash memory.

8.2.3.17 User Multiple Input Signature Register 3 (UMISR3)

UMISR3 provides a means to evaluate the array integrity. UMISR3 represents the bits 127–96 of the whole 144-bit word (2 double words including ECC for each double word).

The MS can be seeded to any value by writing the UMISR3 register.

UMISR3 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

NOTE

UMISR3 is implemented in the code flash memory modules, but not in the data flash memory module.

Address: Base + 0x0054 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-27. User Multiple Input Signature Register 3 (UMISR3)—Code flash memory modules

Table 8-21. UMISR3 field descriptions

Field	Description
MS[127:096]	Multiple input Signature 127–096. These bits represents the MISR value obtained accumulating the bits 127–64 of all the pages read from the code flash memory.

8.2.3.18 User Multiple Input Signature Register 4 (UMISR4)

UMISR4 provides a means to evaluate the array integrity.

UMISR4 represents the ECC bits of the whole 144-bit word (2 double words including ECC for each double word). Bits 8–15 are ECC bits for the odd double word and bits 24–31 are the ECC bits for the even double word. Bits 4–5 and 20–21 of UMISR4 are the double and single ECC error detection for the odd and even double word, respectively.

The MS can be seeded to any value by writing the UMISR4 register. UMISR4 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.

NOTE

UMISR4 is implemented in the code flash memory modules, but not in the data flash memory module.

Address: Base + 0x0058 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	MS155	MS154	0	0	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	MS139	MS138	0	0	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-28. User Multiple Input Signature Register 4 (UMISR4)—Code flash memory modules

Table 8-22. UMISR4 field descriptions

Field	Description
MS155	Multiple input Signature 155. This bit represents the MISR value obtained accumulating the double ECC error detection for the odd double word. The MS can be seeded to any value by writing the UMISR4 register.
MS154	Multiple input Signature 154. This bit represents the MISR value obtained accumulating the single ECC error detection for the odd double word. The MS can be seeded to any value by writing the UMISR4 register.
MS[151:144]	Multiple input Signature 151–144. These bits represents the MISR value obtained accumulating the 8 ECC bits for the odd double word. The MS can be seeded to any value by writing the UMISR4 register.
MS139	Multiple input Signature 139. This bit represents the MISR value obtained accumulating the double ECC error detection for the even double word. The MS can be seeded to any value by writing the UMISR4 register.
MS138	Multiple input Signature 138. This bit represents the MISR value obtained accumulating the single ECC error detection for the even double word. The MS can be seeded to any value by writing the UMISR4 register.
MS[135:128]	Multiple input Signature 135–128. These bits represents the MISR value obtained accumulating the 8 ECC bits for the even double word. The MS can be seeded to any value by writing the UMISR4 register.

8.2.4 Shadow sector

A shadow block is present in each code flash memory module, but not in the data flash memory module. The shadow block can be enabled by the BIU.

When the shadow space is enabled, all the operations are mapped to the shadow block. User mode program and erase of the shadow block are enabled only when MCR[PEAS] is high.

The shadow block can be locked or unlocked against program/erase with the LML[TSLK] and SLL[STSLK] bitfields.

Program of the shadow block has similar restriction as the array in terms of how ECC is calculated. Only one program is allowed per 64-bit ECC segment between erases. Erase of the shadow block is done similarly as an array erase.

The shadow block contains data that are needed for device features.

The first 8 KB of the shadow block is available for user-defined functions such as storing boot code, configuration words, or factory process codes.

The use of the shadow sector is detailed in Table 8-23.

Table 8-23. Shadow Sector Structure for Code Flash Module 0 (CFM0 B0SH)

Offset from FLASH_ SECTOR_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000–0x000F	Reserved			
0x0010–0x3DCF	User Area (DCF start address)	R/W	—	—
0x3DD0–0x3DD7	Reserved			
0x3DD8	Nonvolatile Private Censorship Password 0 (NVPWD0[3])	R/W	0x FEED_FACE	on page 203
0x3DDC	Nonvolatile Private Censorship Password 1 (NVPWD1[3])	R/W	0xCAFE_BEEF	on page 203
0x3DE0	Nonvolatile System Censorship Information 0 (NVSCI0[3])	R/W	0x55AA_55AA	on page 204
0x3DE4	Nonvolatile System Censorship Information 1 (NVSCI1[3])	R/W	0x55AA_55AA	on page 204
0x3DE8–0x3DFF	Reserved			
0x3E00	Nonvolatile Bus Interface Unit 2 (NVBIU2[3,4]) (Reset value for PFAPR)	R/W		
0x3E04	Nonvolatile Bus Interface Unit 2 (32 bits not used)	R/W		
0x3E08				
0x3E18	Nonvolatile User Options (NVUSRO[3])	R/W	0xFFBF_FFFF	on page 205
0x3E1C	Nonvolatile User Options (32 bits not used)	R/W		
0x3E20–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 Because the ECC boundary for this flash memory is 64-bit, eight bytes (starting from a 64-bit address boundary) shall always be updated to ensure a valid ECC signature.

4 See Section 8.2.3.10, Platform Flash Access Protection Register (PFAPR) for register details.

The following data is related to the shadow sector base addresses of the two code flash memory modules.

8.2.4.1 Nonvolatile Private Censorship Password 0 Register (NVPWD0)

The Nonvolatile Private Censorship Password 0 Register (NVPWD0) contains the 32 LSB of the password used to validate the censorship information contained in the NVSCI0–1 registers.

Address: Base + 3DD8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD31	PWD32	PWD29	PWD28	PWD27	PWD26	PWD25	PWD24	PWD23	PWD22	PWD21	PWD20	PWD19	PWD18	PWD17	PWD16
Reset	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD15	PWD14	PWD13	PWD12	PWD11	PWD10	PWD09	PWD08	PWD07	PWD06	PWD05	PWD04	PWD03	PWD02	PWD01	PWD00
Reset	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0

Figure 8-29. Nonvolatile Private Censorship Password 0 Register (NVPWD0)

Table 8-24. NVPWD0 field descriptions

Field	Description
PWD[31:00]	Password 31–00. The PWD[31:00] bits represent the 32 LSB of the private censorship password (0xFEED_FACE).

NOTE

These bits form the lower 32 bits of the Censorship password.

8.2.4.2 Nonvolatile Private Censorship Password 1 Register (NVPWD1)

The Nonvolatile Private Censorship Password 1 Register (NVPWD1) contains the 32 MSB of the password used to validate the censorship information contained in the NVSCI0–1 registers.

Address: Base + 3DDC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD63	PWD62	PWD61	PWD60	PWD59	PWD58	PWD57	PWD56	PWD55	PWD54	PWD53	PWD52	PWD51	PWD50	PWD49	PWD48
Reset	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD47	PWD46	PWD45	PWD44	PWD43	PWD42	PWD41	PWD40	PWD39	PWD38	PWD37	PWD36	PWD35	PWD34	PWD33	PWD32
Reset	1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1

Figure 8-30. Nonvolatile Private Censorship Password 1 Register (NVPWD1)

Table 8-25. NVPWD1 field descriptions

Field	Description
PWD[63:32]	Password 63–32. The PWD[63:32] bits represent the 32 MSB of the private censorship password (0xCAFE_BEEF).

NOTE

These bits form the upper 32 bits of the Censorship password.

8.2.4.3 Nonvolatile System Censoring Information 0 Register (NVSCI0)

The Nonvolatile System Censoring Information 0 Register (NVSCI0) stores the 32 LSB of the censorship control word of the device. The NVSCI0 is a nonvolatile register located in shadow sector. It is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently. The devices are delivered uncensored to the user.

Address: Base + 0x3DE0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC15	SC14	SC13	SC12	SC11	SC10	SC9	SC8	SC7	SC6	SC5	SC4	SC3	SC2	SC1	SC0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW15	CW14	CW13	CW12	CW11	CW10	CW9	CW8	CW7	CW6	CW5	CW4	CW3	CW2	CW1	CW0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Figure 8-31. Nonvolatile System Censoring Information 0 Register (NVSCI0)

Table 8-26. NVSCI0 field descriptions

Field	Description
SC[15:0]	Serial Censorship control word 15–0. These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW). If SC[15:0] = 0x55AA and NVSCI1 = NVSCI0 the Public Access is disabled. If SC[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0 the Public Access is enabled.
CW[15:0]	Censorship control Word 15–0. These bits represent the 16 LSB of the Censorship Control Word (CCW). If CW[15:0] = 0x55AA and NVSCI1 = NVSCI0 the Censored mode is disabled. If CW[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0 the Censored mode is enabled.

8.2.4.4 Nonvolatile System Censoring Information 1 Register (NVSCI1)

The Nonvolatile System Censoring Information 1 Register (NVSCI1) stores the 32 MSB of the censorship control word of the device. The NVSCI1 is a nonvolatile register located in shadow sector. It is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

Address: Base + 0x3DE4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SC31	SC30	SC29	SC28	SC27	SC26	SC25	SC24	SC23	SC22	SC21	SC20	SC19	SC18	SC17	SC16
W																
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CW31	CW30	CW29	CW28	CW27	CW26	CW25	CW24	CW23	CW22	CW21	CW20	CW19	CW18	CW17	CW16
W																
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Figure 8-32. Nonvolatile System Censoring Information 1 Register (NVSCI1)

Table 8-27. NVSCI1 field descriptions

Field	Description
SC[31:16]	Serial Censorship control word 31–16. These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC[31:16] = 0x55AA and NVSCI1 = NVSCI0 the Public Access is disabled. If SC[31:16] ≠ 0x55AA or NVSCI1 ≠ NVSCI0 the Public Access is enabled.
CW[31:16]	Censorship control Word 31–16. These bits represent the 16 MSB of the Censorship Control Word (CCW). If CW[31:16] = 0x55AA and NVSCI1 = NVSCI0 the Censored mode is disabled. If CW[31:16] ≠ 0x55AA or NVSCI1 ≠ NVSCI0 the Censored mode is enabled.

8.2.4.5 Nonvolatile User Options Register (NVUSRO)

The Nonvolatile User Options Register (NVUSRO) contains configuration information for the user application. The NVUSRO register is a 64-bit register, the 32 most significant bits of which (bits 63–32) are “don’t care” bits. However, the content of bits 63–32 influence the ECC signature of this 64-bit word.

This register is mirrored in the SSCM_UOPS register. See Section 49.3.1.8, User Option Status (SSCM_UOPS) Register. For more information on bits 20:31, please see Section 25.7.3, FCCU Configuration Register (FCCU_CFG).

Address: Test flash base + 0x3E18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SWT	XMA	1	1	1	1	FCR	CR		LSM	0	0	1	1	1	1
W																
Reset[1]	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	CM	SM	PS	FOM			FOP					
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 8-33. Nonvolatile User Options Register (NVUSRO)

[1] The reset value shows the preprogrammed value of the device. The content can be changed by the customer.

Table 8-28. NVUSRO field description

Field	Description
SWT	Software watchdog timer default enable 0 SWT off 1 SWT on **Note:** In DPM, applies only to SWT_0; SWT_1 is always off until enabled by software.
XMA	XOSC oscillator margin 0 For 4 MHz quartz oscillator connected to XOSC pins 1 For 8, 16, or 40 MHz quartz oscillator connected to XOSC pins
FCR	FEC clock ratio Selects clock ratio between platform clock and FEC clock (Ethernet Module) 0 FEC CLK = SYS_CLK. Use when SYS_CLK < 100 MHz 1 FEC CLK = SYS_CLK / 2. Use when SYS_CLK ≥ 100 MHz
CR	Clock ratio This field selects the clock ratio between core, platform, and flash clock. 00 Reset Phase Configuration 01 1:1 mode 10 1:3 mode 11 1:2 mode (default after reset) See Section 13.3, System clock dividers, for more information.
LSM	Lock Step Mode This bit selects the mode of operation between LSM and DPM. 0 DPM 1 LSM
CM	Configuration mode 0 Configuration labeling: a specific FCCU_F[0:1] configuration is assigned in CONFIG state. 1 Configuration transparency: the FCCU_F[0:1] protocol is the same in CONFIG and NORMAL states.
SM	Switching mode 0 FCCU_F[0:1] protocol (dual-rail, time-switching) slow switching mode 1 FCCU_F[0:1] protocol (dual-rail, time-switching) fast switching mode SM has no effect on the bi-stable protocol.
PS	Polarity selection 0 FCCU_F[1] active high, FCCU_F[0] active high 1 FCCU_F[1] active low, FCCU_F[0] active low
FOM	Fault output mode selection 000 Dual-Rail (default state) [fccu_eout[1:0]= outputs] 001 Time Switching [fccu_eout[1:0]= outputs] 010 Bi-Stable [fccu_eout[1:0]= outputs] 011 Reserved 100 Reserved 101 Test0 [FCCU_F[0]= input, FCCU_F[1]= output] 110 Test1 [FCCU_F[0]= output, FCCU_F[1]= output] 111 Test2 [FCCU_F[0]= output, FCCU_F[1]= input] The output level of the FCCU_F[0:1] pins in the test modes is controlled in the FCCU I/O Control Register (FCCU_EINOUT). See Section 25.7.16, FCCU I/O Control Register (FCCU_EINOUT). **Note:** In Test*[1:3]* mode, a simple double-stage resynchronization stage is used to resynchronize the FCCU_F[0:1] input/outputs on the system/safe clock.

Table 8-28. NVUSRO field description (continued)

Field	Description
FOP	Fault output prescaler FOP defines the prescaler setting used to generate the FCCU_F[0:1] protocol frequency. 00 0000 Input clock frequency (RC_{16MHz} clock) is divided by 2048. 00 0001 Input clock frequency (RC_{16MHz} clock) is divided by 4096. 00 0010 Input clock frequency (RC_{16MHz} clock) is divided by 6144. 00 0011 Input clock frequency (RC_{16MHz} clock) is divided by 8192. 00 0100 Input clock frequency (RC_{16MHz} clock) is divided by 10240. 00 0101 Input clock frequency (RC_{16MHz} clock) is divided by 12288. 00 0110 Input clock frequency (RC_{16MHz} clock) is divided by 14336. 00 0111 Input clock frequency (RC_{16MHz} clock) is divided by 16384. 00 1000 Input clock frequency (RC_{16MHz} clock) is divided by 18432 (This is the reset value, i.e., ~ 868 Hz is the out-of-reset frequency of the FCCU_F[0:1] pins). This equation gives the FCCU_F[0:1] frequency: $$Fccu0F_{freq} = RC_{16MHz}/(1024 \times (FOP + 1) \times 2)$$

8.2.5 Test sector

The test flash memory block:

- Exists outside the normal address space
- Has a base address of 0x0040_0000
- Is programmed, erased and read independently of the other blocks

The independent test flash memory block is reserved to store the nonvolatile information related to redundancy, configuration, and protection.

NVM test flash memory contains data related to RAM repair, ADC self-test calibration thresholds, and STCU Logic BIST and Memory BIST operation. Test flash memory is programmed at the factory and cannot be changed by the user.

NVM shadow flash supports user configuration of the STCU Memory BIST and Logic BIST operation. The user can bypass BIST, enable BIST, and configure Stay In Reset upon BIST fault detection.

See Chapter 50, Self-Test Control Unit (STCU), for details on how NVM (Test and Shadow Flash), SSCM, and STCU operate together to drive BIST execution.

The use of reserved test flash memory sector is detailed in Table 8-29.

Table 8-29. Test flash memory structure

Name	Description	Addresses	Size
—	User Reserved	0x40_3D00–0x40_3DE7	232 bytes
NVLML	NV Low/Mid Address Space Block Locking Register	0x40_3DE8–0x40_3DEF	8 bytes
—	Reserved	0x40_3DF0–0x40_3DF7	8 bytes
NVSLL	NV Secondary Low/Mid Add Space Block Lock Register	0x40_3DF8–0x40_3DFF	8 bytes
—	User Reserved	0x40_3E00–0x40_3EFF	256 bytes

Table 8-29. Test flash memory structure (continued)

Name	Description	Addresses	Size
—	Reserved	0x40_3F00–0x40_3FB7	184 bytes
—	Reserved	0x40_3FB8–0x40_3FBF	8 bytes

The test flash memory block can be enabled by the BIU.

When the test space is enabled, program operations to the test block are allowed from 0x40_3D00 to 0x40_3EFF (User/Lock area is One-Time Programmable). User mode programming of the test block is enabled only when MCR[PEAS] is high.

The test flash memory block may be locked/unlocked against programming by using the LML[TSLK] and SLL[STSLK] registers. Erase of the test flash memory block is always locked in user mode.

Programming of the test flash memory block has similar restrictions as the array in terms of how ECC is calculated. Only one program is allowed per 32-bit ECC segment, unless ECC evaluation is disabled on the test flash memory block. The test flash memory block contains data that are needed for flash memory module or device features. This information is summarized in Table 8-30.

Table 8-30. Test flash memory information

Address	Word name	Function	Note
0x0000	TSENS1_CAL W1/W2	Temperature sensor Calibration data W1 and W2	—
0x0004	TSENS1_CAL W3/W4	Temperature sensor Calibration data W3 and W4	—
0x0008	Reserved		
0x000C	Reserved		
0x0010	ADC0_CAL W1	ADC0 Self-Test calibration—RC algorithm	2 × 12-bit words (STAW3RH/L)
0x0014	ADC0_CAL W2	ADC0 Self-Test calibration—C algorithm step 0	2 × 12-bit words (STAW4RH/L)
0x0018	ADC0_CAL W3	ADC0 Self-Test calibration—C algorithm step 1 to N	2 × 12-bit words (STAW5RH/L)
0x001C	ADC0_CAL W4	ADC0 Self-Test calibration—S algorithm step 0–3.3 V	2 × 12-bit words (STAW0RH/L)
0x0020	ADC0_CAL W5	ADC0 Self-Test calibration—S algorithm step 0–5.0 V	2 × 12-bit words (STAW0RH/L)
0x0024	ADC0_CAL W6	ADC0 Self-Test calibration—S algorithm step 1–integer	2 × 12-bit words (STAW1ARH/L)
0x0028	ADC0_CAL W7	ADC0 Self-Test calibration—S algorithm step 1–float	2 × 12-bit words (STAW1BRH/L)
0x002C	ADC0_CAL W8	ADC0 Self-Test calibration—S algorithm step 2	1 × 12-bit word (STAW2R)
0x0030	ADC0 Reserved	ADC0 reserved	For future expansion

Table 8-30. Test flash memory information (continued)

Address	Word name	Function	Note
0x0034	ADC1_CAL W1	ADC1 Self-Test calibration—RC algorithm	2 × 12-bit words (STAW3RH/L)
0x0038	ADC1_CAL W2	ADC1 Self-Test calibration—C algorithm step 0	2 × 12-bit words (STAW4RH/L)
0x003C	ADC1_CAL W3	ADC1 Self-Test calibration—C algorithm step 1 to N	2 × 12-bit words (STAW5RH/L)
0x0040	ADC1_CAL W4	ADC1 Self-Test calibration—S algorithm step 0–3.3 V	2 × 12-bit words (STAW0RH/L)
0x0044	ADC1_CAL W5	ADC1 Self-Test calibration—S algorithm step 0–5.0 V	2 × 12-bit words (STAW0RH/L)
0x0048	ADC1_CAL W6	ADC1 Self-Test calibration—S algorithm step 1–integer	2 × 12-bit words (STAW1ARH/L)
0x004C	ADC1_CAL W7	ADC1 Self-Test calibration—S algorithm step 1–float	2 × 12-bit words (STAW1BRH/L)
0x0050	ADC1_CAL W8	ADC1 Self-Test calibration—S algorithm step 2	1 × 12-bit word (STAW2R)
0x0054	ADC1 Reserved	ADC1 Reserved	For future expansion
0x0058	PART ID1 Low	Plant and Lot Number ASCII format	—
0x005C	PART ID1 High	Plant and Lot Number ASCII format	—
0x0060	PART ID2	Wafer number and coordinates	—
0x0064	Reserved		
0x0068	Reserved		
0x006C	ADC2_CAL W1	ADC2 Self-Test calibration—RC algorithm	2 × 12-bit words (STAW3RH/L)
0x0070	ADC2_CAL W2	ADC2 Self-Test calibration—C algorithm step 0	2 × 12-bit words (STAW4RH/L)
0x0074	ADC2_CAL W3	ADC2 Self-Test calibration—C algorithm step 1 to N	2 × 12-bit words (STAW5RH/L)
0x0078	ADC2_CAL W4	ADC2 Self-Test calibration—S algorithm step 0–3.3 V	2 × 12-bit words (STAW0RH/L)
0x007C	ADC2_CAL W5	ADC2 Self-Test calibration—S algorithm step 0–5.0 V	2 × 12-bit words (STAW0RH/L)
0x0080	ADC2_CAL W6	ADC2 Self-Test calibration—S algorithm step 1–integer	2 × 12-bit words (STAW1ARH/L)
0x0084	ADC2_CAL W7	ADC2 Self-Test calibration—S algorithm step 1–float	2 × 12-bit words (STAW1BRH/L)
0x0088	ADC2_CAL W8	ADC2 Self-Test calibration—S algorithm step 2	1 × 12-bit word (STAW2R)
0x008C	ADC2 Reserved	ADC2 reserved	For future expansion

Table 8-30. Test flash memory information (continued)

Address	Word name	Function	Note
0x0090	ADC3_CAL W1	ADC3 Self-Test calibration—RC algorithm	2 × 12-bit words (STAW3RH/L)
0x0094	ADC3_CAL W2	ADC3 Self-Test calibration—C algorithm step 0	2 × 12-bit words (STAW4RH/L)
0x0098	ADC3_CAL W3	ADC3 Self-Test calibration—C algorithm step 1 to N	2 × 12-bit words (STAW5RH/L)
0x009C	ADC3_CAL W4	ADC3 Self-Test calibration—S algorithm step 0–3.3 V	2 × 12-bit words (STAW0RH/L)
0x00A0	ADC3_CAL W5	ADC3 Self-Test calibration—S algorithm step 0–5.0 V	2 × 12-bit words (STAW0RH/L)
0x00A4	ADC3_CAL W6	ADC3 Self-Test calibration—S algorithm step 1–integer	2 × 12-bit words (STAW1ARH/L)
0x00A8	ADC3_CAL W7	ADC3 Self-Test calibration—S algorithm step 1–float	2 × 12-bit words (STAW1BRH/L)
0x00AC	ADC3_CAL W8	ADC3 Self-Test calibration—S algorithm step 2	1 × 12-bit word (STAW2R)
0x00B0	ADC3 Reserved	ADC3 Reserved	For future expansion
0x00B4	Reserved		
0x00B8–0x03FF	Reserved		
0x0400	SSCM_START_WORD	Start Word for DCF protocol	—
0x0410	Reserved		
0x0420	Reserved		
0x0430	Reserved		
0x0440	Reserved		
0x0450	Reserved		
0x0460	Reserved		
0x0470	Reserved		
0x0480	Reserved		
0x0490	Reserved		
0x04A0	Reserved		
0x04B0	Reserved		
0x04C0	Reserved		
0x04D0	Reserved		
0x04E0	Reserved		
0x04F0	Reserved		

Table 8-30. Test flash memory information (continued)

Address	Word name	Function	Note
0x0500	Reserved		
0x0510–0x0A80	STCU_1–STCU_104	STCU Configuration (MBIST and LBIST)	—
0x0A90	SSCM_STOP	Stop Word for DCF protocol[1]	—
0x0A94–0x3FFF	Reserved		

1 SSCM_STOP word is located immediately after the last valid STCU Configuration word. Test flash memory configuration does not require full usage of the STCU_1 to STCU_n address space. For example, typically a test flash memory program consists of STCU Configuration words from address 0x0510 to 0x0A50, where 0x0A40 is the last valid STCU Configuration word and 0x0A50 contains the Stop Word for the DCF protocol. The remainder of the STCU configuration space (0x0A50 to 0x0A90) contains erased values of 0xFFFF_FFFF.

8.2.6 Functional description

8.2.6.1 Data flash memory module structure

The data flash memory module is designed for use in devices that require data nonvolatile memories for EE emulation. The data flash memory module is addressable by 32-bit word for program and for read.

The data flash memory module supports fault tolerance through Error Correction Code (ECC) and/or error detection. The ECC implemented within the data flash memory module corrects single-bit failures and detects double-bit failures.

The data flash memory module uses an embedded hardware algorithm implemented in the memory interface to program and erase the data flash memory core. Control logic that works with the software block enables, and software lock mechanisms, is included in the embedded hardware algorithm to guard against accidental program/erase. The hardware algorithm perform the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to improve data integrity and reliability. A programmed bit in the data flash memory module reads as logic level 0 (or low). An erased bit in the data flash memory module reads as logic level 1 (or high). Program and erase of the data flash memory module requires multiple system clock cycles to complete. The erase sequence may be suspended. The program and erase sequences may be aborted.

Since the data flash memory module is a slave module, it requires the code flash memory modules to be active. In other words, the code flash memory modules must not be in reset, in Disable mode, or in Sleep mode in order for the data flash memory module to be active.

The data flash memory module contains one module, composed of a single bank: Bank 0, normally used for code storage. No Read-While-Modify operations are possible.

The modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a user registers interface.

The read data bus is 32 bits wide, while the data flash memory registers are on a separate 32-bit wide bus.

The high voltages needed for program/erase operations are internally generated. Figure 8-34 shows the data flash memory module structure.

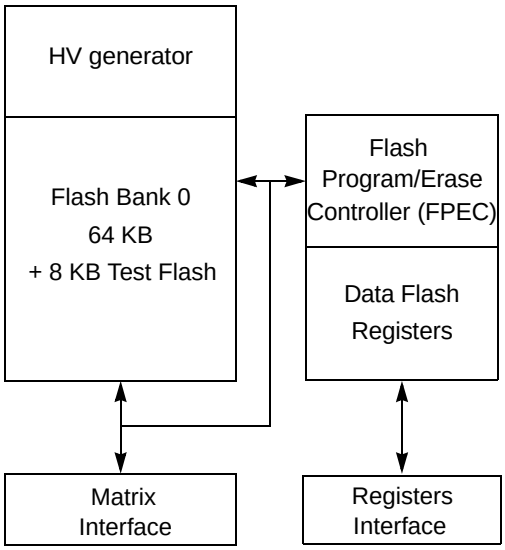


Figure 8-34. Data flash memory module structure

8.2.6.2 Code flash memory module structure

The code flash memory module is designed for use in devices that require high density nonvolatile memories with high speed read access.

The code flash memory modules are addressable by word (32 bits) or double word (64 bits) for program, and page (128 bits) for read. Reads done to a code flash memory module always return 128 bits, although read page buffering may be done in the device BIU.

Each read of a code flash memory module retrieves a page, or 4 consecutive words (128 bits) of information. The address for each word retrieved within a page differ from the other addresses in the page only by address bits (3:2). The code flash memory page read architecture easily supports both cache and burst mode at the BIU level for high speed read application. The code flash memory module supports fault tolerance through Error Correction Code (ECC) and/or error detection. The ECC implemented within the code flash memory module corrects single-bit failures and detects double-bit failures. The code flash memory module uses an embedded hardware algorithm implemented in the memory interface to program and erase the code flash memory core.

Control logic that works with the software block enables, and software lock mechanisms, is included in the embedded hardware algorithm to guard against accidental program/erase. The hardware algorithm perform the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to improve data integrity and reliability. A programmed bit in the code flash memory module reads as logic level 0 (or low).

An erased bit in the code flash memory module reads as logic level 1 (or high). Program and erase of the code flash memory module requires multiple system clock cycles to complete. The erase sequence may be suspended. The program and erase sequences may be aborted.

The code flash memory modules contain the matrix modules normally used for code storage. Read-While-Modify operations are not possible.

The modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a user registers interface.

The read data bus is 2 × 128 bits wide, while the code flash memory registers are on a separate 32-bit wide bus.

The high voltages needed for program/erase operations are internally generated. Figure 8-35 shows the code flash memory module structure.

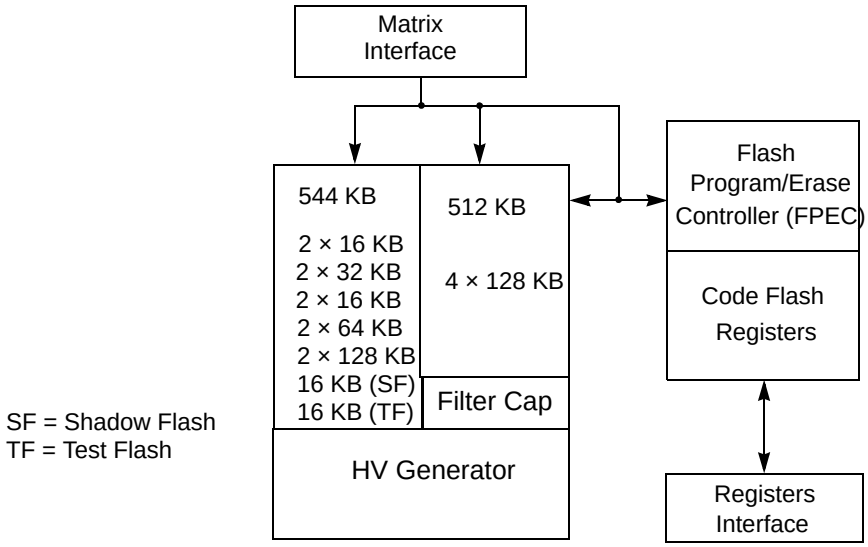


Figure 8-35. Code flash memory module structure

8.2.6.3 User mode operation

8.2.6.3.1 Data flash memory

In User mode, the data flash memory module may be read and written (register writes and interlock writes), programmed, or erased.

The default state of the data flash memory module is read. The main and test address space can be read only in the read state. The data flash memory registers are always available for read, also when the module is in disable mode (except few documented registers). The data flash memory module enters the read state on reset.

The module is in the read state under two sets of conditions:

- The read state is active when the module is enabled (User mode read)
- The read state is active when MCR[ERS] and MCR[ESUS] are high and MCR[PGM] is low (erase suspend).

Data flash memory core reads return 32 bits. Register reads return 32 bits (one word).

Data flash memory core reads are done through the BIU. Registers reads to unmapped register address space return all 0s. Registers writes to unmapped register address space have no effect.

Array reads attempted to invalid locations result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non-2^n array sizes.

Interlock writes attempted to invalid locations result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase occurs on selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous read cycles on the data flash memory module and read/write cycles on the registers are possible. However, register Read/Write accesses simultaneous to a data flash memory module interlock write are forbidden.

Read-While-Modify is not available, because the data flash memory module has only one block.

8.2.6.4 Data flash memory module power-on

The data flash memory module power-on sequence is required to wait until the code flash memory modules have completed the analog part of its own power-on sequence. This is achieved through internal synchronization signals between the two code flash memory and the data flash memory modules.

8.2.6.5 Reset

A reset is the highest priority operation for the flash memory module and terminates all other operations. The code and data flash memory modules use reset to initialize register and status bits to their default reset values.

If a flash memory module is executing a program or erase operation (MCR[PGM] = 1 or MCR[ERS] = 1) and a reset is issued, the operation is suddenly terminated and the module disables the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the flash memory module into User mode ready to receive accesses.

After reset is negated, read register access may be done, although it should be noted that registers that require updating from test block, or other inputs, may not read updated values until MCR[DONE] transitions. MCR[DONE] may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until MCR[DONE] is high.

8.2.6.6 Disable mode (Power-down)

The Disable (or Power-down) mode allows turning off all flash memory DC current sources, so that all power dissipation is due only to leakage in this mode. In Disable mode, no reads from or write to the module are possible.

The user may not read some registers (UMISR0–1, UT1, and part of UT0) until the Disable mode is exited. Write access is locked on all the registers in Disable mode.

When enabled, the flash memory module returns to its pre-disable state in all cases unless in the process of executing an erase high voltage operation at the time of disable. If a flash memory module is disabled during an erase operation, the MCR[ESUS] bit is set to 1. This means that the flash memory module is first

put into suspend state (after t_{SUSP}). The user may resume the erase operation at the time the module is enabled by clearing the MCR[ESUS] bit. MCR[EHV] must be high to resume the erase operation.

If the flash memory module is disabled during a program operation, the operation is completed in any case and the Disable mode entered only after the programming ends.

8.2.6.7 Data flash memory module slave status

Since the data flash memory module always functions as a slave to the code flash memory module, the code flash memory module cannot be put into Disable mode or Sleep mode, nor placed under reset when the data flash memory module is active.

8.2.6.8 Sleep mode

The Sleep mode turns off most of the DC current sources within the flash memory module. Wake-up time from Sleep mode is faster than wake-up time from Disable mode. In Sleep mode, reads from and writes to the module are not possible.

The user may not read some registers (UMISR0–4, UT1–2, and part of UT0) until Sleep mode is exited. Write access is locked on all the registers in Sleep mode.

When exiting from Sleep mode, the flash memory module returns to its pre-sleep state in all cases unless in the process of executing an erase high voltage operation at the time of sleep entering.

If the flash memory module is put in Sleep mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation at the time the module is exited from Sleep mode by clearing MCR[ESUS] bit. MCR[EHV] must be high to resume the erase operation.

If the flash memory module is put in Sleep mode during a program operation, the program operation is completed and the Sleep mode entered only after the programming is completed. It is forbidden to enter Disable mode when the Sleep mode is active.

8.2.7 Interaction in LSM and DPM

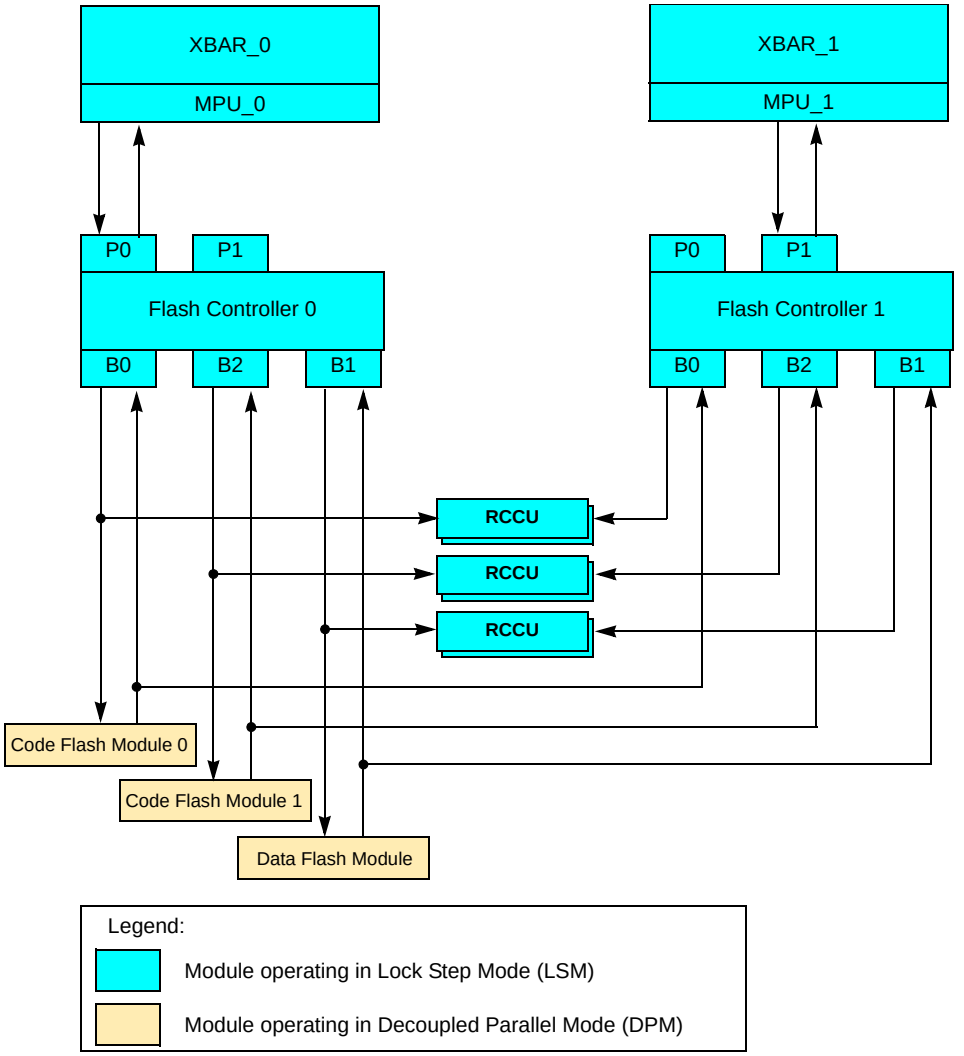


Figure 8-36. Flash memory subsystem integration in LSM

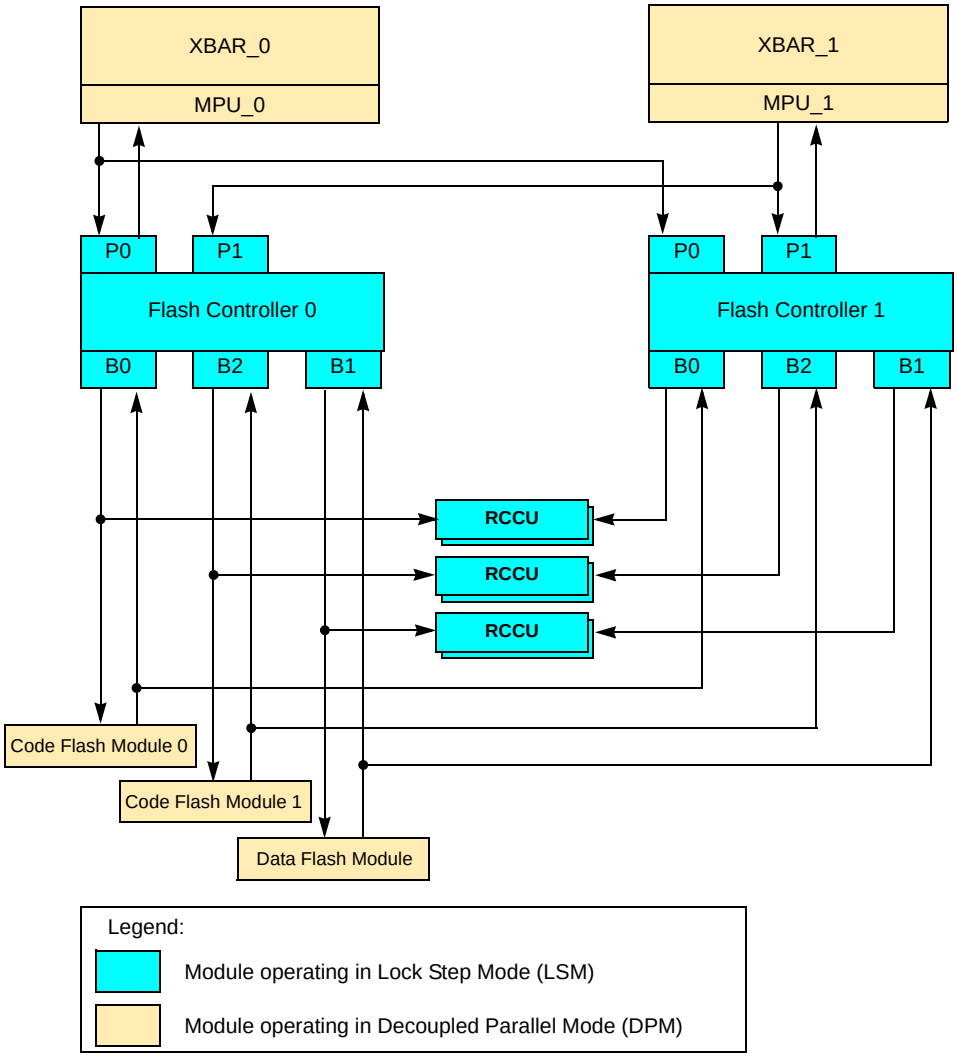


Figure 8-37. Flash memory subsystem integration in DPM

8.3 Programming considerations

8.3.1 Modify operations

All the modify operations of the flash memory module are managed through the flash memory user registers interface.

All the sectors of the flash memory module belong to the same partition (Bank). Therefore, when a modify operation is active on some sectors, no read access is possible on any other sector (Read-While-Modify is not supported).

During a flash memory modify operation, any attempt to read a flash memory location will output invalid data and set the MCR[RWE] bit. This means that the flash memory module is not fetchable when a modify operation is active. The modify operation commands must be executed from another memory (internal RAM or external memory).

If a reset occurs during a modify operation, the operation is suddenly terminated and the module is reset to Read mode. The data integrity of the flash memory section where the modify operation has been terminated or aborted is not guaranteed, and the interrupted flash memory modify operation must be repeated.

NOTE

Software executing from flash memory must not write to registers that control flash memory behavior, for example, wait state settings or prefetch enable/disable. Doing so can cause data corruption. On the MPC5675K, these registers include PFCR0, PFCR1, and PFAPR.

Further, flash memory configuration registers should be written only with 32-bit write operations to avoid any issues associated with register "incoherency" caused by bit fields spanning smaller size (8- or 16-bit) boundaries.

In general, each modify operation is started through a sequence of three steps:

1. The first instruction selects the desired operation by setting its corresponding selection bit in MCR (PGM or ERS) or UT0 (MRE or EIE).
2. The second step is the definition of the operands, the address, and the data for programming or the Sectors for erase or margin read.
3. The third instruction starts the modify operation by setting the MCR[EHV] bit or the UT0[AIE] bit.

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

Table 8-31 lists a summary of the available flash memory modify operations.

Table 8-31. Flash memory modify operations

Operation	Select bit	Operands	Start bit
Double Word Program	MCR[PGM]	Address and Data by Interlock Writes	MCR[EHV]
Sector Erase	MCR[ERS]	LMS	MCR[EHV]
Array Integrity Check	None	LMS	UT0[AIE]
Margin Read	UT0[MRE]	UT0[MRV] + LMS	UT0[AIE]
ECC logic check	UT0[EIE]	UT0[DSI], UT1, UT2	UT0[AIE]

Once the MCR[EHV] bit or the UT0[AIE] bit is set, all the operands can no more be modified until the MCR[DONE] bit or the or UT0[AID] bit is high.

In general, each modify operation is completed through a sequence of four steps:

1. Wait for operation completion, wait for MCR[DONE] or UT0[AID] to go high.
2. Check operation result, check bit MCR[PEG] (or compare UMISR0–1 with expected value).
3. Switch off FPEC by resetting MCR[EHV] or UT0[AIE].
4. Deselect current operation by clearing MCR[PGM/ERS] or UT0[MRE/EIE].

In the following sections, all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

8.3.1.1 Word program

A flash memory program sequence operates on any word within the flash memory core.

Whenever flash memory bits are programmed, ECC bits also get programmed, unless the selected address belongs to a sector in which the ECC has been disabled in order to allow bit manipulation. ECC is handled on a 32-bit boundary.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked or disabled blocks cannot be programmed.

The user may program the values in any words with a single program sequence.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
   a) Write the first address to be programmed with the program data.
   b) The flash memory module latches address bits (22:2) at this time.
   c) The flash memory module latches data written as well.
   d) This write is referred to as a program data interlock write. An interlock is at 32 bits.
3. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 8 to terminate. MCR[DONE] goes low.
4. Wait until the MCR[DONE] bit goes high.
5. Confirm MCR[PEG] = 1.
6. Write a logic 0 to the MCR[EHV] bit.
7. If more addresses are to be programmed, return to step 2.
8. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

Program may be initiated with the 0 to 1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program. The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines if the test or normal array space will be programmed by causing MCR[PEAS] to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

After the interlock write, additional writes only affect the data to be programmed in the word. If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low and MCR[EHV] is high, the user may clear EHV, resulting in a program abort. A Program abort forces the module to step 7 of the program sequence. An aborted program results in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed. The data space being operated on before the abort will contain

indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

Example 8-1. Word Program of data 0x55AA_55AA at address 0x00_AAA8

```
MCR               = 0x00000010;             /* Set PGM in MCR: Select Operation */
(0x00AAA8)        = 0x55AA55AA;             /* Latch Address 32 data */
MCR               = 0x00000011;             /* Set EHV in MCR: Operation Start */
do                                          /* Loop to wait for DONE=1 */
{ tmp             = MCR;                    /* Read MCR */
} while ( !(tmp & 0x00000400) );
status            = MCR & 0x00000200;       /* Check PEG flag */
MCR               = 0x00000010;             /* Reset EHV in MCR: Operation End */
MCR               = 0x00000000;             /* Reset PGM in MCR: Deselect Operation */
```

8.3.1.2 Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1. An erase sequence operates on any combination of blocks (sectors). The test block cannot be erased. The erase sequence is fully automated within the flash memory. The user only needs to select the blocks to be erased and initiate the erase sequence. Locked/disabled blocks cannot be erased.

If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing 1’s to the appropriate register(s) in LMS.
   Note that Lock and Select are independent. If a block is selected and locked, no erase occurs.
3. Write to any address in flash memory. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start the internal erase sequence or skip to step 9 to terminate. MCR[DONE] goes low.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG]=1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.

After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to 1. Data words written during erase sequence interlock writes are ignored. The user may terminate the erase sequence by clearing ERS before setting EHV.

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high and MCR[ESUS] is low. An erase abort forces the module to step 8 of the erase sequence. An aborted erase results in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed. The block(s) being operated on before the

abort contain indeterminate data. This may be recovered by executing an erase on the affected blocks. The user may not abort an erase sequence while in erase suspend.

Example 8-2. Erase of sectors B0F1 and B0F2

```
MCR              = 0x00000004;              /* Set ERS in MCR: Select Operation */
LMS              = 0x00000006;              /* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000)       = 0xFFFFFFFF;              /* Latch a flash Address with any data */
MCR              = 0x00000005;              /* Set EHV in MCR: Operation Start */
do                                          /* Loop to wait for DONE=1 */
{ tmp            = MCR;                     /* Read MCR */
} while ( !(tmp & 0x00000400) );
status           = MCR & 0x00000200;        /* Check PEG flag */
MCR              = 0x00000004;              /* Reset EHV in MCR: Operation End */
MCR              = 0x00000000;              /* Reset ERS in MCR: Deselect Operation */
```

8.3.1.3 Erase suspend/resume

The erase sequence may be suspended to allow read access to the flash memory core. It is not possible to program or to erase during an erase suspend. During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0-to-1 transition of MCR[ESUS] causes the module to start the sequence that places it in erase suspend.

The user must wait until MCR[DONE] = 1 before the module is suspended and further actions are attempted. MCR[DONE] will go high no more than t_{ESUS} after MCR[ESUS] is set to 1.

Once suspended, the array may be read. Flash memory core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 8-3. Sector erase suspend

```
MCR              = 0x00000007;              /* Set ESUS in MCR: Erase Suspend */
do                                          /* Loop to wait for DONE=1 */
{ tmp            = MCR;                     /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend.

The erase sequence is resumed by writing a logic 0 to MCR[ESUS]. MCR[EHV] must be set to 1 before MCR[ESUS] can be cleared to resume the operation. The module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Example 8-4. Sector erase resume

```
MCR              = 0x00000005;              /* Reset ESUS in MCR: Erase Resume */
```

8.3.2 User test mode

User Test mode is a procedure to check the integrity of the flash memory module.

Three kinds of test can be performed:

- Array integrity self check
- Margin mode read
- ECC logic check

The User test mode is equivalent to a modify operation: read accesses attempted by the user during User test mode generates a Read-While-Write Error and the MCR[RWE] bit is set. User Test operations cannot be performed on the Test block.

8.3.2.1 Array integrity self check

Array integrity is checked using a pre-defined address sequence (proprietary), and this operation is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the Multiple Input Signature Register (MISR) value, stored in the UMISR0–1 registers, to determine if an incorrect read or ECC detection was noted.

The 32-bit data, the 7 ECC data bits, and the single and double ECC errors of the word are captured by the MISR through two different read accesses at the same location. The whole check is done through two complete scans of the memory address space.

1. The first pass scans only bits 31–0 of each word.
2. The second pass scans only the 7 ECC bits and the single- and double-bit ECC errors (1 + 1) of each word.

The 32 data bits and the 7 ECC data are sampled before the eventual ECC correction, while the single- and double-bit error flags are sampled after the ECC evaluation. Only data from existing and unlocked locations are captured by the MISR. The MISR can be seeded to any value by writing the UMISR0–1 registers.

Once command is started, the array integrity check is run by FPEC using the system clock (ipg_clk) and the number of wait states identified by f90_adws(4:0).

The array integrity self check consists of the following sequence of events:

1. Set the UT0[UTE] bit by writing the related password (0xF9F9_9999) in the UT0 register.
2. Select the block(s) to be checked by writing 1s to the appropriate register(s) in LMS.
   Note that Lock and Select are independent. If a block is selected and locked, no array integrity check is performed.
3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Clear (or insert seed) UMISR0–1.
5. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
6. Wait until the UT0[AID] bit goes high.
7. Compare UMISR0–1 content with the expected result.
8. Write a logic 0 to the UT0[AIE] bit.

9. If more blocks are to be checked, return to step 2.
10. Clear UT0 writing UT0[UTE] to 0.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in an array integrity check abort. UT0[AID] must be checked to know when the aborting command has completed.

Example 8-5. Array integrity check of sectors B0F1 and B0F2

```
UT0              = 0xF9F99999;              /* Set UTE in UT0: Enable User Test */
LMS              = 0x00000006;              /* Set LSL2-1 in LMS: Select Sectors */
UT0              = 0x80000002;              /* Set AIE in UT0: Operation Start */
do                                          /* Loop to wait for AID=1 */
{ tmp            = UT0;                     /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0            = UMISR0;                  /* Read UMISR0 content*/
data1            = UMISR1;                  /* Read UMISR1 content*/
UT0              = 0x00000000;              /* Reset UTE and AIE in UT0: Operation End */
```

8.3.2.2 Margin read

The margin read procedure (either Margin 0 or Margin 1) can be run on unlocked blocks in order to unbalance the sense amplifiers, with reference to standard read conditions, so that all the read accesses reduce the margin vs. ‘0’ (UT0[MRV] = ‘0’) or vs. ‘1’ (UT0[MRV] = ‘1’). Locked sectors are ignored by MISR calculation and ECC flagging. The results of the margin reads can be checked comparing checksum value in UMISR0–1. Since Margin reads are done at voltages that differ than the normal read voltage, lifetime expectancy of the flash memory macrocell is impacted by the execution of Margin reads. Doing Margin reads repetitively results in degradation of the flash memory Array, and shorter expected lifefime experienced at normal read levels. For these reasons the Margin read usage is allowed only in Factory, while it is forbidden to use it inside the User Application.

The Margin Read Setup operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the appropriate register(s) in LMS.
   Note that Lock and Select are independent. If a block is selected and locked, no Margin Read occurs.
3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Change the value in the UT0[MRE] bit from 0 to 1.
5. Select the Margin level: UT0[MRV] = 0 for 0’s margin, UT0[MRV] = 1 for 1’s margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read Setup or skip to step 6 to terminate.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR0–1 content with the expected result.
9. Write a logic 0 to the UT0[AIE] UT0[MRE] and UT0[MRV] bits.

It is recommended to leave UT0[AIS] at 1 and use the linear address sequence and take less time.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in a Margin mode abort. UT0[AID] must be checked to know when the aborting command has completed.

Example 8-6. Margin Read Setup versus 1's

```
UT0              = 0xF9F99999;              /* Set UTE in UT0: Enable User Test */
UT0              = 0x80000020;              /* Set MRE in UT0: Select Operation */
UT0              = 0x80000030;              /* Set MRV in UT0: Select Margin versus 1's */
UT0              = 0x80000032;              /* Set AIE in UT0: Operation Start */
do                                          /* Loop to wait for AID=1 */
{ tmp            = UT0;                     /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0            = UMISR0;                  /* Read UMISR0 content*/
data1            = UMISR1;                  /* Read UMISR1 content*/
UT0              = 0x00000000;              /* Reset UTE, AIE, MRE, MRV in UT0: Deselect Operation*/
```

8.3.2.3 ECC logic check

ECC logic can be checked by forcing the input of ECC logic, the 32 bits of data and the 7 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the word.

The results of the ECC logic check can be verified by reading the MISR value.

The ECC logic check operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Write in UT1[DAI31–0] Word Input value.
3. Write in UT0[DSI6–0] the Syndrome Input value.
4. Select the ECC logic check: write a logic 1 to the UT0[EIE] bit.
5. Write a logic 1 to the UT0[AIE] bit to start the ECC logic check.
6. Wait until the UT0[AID] bit goes high.
7. Compare UMISR0–1 content with the expected result.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] is low UMISR0–1, UT1 and bits MRE, MRV, EIE, AIS, and DSI6–0 of UT0 are not accessible. Reads return indeterminate data, and writes have no effect.

Example 8-7. ECC logic check

```
UT0              = 0xF9F99999;              /* Set UTE in UT0: Enable user test */
UT1              = 0x55555555;              /* Set DAI31-0 in UT1: Word input data */
UT0              = 0x80380000;              /* Set DSI6-0 in UT0: Syndrome input data */
UT0              = 0x80380008;              /* Set EIE in UT0: Select ECC logic check */
UT0              = 0x8038000A;              /* Set AIE in UT0: Operation start */
do                                          /* Loop to wait for AID=1 */
{ tmp                    = UT0;             /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0            = UMISR0;                  /* Read UMISR0 content (expected 0x55555555) */
UT0              = 0x00000000;              /* Reset UTE, AIE and EIE in UT0: Operation end */
```

8.3.3 Error Correction Code (ECC)

The flash memory modules use ECC to improve the reliability of the data stored in flash memory. This circuitry provides correction of single-bit faults and is used to achieve automotive reliability targets. Some units will experience single-bit corrections throughout the life of the product with no impact to product reliability. For the code flash memory module, word size is fixed at 64 bits. Each 64-bit doubleword is associated with 8 ECC bits that are programmed to enable a Single-bit Error Correction and a Double-bit Error Detection (SECDED).

For the data flash memory module, word size is fixed at 32 bits. Each 32-bit word is associated with 7 ECC bits that are programmed to enable SECDED.

8.3.3.1 ECC algorithms features

The flash memory module ECC algorithm supports the following features:

- All ‘0’s Error—The All ‘0’s Error Algorithm detects as a Double ECC error any word in which all 72 bits (code flash memory) or 39 bits (data flash memory) are 0s.
- All ‘1’s No Error—The All ‘1’s No Error Algorithm detects as valid any word read on a just erased sector in which all 72 bits (code flash memory) or 39 bits (data flash memory) are 1s. This option allows performing a blank check after a sector erase operation.

8.3.3.2 Bit manipulation

8-bit clears (by byte) are allowed on any erased word maintaining valid the syndrome of the word. 8-bit clears can be done on any byte of the word without a specific order. This feature is intended as a counter for EE-Emulation.

Example 8-8. Data patterns with the same ECC syndrome (equal to 0x7F)

```
0xFFFFFFFF -> 7F
0xFFFFFF00 -> 7F
0xFFFF00FF -> 7F
0xFF00FFFF -> 7F
0x00FFFFFF -> 7F
0xFFFF0000 -> 7F
0x0000FFFF -> 7F
0xFF000000 -> 7F
0x000000FF -> 7F
0x00000000 -> 7F
```

In case flagging method is required for more than 4 writes, the following sequence allows up to 7 pattern with the same ECC syndrome.

Example 8-9. Enhanced flagging

```
0xFFFFFFFF -> 7F
0xFFFFFFB1 -> 7F
0xFFFFFF00 -> 7F
```

```
0xFFACFF00 -> 7F
0xFF00FF00 -> 7F
0xCA00FF00 -> 7F
0x0000FF00 -> 7F
0x00000000 -> 7F
```

8.3.3.2.1 3-bit error detection

40.21% of the possible 3-bit errors are detected as Double-bit ECC errors. 59.79% of the possible 3-bit errors are instead detected as Single-bit ECC errors and miscorrected.

8.3.3.2.2 EEPROM emulation (code flash memory)

The chosen ECC algorithm allows some bit manipulations so that a double word can be rewritten several times without needing an erase of the sector. This allows to use a double word to store flags useful for the Emulation.

As an example, the chosen ECC algorithm allows starting from an All ‘1’s double word value and rewriting whichever of its four 16-bit half-words to an All ‘0’s content by keeping the same ECC value.

Table 8-32 shows a set of double words sharing the same ECC value:

Table 8-32. Bit manipulation: double words with the same ECC value

Double word	ECC All ‘1’s error	ECC All ‘1’s no error
0xFFFF_FFFF_FFFF_FFFF	0x3F	0xFF
0xFFFF_FFFF_FFFF_0000	0x3F	0xFF
0xFFFF_FFFF_0000_FFFF	0x3F	0xFF
0xFFFF_0000_FFFF_FFFF	0x3F	0xFF
0x0000_FFFF_FFFF_FFFF	0x3F	0xFF
0xFFFF_FFFF_0000_0000	0x3F	0xFF
0xFFFF_0000_FFFF_0000	0x3F	0xFF
0x0000_FFFF_FFFF_0000	0x3F	0xFF
0xFFFF_0000_0000_FFFF	0x3F	0xFF
0x0000_FFFF_0000_FFFF	0x3F	0xFF
0x0000_0000_FFFF_FFFF	0x3F	0xFF
0xFFFF_0000_0000_0000	0x3F	0xFF
0x0000_FFFF_0000_0000	0x3F	0xFF
0x0000_0000_0000_0000	0x3F	0xFF

When some flash memory sectors are used to perform an Emulation, it is recommended for safety reasons to reserve at least 3 sectors to this purpose.

8.3.4 Protection strategies

Two kind of protections are available: modify protection avoids unwanted program/erase in flash memory sectors. The Censored mode avoids piracy, and must be managed by the associated code flash memory module embedded in the same device.

8.3.4.1 Modify protection

The flash memory modify protection information is stored in nonvolatile flash memory cells located in the test flash memory block. This information is read once during the flash memory initialization phase following the exit from Reset and they are stored in volatile registers that act as actuators. The reset state of all the volatile modify protection registers is the protected state.

All the nonvolatile modify protection registers can be programmed through a normal word program operation at the related locations in test flash memory. The nonvolatile modify protection registers cannot be erased.

- The nonvolatile modify protection registers are physically located in test flash memory: their bits can be programmed to ‘0’ only once and they can no more be restored to ‘1’.
- The volatile modify protection registers are read/write registers whose bits can be written at ‘0’ or ‘1’ by the user application

A software mechanism is provided to independently lock/unlock each Low, Mid Address Space Block against program and erase.

Software locking is done through the LML (Low/Mid Address Space Block Lock Register). An alternate means to enable software locking for blocks of Low Address Space only is through the SLL (Secondary Low/Mid Address Space Block Lock Register).

All these registers have a nonvolatile image stored in test flash memory (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery, the test flash memory nonvolatile image is at all 1’s, meaning that all sectors are locked.

By programming the nonvolatile locations in test flash memory, the selected sectors can be unlocked.

As the test flash memory is One Time Programmable (i.e., not erasable), once unlocked, the sectors cannot be locked again.

Of course, on the contrary, all the volatile registers can be written at 0 or 1 at any time, therefore the user application can lock and unlock sectors when desired.

8.3.4.2 Censored mode

The Censored mode information is stored in nonvolatile flash memory cells located in the shadow sector. This information is read once during the flash memory initialization phase following the exit from Reset and is stored in volatile registers that act as actuators.

The reset state of all the volatile Censored mode registers is the protected state.

All the nonvolatile Censored mode registers can be programmed through a normal double word Program operation at the related locations in the shadow sector. The nonvolatile Censored mode registers can be erased by erasing the shadow sector.

- The nonvolatile Censored mode registers are physically located in the shadow sector: their bits can be programmed to ‘0’ and eventually restored to ‘1’ by erasing the shadow sector.
- The volatile Censored mode registers are registers not accessible by the user application.

The flash memory module provides two levels of protection against piracy:

- If bits CW15–0 of NVSCI0 are programmed at 0x55AA and NVSC1 = NVSCI0, the Censored mode is disabled, while all the other possible values enable the Censored mode.
- If bit SC15–0 of NVSCI0 are programmed at 0x55AA and NVSC1 = NVSCI0, the Public Access is Disabled, while all the other possible values enable the public access.

The parts are delivered to the user with Censored mode and public access disabled.

NOTE

In a secured device, starting with a serial boot, it is possible to read the content of the four flash memory locations where the RCHW is stored.

For example, if the RCHW is stored at address 0x0000_0000, the reads at addresses 0x0000_0000, 0x0000_0004, 0x0000_0008, and 0x0000_000C return a correct value. Any other flash memory address is not readable.

The chosen flash memory ECC algorithm allows to modify the censorship status without erasing the shadow sector, as shown in Table 8-33.

Table 8-33. Bits manipulation: censorship management

Censored mode	Public access	NVSCI0	NVSCI1
Enabled	Enabled	0xFFFF_FFFF	0xFFFF_FFFF
Disabled	Enabled	0xFFFF_55AA	0xFFFF_55AA
Enabled	Disabled	0x55AA_FFFF	0x55AA_FFFF
Disabled	Disabled	0x55AA_55AA	0x55AA_55AA
Enabled	Disabled	0x55AA_0000	0x55AA_0000
Disabled	Enabled	0x0000_55AA	0x0000_55AA
Enabled	Enabled	0x0000_0000	0x0000_0000

Chapter 9
Multi-Layer AHB Crossbar Switch (XBAR)

9.1 Introduction

The MPC5675K has three instantiations of the XBAR, referred to as XBAR_0, XBAR_1, and XBAR_2. The XBAR is a duplicated periphery to meet functional safety integrity requirements (for example, SIL3 and ASIL D).

The main features of XBAR_0 and XBAR_1 are:

- Four master ports each for XBAR_0 and XBAR_1 in Lock Step Mode (LSM)
- Five master ports each for XBAR_0 and XBAR_1 in Decoupled Parallel Mode (DPM)
- Four slave ports each for XBAR_0 and XBAR_1
- Support for four concurrent transfers
- Read-only slave port access protection
- Programmable parking control
- Programmable priorities for different masters
- Configuration support for LSM and DPM

The main features of XBAR_2 are:

- Three master ports
- Three slave ports
- Support for three concurrent transfers
- 64-bit wide data path
- Support for round-robin and fixed priority arbitration scheme
- Supported for LSM and DPM

9.1.1 Logical master IDs

Table 9-1 defines the logical master IDs used for the MPC5675K XBAR masters in LSM and DPM. The logical master IDs for the two cores are different in DPM so that they both can access either XBAR_0 or XBAR_1.

Table 9-1. Logical master IDs

Master	Logical master ID	
	LSM	DPM
e200z7_0 core complex Instruction port e200z7_0 core complex Load/Store port	0	0
e200z7_1 core complex Instruction port e200z7_1 core complex Load/Store port		1

Table 9-1. Logical master IDs (continued)

Master	Logical master ID	
	LSM	DPM
DMA_0	2	2
DMA_1		6
FlexRay	3	
Ethernet (FEC)	4	
PDI	5	
e200z7_0 core Nexus	8[1]	8[1]
e200z7_1 core Nexus		9[1]

1 HMASTER[3] is not used at the MPU. Therefore the MPU cannot differentiate between core and Nexus accesses to slave ports.

9.1.2 Master port allocation

Table 9-2 defines the XBAR_0 and XBAR_1 master port allocation for the MPC5675K.

Table 9-2. XBAR_0 and XBAR_1 master port allocation

XBAR master port	LSM		DPM	
	XBAR_0 module	XBAR_1 module	XBAR_0 module	XBAR_1 module
M0[1]	Core_0 complex Instruction port	Core_1 complex Instruction port	Core_0 complex Instruction port	Core_1 complex Instruction port
M1[1]	Core_0 complex Load/Store port + Nexus port	Core_1 complex Load/Store port + Nexus port	Core_0 complex Load/Store port + Nexus port	Core_1 complex Load/Store port + Nexus port
M2[1]	From XBAR_2	From XBAR_2	From XBAR_2	From XBAR_2
M3[1]	FlexRay	FlexRay	FlexRay	FlexRay
M4	not implemented on device			
M5	not implemented on device			
M6[2]	not implemented in LSM		Core_1 complex Load/Store port + Nexus port	Core_0 complex Load/Store port + Nexus port
M7	not implemented on device			

1 Unchanged for LSM and DPM.

2 This master is dependent on DPM.

Table 9-3 defines the XBAR_2 master port allocation for the MPC5675K. These masters are mode-independent in LSM and DPM.

Table 9-3. XBAR_2 master port allocation

XBAR_2 master port	LSM and DPM
	XBAR_2 module
M0	DMA_0
M1	DMA_1
M2	PDI
M3	Ethernet (FEC)

9.1.3 Slave port allocation

Table 9-4 defines the XBAR slave port allocation for the MPC5675K.

Table 9-4. XBAR_0 and XBAR_1 slave port allocation

XBAR slave port	LSM		DPM	
	XBAR_0 module	XBAR_1 module	XBAR_0 module	XBAR_1 module
S0	PFLASH_0_PORT_0	PFLASH_1_PORT_1[1]	PFLASH_0_PORT_0 PFLASH_1_PORT_0	PFLASH_0_PORT_1 PFLASH_1_PORT_1[2]
S1	not implemented on device			
S2	SRAMC_0	SRAMC_1	SRAMC_0	SRAMC_1
S3	EBI/DRAMC	EBI/DRAMC	EBI/DRAMC	EBI/DRAMC
S4	not implemented on device			
S5	not implemented on device			
S6	not implemented on device			
S7	PBRIDGE_0	PBRIDGE_1	PBRIDGE_0	PBRIDGE_1

1 This slave is dependent on LSM.
2 This slave is dependent on DPM.

Table 9-5. XBAR_2 slave port allocation

XBAR master port	LSM and DPM
	XBAR_2 module
S0	To Master Port XBAR_0
S1	To Master Port XBAR_1
S2	To DRAMC Port 0
S3–S7	not allocated

9.2 XBAR registers

This section provides information on XBAR registers.

Table 9-6 shows the base addresses for the XBAR modules. Table 9-7 shows the memory map for the XBAR program-visible registers.

Table 9-6. XBAR module base addresses

Mode	Module	Module base address
LSM	XBAR_0	0xFFF0_4000
	XBAR_1	
DPM	XBAR_0	0xFFF0_4000 (same as LSM)
	XBAR_1	0x8FF0_4000

NOTE

XBAR_2 is not memory-mapped and thus not configurable. The default settings for XBAR_2 provide fully functional connectivity for its master and slave ports.

Table 9-7. XBAR memory map

Offset from XBAR_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Master Priority Register for Slave port 0 (MPR0)	R/W	0x0400_3210	on page 233
0x0004–0x000F	Reserved			
0x0010	General Purpose Control Register for Slave port 0 (SGPCR0)	R/W	0x0000_0000	on page 234
0x0014–0x01FF	Reserved			
0x0200	Master Priority Register for Slave port 2 (MPR2)	R/W	0x0400_3210	on page 233
0x0204–0x020F	Reserved			
0x0210	General Purpose Control Register for Slave port 2 (SGPCR2)	R/W	0x0000_0000	on page 234
0x0214–0x02FF	Reserved			
0x0300	Master Priority Register for Slave port 3 (MPR3)	R/W	0x0400_3210	on page 233
0x0304–0x030F	Reserved			
0x0310	General Purpose Control Register for Slave port 3 (SGPCR3)	R/W	0x0000_0000	on page 234
0x0314–0x06FF	Reserved			
0x0700	Master Priority Register for Slave port 7 (MPR7)	R/W	0x0400_3210	on page 233
0x0704–0x070F	Reserved			

Table 9-7. XBAR memory map (continued)

Offset from XBAR_BASE	Register	Access[1]	Reset Value[2]	Location
0x0710	General Purpose Control Register for Slave port 7 (SGPCR7)	R/W	0x0000_0000	on page 234
0x0714–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where "H" is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

9.2.1 Register summary

There are two registers that reside in each slave port of the XBAR. These registers are IP bus compliant registers. Read and write transfers both require two IP bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

The registers are fully decoded and a bus error is returned if an unimplemented location is accessed within the XBAR.

The slave registers also feature the RO bit that, when written with a 1, prevents the registers from being written to again. The registers are still readable, but future write attempts have no effect on the registers and are terminated with an error response.

9.2.2 XBAR register descriptions

The following paragraphs provide detailed descriptions of the various XBAR registers.

9.2.2.1 Master Priority Register *n* (MPR*n*)

The Master Priority Register *n* (MPR*n*) sets the priority of each master port on a per slave port basis and resides in each slave port.

Address: Base + 0x0000 (MPR0), Base + 0x0100 (MPR1), Base + 0x0200 (MPR2), Base + 0x0300 (MPR3), Base + 0x0600 (MPR6)

Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	MSTR_6			0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MSTR_3			0	MSTR_2			0	MSTR_1			0	MSTR_0		
W																
Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 9-1. Master Priority Register *n* (MPR*n*)

Table 9-8. MPR*n* field descriptions

Field	Description
MSTR_6	Master 6 Priority. These bits set the arbitration priority for master port 6 on the associated slave port. These bits are initialized by hardware reset. The reset value is 0b100. 000 This master has the highest priority when accessing the slave port. . . . 111 This master has the lowest priority when accessing the slave port.
MSTR_3	Master 3 Priority. These bits set the arbitration priority for master port 3 on the associated slave port. These bits are initialized by hardware reset. The reset value is 0b011. 000 This master has the highest priority when accessing the slave port. . . . 111 This master has the lowest priority when accessing the slave port.
MSTR_2	Master 2 Priority. These bits set the arbitration priority for master port 2 on the associated slave port. These bits are initialized by hardware reset. The reset value is 0b010. 000 This master has the highest priority when accessing the slave port. . . . 111 This master has the lowest priority when accessing the slave port.
MSTR_1	Master 1 Priority. These bits set the arbitration priority for master port 1 on the associated slave port. These bits are initialized by hardware reset. The reset value is 0b001. 000 This master has the highest priority when accessing the slave port. . . . 111 This master has the lowest priority when accessing the slave port.
MSTR_0	Master 0 Priority. These bits set the arbitration priority for master port 0 on the associated slave port. These bits are initialized by hardware reset. The reset value is 0b000. 000 This master has the highest priority when accessing the slave port. . . . 111 This master has the lowest priority when accessing the slave port.

The MPR*n* can only be accessed in supervisor mode with 32-bit accesses. Once the RO (Read Only) bit has been set in the Slave General Purpose Control Register (SGPCR*n*), the MPR*n* can only be read. Attempts to write to it have no effect on the MPR*n*, and result in an error response.

NOTE

No two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level result in an error response, and the MPR*n* is not updated.

9.2.2.2 Slave General Purpose Control Register (SGPCR*n*)

The Slave General Purpose Control Register *n* (SGPCR*n*) controls several features of each slave port.

Once set, the RO bit prevents any registers associated with this slave port from being written. This bit may be written with 0 as many times as the user desires, but once it is written to a 1, only a reset condition allows it to be written again.

The Halt Low Priority (HLP) bit sets the priority of the **max_halt_request** input to the lowest possible priority for initial arbitration of the slave ports. By default it is the highest priority. Setting this bit does not affect the **max_halt_request** from attaining highest priority once it has control of the slave ports.

The PCTL bits determine how the slave port is parked when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low-power park mode that forces all the outputs of the slave port to inactive states when no master is requesting an access. The low-power park feature can result in an overall power savings if a the slave port is not saturated; however, it forces an extra clock of latency whenever any master tries to access it when it is not in use because it is not parked on any master.

The PARK bits determine which master the slave parks on when no master is making an active request and the **max_halt_request** input is negated. Please use caution to only select master ports that are actually present in the design. If the user programs the PARK bits to a master not present on the device, undefined behavior can result.

Address: Base + 0x0010 (SGPCR0) Base + 0x0210 (SGPCR2) Base + 0x0310 (SGPCR3) Base + 0x0710 (SGPCR7) Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO	HLP	0	0	0	0	0	0	0	HPE 6	0	0	HPE 3	HPE 2	HPE 1	HPE 0
W																
Reset	0[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-2. Slave General Purpose Control Register *n* (SGPCR*n*)

[1] Once the RO bit is written to a 1, only a hardware reset returns it to a 0.

Table 9-9. SGPCR*n* field descriptions

Field	Description
RO	Read Only. This bit forces all of a slave port's registers to be read-only. Once written to 1, it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0b0. 0 All this slave port's registers can be written. 1 All this slave port's registers are read-only and cannot be written. Attempted writes have no effect, and result in an error response.
HLP	Halt Low Priority. This bit sets the initial arbitration priority of the max_halt_request input. This bit is initialized by hardware reset. The reset value is 0b0. 0 The max_halt_request input has the highest priority for arbitration on this slave port. 1 The max_halt_request input has the lowest initial priority for arbitration on this slave port.
HPE*x*	High Priority Enable. These bits are used to enable the mX_high_priority inputs for the respective master. These bits are initialized by hardware reset. The reset value is 0b0. 0 The mX_high_priority input is disabled on this slave port. 1 The mX_high_priority input is enabled on this slave port.

Table 9-9. SGPCR*n* field descriptions (continued)

Field	Description
ARB	Arbitration mode. These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 0b00. 00 Fixed priority. 01 Round-robin (rotating) Priority. 10 Reserved. 11 Reserved.
PCTL	Parking control. These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 0b00. 00 When no master is making a request, the arbiter parks the slave port on the master port defined by the PARK bit field. 01 When no master is making a request, the arbiter parks the slave port on the last master to be in control of the slave port. 10 When no master is making a request, the arbiter parks the slave port on no master and drives all outputs to a constant safe state. 11 Reserved.
PARK	PARK. These bits are used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. These bits are initialized by hardware reset. The reset value is 0b000. 000 Park on Master Port 0. 001 Park on Master Port 1. 010 Park on Master Port 2. 011 Reserved. 100 Reserved. 101 Reserved. 110 Park on Master Port 6. 111 Reserved.

The SGPCR*n* can only be accessed in supervisor mode with 32-bit accesses. Once the RO bit has been set in the SGPCR*n*, the SGPCR*n* can only be read. Attempts to write to it have no effect on the SGPCR*n*, and result in an error response.

9.3 Overview

This section provides an overview of the generic multi-layer AHB crossbar switch (XBAR). The purpose of the XBAR is to concurrently support up to eight simultaneous connections between master ports and slave ports. The XBAR supports a 32-bit address bus width and a 64-bit data bus width at all master and slave ports.

9.3.1 Features

The XBAR has the ability to gain control of all the slave ports and prevent any masters from making accesses to the slave ports. This feature is useful when the user wishes to turn off the clocks to the system and needs to ensure that bus activity is not interrupted.

The XBAR can put each slave port into a low-power park mode so that slave port does not dissipate any power transitioning address, control, or data signals when not being actively accessed by a master port.

Each slave port can also support multiple master priority schemes. Each slave port has a hardware input that selects the master priority scheme so the user can dynamically change master priority levels on a slave port by slave port basis.

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

9.3.2 Limitations

The XBAR routes bus transactions initiated on the master ports to the appropriate slave ports. No provision is included to route transactions initiated on the slave ports to other slave ports or to master ports. Simply put, the slave ports do not support the bus request/bus grant protocol. The XBAR assumes it is the sole master of each slave port.

Since the XBAR does not support the bus request/bus grant protocol, if multiple masters are to be connected to a single master port, an external arbiter needs to be used. In the case of a single master connecting to a master port, the single master's bus grant signal must be tied off in the asserted state.

Undefined length burst transfers are not supported on the XBAR for the MPC5675K device.

9.3.3 General operation

When a master accesses the XBAR, the access is immediately taken by the XBAR. If the targeted slave port of the access is available, then the access is immediately presented on the slave port. It is possible to make single-clock (0 wait state) accesses through the XBAR. If the targeted slave port of the access is busy or parked on a different master port, the requesting master sees wait states inserted (**hready** held negated) until the targeted slave port can service the master's request. The latency in servicing the request depends on each master's priority level and the responding peripheral's access time.

Since the XBAR appears to be just another slave to the master device, the master device has no knowledge of whether or not it actually owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

Once the master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master could also lose control of the slave port if another higher priority master makes a request to the slave port; however, if the master is running a locked or fixed length burst transfer, it retains control of the slave port until that transfer is completed.

The XBAR terminates all master IDLE transfers (as opposed to allowing the termination to come from one of the slave buses). Additionally, when no master is requesting access to a slave port, the XBAR drives IDLE transfers onto the slave bus, even though a default master may be granted access to the slave port. When the XBAR is controlling the slave bus (that is, during low-power park or halt mode), the **hmaster** field indicates 4'b0000.

When a slave bus is being IDLEd by the XBAR, it can park the slave port on the master port indicated by the PARK bits in the SGPCR (Slave General Purpose Control Register). This can be done in an attempt to save the initial clock of arbitration delay that would otherwise be seen if the master had to arbitrate to gain control of the slave port. The slave port can also be put into low-power park mode to save power.

9.3.4 Coherency

Since the content of the registers has a real time effect on the operation of the XBAR, it is important for the user to understand that any register modifications take effect as soon as the register is written. The values of the registers do not track with slave port-related AHB accesses but instead track only with IP bus accesses.

9.3.5 XBAR_2 configuration

This section describes in more detail the functionality of the XBAR_2, which has limited functionality compared to XBAR_0 and XBAR_1.

The XBAR_2 arbitration scheme is selectable between round-robin and fixed priority. The configuration is controlled by the MUDCR[MUDCR0] bit in the ECSM (see Section 23.3.2.8, Miscellaneous User-Defined Control Register (MUDCR)). In LSM, both ECSMs provide this information, which is then compared by an RCCU pair. The output of ECSM_0 controls the XBAR_2 configuration. In DPM, only ECSM_0 configures the arbitration scheme.

The following configuration is used for the master parking:

- Slave Port 0 is parked on DMA_0
- Slave Port 1 is parked on DMA_1
- Slave Port 2 is parked on PDI at reset, and during normal operation parks on the last master to access it

In fixed priority mode, the FEC has the highest priority, followed by the PDI, then DMA_1, and lastly DMA_0.

9.3.6 Port splitter

A port splitter is used in front of the EBI and the DRAMC port 1 that splits accesses coming from XBAR_0 to the EBI or to the DRAMC port 1, based on the address.

9.3.7 Arbitration

The XBAR supports two arbitration schemes: a simple fixed-priority comparison algorithm, and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

9.3.7.1 Fixed-priority operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the MPR*n* (Master Priority Register *n*). If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the new requesting master is granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case, the new requesting master must wait until the end of the burst transfer or locked transfer before it is granted control of the slave port.

If the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

9.3.7.2 Round-robin priority operation

When operating in round-robin mode, each master is assigned a relative priority based on the master number. This relative priority is compared to the ID of the last master to perform a transfer on the slave bus. The highest priority requesting master becomes owner of the slave bus at the next transfer boundary (accounting for locked and fixed-length burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number, not the **hmaster** field).

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port at the next assertion of **sX_hready**, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the XBAR is implemented with master ports 0, 1, 4, and 5. If the last master of the slave port was master 1, and master 0, 4 and 5 make simultaneous requests, they are serviced in the order 4, 5, and then 0.

Parking may still be used in a round-robin mode, but does not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff occurs to the next master in line after one cycle of arbitration. If the slave port is put into low-power park mode, the round-robin pointer is reset to point at master port 0, giving it the highest priority.

Each master port has an **mX_high_priority** input that can be enabled by writing the correct data to the SGPCR. If a master's **mX_high_priority** input is enabled for a slave port programmed for round-robin mode, that master can force the slave port into fixed priority mode by asserting its **mX_high_priority** input while making a request to that particular slave port. While that (or any enabled) master's **mX_high_priority** input is asserted while making an access attempt to that particular slave port, the slave port remains in fixed priority mode. Once that (or any enabled) master's **mX_high_priority** input is negated, or the master no longer attempts to make accesses to that particular slave port, the slave port reverts back to round-robin priority mode and the pointer is e set on the last master to access the slave port.

9.3.8 Priority assignment

Each master port needs to be assigned a unique 3-bit priority level. If an attempt is made to program multiple master ports with the same priority level within a register (using the MPR*n*) the XBAR responds with an error and the registers are not updated.

9.3.8.1 Priority elevation

The XBAR has a hardware input per master port (**mX_high_priority**) that temporarily elevates the master's priority level on all slave ports. When the master's **mX_high_priority** input is asserted, the master port automatically has higher priority than all other master ports that do not have their **mX_high_priority** input asserted, regardless of the priority levels programmed in the MPR*n*. If multiple master ports have their **mX_high_priority** input asserted, they all have higher priority than all master ports that do not have their **mX_high_priority** inputs asserted. The MPR*n* priority level determines which master port that has its **mX_high_priority** input asserted has the highest priority on a slave port by slave port basis.

This functionality is useful because it allows the user to automatically elevate a master port's priority level throughout the XBAR in order to quickly perform temporary tasks such as servicing interrupts.

The HPE*x* bits must be set in the SGPCR in the slave port in order for the **mX_high_priority** inputs to be received by the slave port.

9.3.9 Master port functionality

9.3.9.1 General

Each master port consists of:

- Two decoders
- A capture unit
- A register slice
- A mux
- A small state machine

The first decoder is used to decode the **mX_hsel_slv** and control signals coming directly from the master, telling the state machine where the master's next access will be and if it is in fact a legal access. The second decoder receives its input from the capture unit, so it may be looking directly at the signals coming from

the master or it may be looking at captured signals coming from the master, depending entirely on the state of the targeted slave port. The second decoder is then used to generate the access requests that go to the slave ports.

The capture unit is used to capture the address and control information coming from the master in the event that the targeted slave port cannot immediately service the master. The capture unit is controlled by outputs from the state machine, which tells it to either pass through the original master signals or the captured signals.

The register slice contains the registers associated with the specific master port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The mux selects which slave's read data is sent back to the master. The mux is controlled by the state machine.

The state machine controls all aspects of the master port. It knows which slave port the master wants to make a request to and controls when that request is made. It also has knowledge of each slave port, knowing whether or not the slave port is ready to accept an access from the master port. This determines whether the master may immediately have its request taken by the slave port or whether the master port must capture the master's request and queue it at the slave port boundary.

9.3.9.2 Master port decoders

The decoders are very simple as they ensure an access request is allowed to be made and that the slave port targeted is actually present in the design. The decoders feeding the state machine are always enabled. The decoders that select the slave are enabled only when the master port controlling state machine wants to make a request to a slave port. This is necessary so that if a master port is making an access to a slave port and is being wait stated, and its next access is to a different slave port, the request to the second slave port can be held off until the access to the first slave port is terminated.

The decoders also output a "hole decode" or illegal access signal that tells the state machine if the master is trying to access a slave port that does not exist.

9.3.9.3 Master port capture unit

The capture unit captures the state of the master's address and control signals if the XBAR cannot immediately pass the master's request through to the proper slave port. The capture unit consists of a set of flops and a mux that selects either the asynchronous path from address and control or the flopped (captured) address and control information.

9.3.9.4 Master port registers

The registers in the master port are only those registers associated with this particular master port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

A register control block at the same level of the master port and slave port instantiations in the XBAR ensures that all accesses are 32-bit supervisor accesses before passing them on to the master ports.

The register outputs are connected directly to the state machine.

9.3.9.5 Master port state machine

9.3.9.5.1 Master port state machine states

The master side state machine's main function is to monitor the activities of the master port. The state machine has six states: **busy**, **idle**, **waiting**, **stalled**, **steady state**, **first cycle error response**, and **second cycle error response**.

The **busy** state is used when the master runs a BUSY cycle to the master port. The master port maintains its request to the slave port if it currently owns the slave port; however, if it loses control of the slave port it no longer maintains its request. If the master port loses control of the slave port, it is not allowed to make another request to the slave port until it runs a NSEQ or SEQ cycle.

The **idle** state is used when the master runs a valid IDLE cycle to the master port. The master port makes no requests to the slave ports (disables the slave port decoder) and terminates the IDLE cycle.

The **waiting** state is used when the **hsel** signal is negated to the master port, indicating the master is running valid cycles to a local slave other than the XBAR. In this case the max disables the slave port decoder and holds **hresp** and **hready** negated.

The **stalled** state is used when the master makes a request to a slave port that is not immediately ready to receive the request. In this case, the state machine directs the capture unit to send out the captured address and control signals, and enables the slave port decoder to indicate a pending request to the appropriate slave port.

The **steady** state is used when the master port and slave port are in fully asynchronous mode, making the XBAR completely transparent in the access. The state machine selects the appropriate slave's **hresp, hready**, and **hrdata** to pass back to the master.

The **first cycle error response** and **second cycle error response** states are self-explanatory. The XBAR responds with an error response to the master if the master tries to access an unimplemented memory location through the XBAR (that is, a slave port that does not exist).

9.3.9.5.2 Master port state machine slave swapping

The design of the master side state machine is fairly straightforward. The one real decision to be made is how to handle the master moving from one slave port access to another slave port access. The approach that was taken is to minimize or eliminate when possible any "bubbles" that would be inserted into the access due to switching slave ports.

The state machine does not allow the master to request access to another slave port until the current access being made is terminated. This prevents a single master from owning two slave ports at the same time (the slave port it is currently accessing and the slave port it wishes to access next).

The state machine also maintains watch on the slave port the master is accessing as well as the slave port the master wishes to switch to. If the new slave port is parked on the master, then the master can make the switch without incurring any delays. The termination of the current access also acts as the launch of the

new access on the new slave port. If the new slave port is not parked on the master, then the master incurs a minimum one clock delay before it can launch its access on the new slave port.

This is the same for switching from the **busy, idle**, or **waiting** state to actively accessing a slave port. If the slave port is parked on the master, the state machine goes to the **steady** state and the access begins immediately. If the slave port is not parked on the master (serving another master, parked on another master or in low-power park mode), then the state machine transitions to the **stalled** state and at least a one clock penalty is incurred.

9.3.10 Slave port functionality

9.3.10.1 General

Each slave port consists of a register slice, a bank of muxes, and a state machine.

The register slice contains the registers associated with the specific slave port. The registers have a quasi-IP bus interface at this level for reads and writes and the outputs feed directly into the state machine.

The muxes are a series of 8-to-1 muxes that take in all the address, control and write data information from each of the master ports and then pass the correct master's signals to the slave port. The state machine controls all the muxes.

The state machine is where the main slave port arbitration occurs. It decides which master is in control of the slave port and which master will be in control of the slave port in the next bus cycle.

9.3.10.2 Slave port muxes

The XBAR slave port instantiates many 8-to-1 muxes, one for each master-to-slave signal. All the muxes are designed in an AND - OR fashion, so that if no master is selected, the output of the muxes is zero. (This is an important feature for low-power park mode.)

The muxes also have an override signal that is used by the slave port to asynchronously force IDLE cycles onto the slave bus. When the state machine forces an IDLE cycle, it zeros out **htrans** and **hmastlock**, making sure the slave bus sees a valid IDLE cycle being run by the XBAR.

The enable to the mux controlling **htrans** also contains an additional control signal from the state machine so that a NSEQ transaction can be forced. This is done any time the slave port switches masters to ensure that no IDLE-SEQ, BUSY-SEQ or NSEQ-SEQ transactions are seen on the slave port when they shouldn't be. If the state machine indicates to run both an IDLE and an NSEQ cycle, the IDLE directive has priority.

NOTE

IDLE-SEQ is in fact an illegal access, but a possible scenario given the multi-master environment in the XBAR unless corrected by the XBAR.

9.3.10.3 Slave port registers

A register control block at the same level of the master port and slave port instantiations in the XBAR ensures that all accesses are 32-bit supervisor accesses before passing them on to the master and slave ports.

The registers in the slave port are only those registers associated with this particular slave port. The read and write interface for the registers is a quasi-IP bus interface. It is not a full IP bus interface at this level because not all the IP bus signals are routed this deep in the design.

The register outputs are connected directly to the slave state machine with the **sX_ampr_sel** input determining which priority register values, halt priority value, arbitration algorithm, and parking control bits are passed to the state machine. The registers can be read an unlimited number of times. The registers can only be written to as long as the RO bit is written to 0 in the SGPCR. Once it is written to a 1, only a hardware reset allows the registers to be written again.

9.3.10.4 Slave port state machine

9.3.10.4.1 Slave port state machine states

At the heart of the slave port is the state machine. The state machine is simplicity itself, requiring only four states - **steady state**, **transition state, transition hold state** and **hold state**. Either the slave port is owned by the same master it was in the last clock cycle (either by active use or by parking), it is transitioning to a new master (either for active use or parking), it is transitioning to a new master during wait states or it is being held on the same master pending a transition to a new master.

9.3.10.4.2 Slave port state machine arbitration

The real work of the state machine is to determine which master port will control the slave port in the next clock cycle, the arbitration. Each master is programmed with a fixed 3-bit priority level. A fourth priority bit is derived from the **mX_high_priority** inputs on the master ports, effectively making each master's priority level a 4-bit field with **mX_high_priority** being the MSB. The XBAR uses these bits in determining priority levels when programmed for fixed priority mode of operation or when one of the enabled **mX_high_priority** inputs is asserted.

Arbitration always occurs on a clock edge, but only occurs on edges when a change in mastership will not violate AHB-Lite protocols. Valid arbitrations points include any clock cycle in which **sX_hready** is asserted (provide the master is not performing a burst or locked cycle) and any wait state in which the master owning the bus indicates a transfer type of IDLE (provided the master is not performing a locked cycle).

Since arbitration can occur on every clock cycle, the slave port masks off all master requests if the current master is performing a locked transfer or a protected burst transfer, to support the finalization of a locked or protected portion of a burst sequence regardless of its priority level

9.3.10.4.3 Slave port state machine master handoff

The only times the slave port switches masters when programmed for fixed priority mode of operation is when a higher priority master makes a request or when the current master is the highest priority and it gives up the slave port by either running and IDLE cycle to the slave port or running a valid access to a location other than the slave port.

If the current master loses control of the slave port because a higher priority master takes it away, the slave port does not incur any wasted cycles. The current master has its current cycle terminated by the slave port

at the same time the new master's address and control information are recognized by the slave port. This appears as a seamless transition on the slave port.

If the current master is being wait-stated when the higher priority master makes its request, then the current master is allowed to make one more transaction on the slave bus before giving it up to the new master. Figure 9-3 illustrates the effect of a higher priority master taking control of the bus when the slave port is programmed for a fixed priority mode of operation.

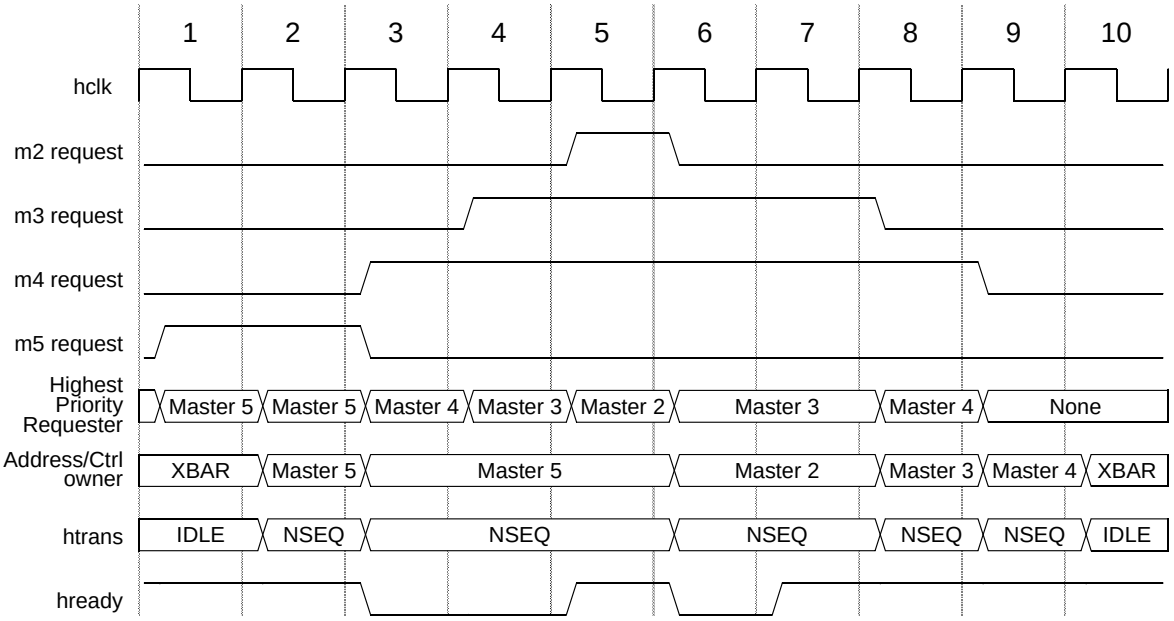


Figure 9-3. Low to high priority mastership change

If the current master is the highest priority master and it gives up the slave port by running an IDLE cycle or by running a valid cycle to another location other than the slave port, the next highest priority master gains control of the slave port. If the current access incurs any wait states then the transition is seamless and no bandwidth is lost; however, if the current transaction is terminated without wait states, one IDLE cycle is forced onto the slave bus by the XBAR before the new master takes control of the slave port. If no other master is requesting the bus. then IDLE cycles are run by the XBAR but no bandwidth actually is lost, since no master is making a request. Figure 9-4 illustrates the effect of a higher priority master giving up control of the bus.

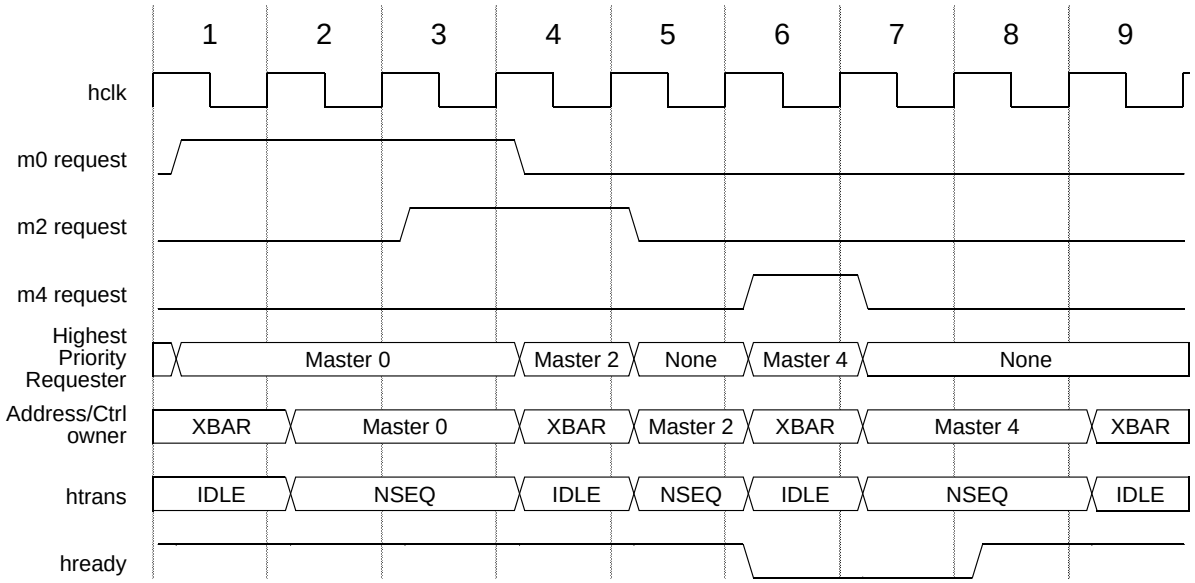


Figure 9-4. High to low priority mastership change

When the slave port is programmed for round-robin mode of arbitration, then the slave port switches masters any time more than one master actively makes a request to the slave port. This happens because any master other than the one which presently owns the bus is considered to have higher priority. Figure 9-5 shows an example of round-robin mode of operation.

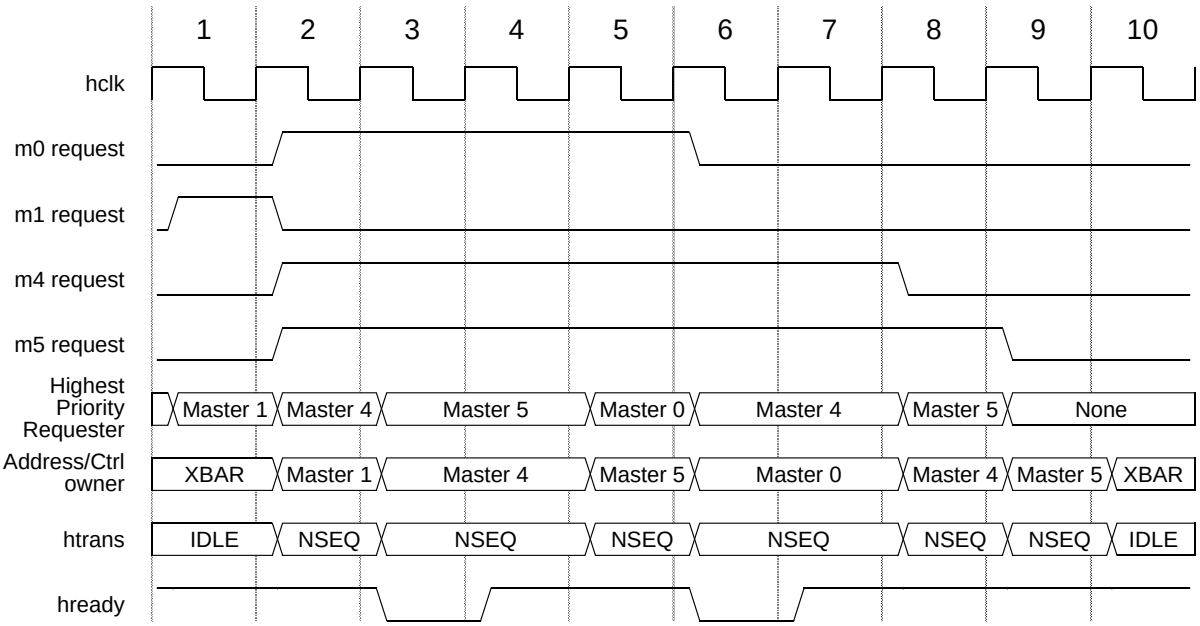


Figure 9-5. Round-robin mastership change

9.3.10.4.4 Slave port state machine parking

If no master is currently making a request to the slave port, then the slave port is parked. in one of four places, dictated by the PCTL and PARK bits in the GPCR or AGPCR (depending on the state of the **sX_ampr_sel**) and the locked state of the last master to access it.

If the last master to access the slave port ran a locked cycle and continues to run locked cycles even after leaving the slave port, the slave port parks on that master without regard to the bit settings in the GPCR and without regard to pending requests from other masters. This is done so that a master can run a locked transfer to the slave port, leave it, and return to it and delaying the access of other masters to it (provided the master maintains all transfers are locked transfers). If locking is not an issue for parking, the GPCR bits dictate the parking method.

If the PCTL bits are set for low-power park mode, then the slave port enters low-power park mode. In this mode, it does not recognize any master as being in control of it and it does not select any master's signals to pass through to the slave bus. In this case, all slave bus activity halts, because all slave bus signals being driven from the XBAR are 0. This of course can save quite a bit of power if the slave port will not be in use for some time. The down side is that when a master does make a request to the slave port, it will be delayed by one clock since it must arbitrate to acquire ownership of the slave port.

If the PCTL bits are set to "park on last" mode, then the slave port parks on the last master to access it, passing all that master's signals through to the slave bus. The XBAR asynchronously forces **htrans[1:0]**, **hmaster[3:0]**, **hburst[2:0]** and **hmastlock** to 0 for all access that the master does not run to the slave port. When that master access the slave port again, it does not pay any arbitration penalty; however, if any other master wishes to access the slave port, a one clock arbitration penalty is imposed.

If the PCTL bits are set to "use PARK/APARK" mode, then the slave port parks on the master designated by the PARK bits. The behavior here is the same as for the "park on last" mode, with the exception that a specific master will be parked on instead of the last master to access the slave port. If the master designated by the PARK bits tries to access the slave port, it does not pay an arbitration penalty, while any other master pays a one clock penalty. Figure 9-6 illustrates parking on a specific master.

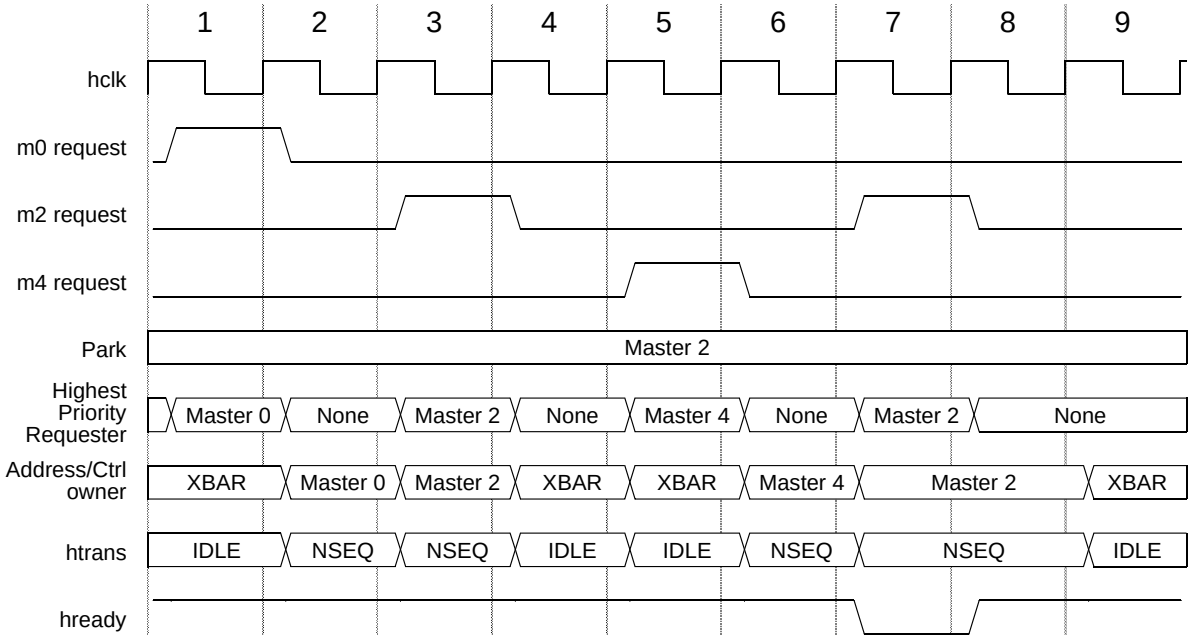


Figure 9-6. Parking on a specific master

Figure 9-7 illustrates parking on the last master. Note that in cycle 6 simultaneous requests are made by master 2 and master 4. Although master 2 has higher priority, the slave bus is parked on master 4 so master 4's access is taken first. The slave port parks on master 2 once it has given control to master 2. This same situation can occur when parking on a specific master as well.

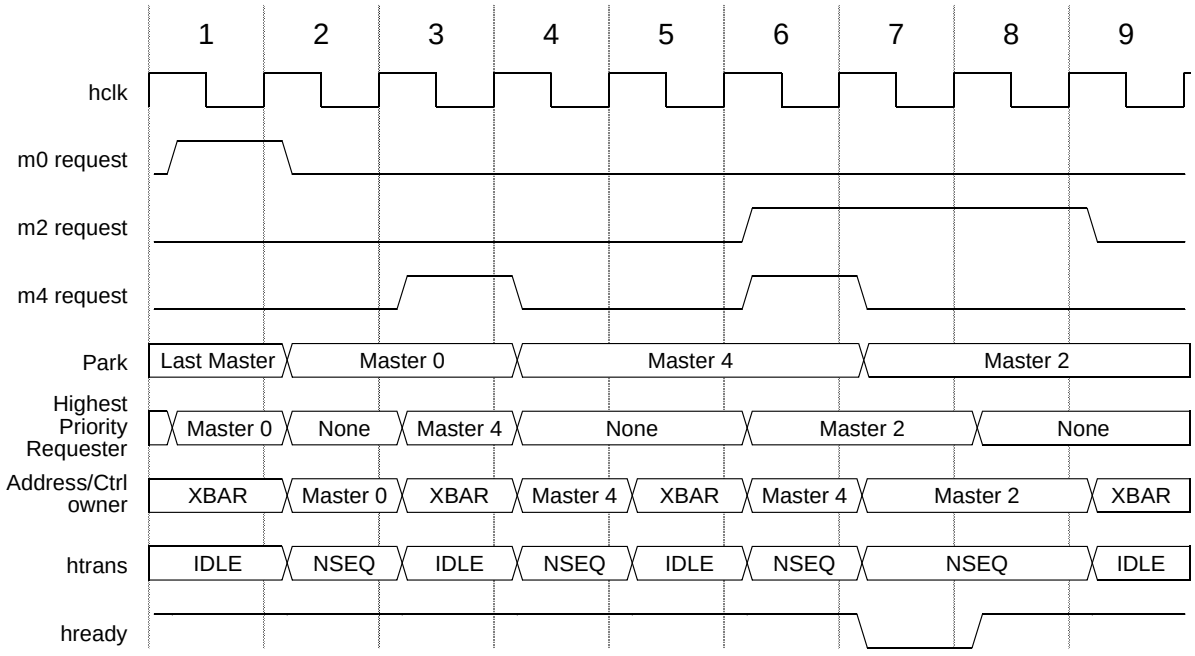


Figure 9-7. Parking on last master

9.3.10.4.5 Slave port state machine halt mode

If the **max_halt_request** input is asserted, the slave port eventually halts all slave bus activity and goes into halt mode, which is almost identical to low-power park mode. The GPCR[HLP] bit controls the priority level of the **max_halt_request** in the arbitration algorithm. If the GPCR[HLP] bit is cleared, then the **max_halt_request** has the highest priority of any master and gains control of the slave port at the next arbitration point (most likely the next bus cycle, unless the current master is running a locked or fixed-length burst transfer). If the GPCR[HLP] bit is set, then the slave port must wait until no masters are actively making requests before moving to halt mode.

Regardless of the state of the GPCR[HLP] bit, once the slave port has gone into halt mode as a result of **max_halt_request** being asserted, it remains in halt mode until **max_halt_request** is negated, regardless of the priority level of any masters that may make requests.

In halt mode, no master is selected to own the slave port, so all the outputs of the slave port are set to 0.

9.4 Initialization/application information

No initialization is required by or for the XBAR. Hardware reset ensures all the register bits used by the XBAR are properly initialized.

9.5 Interface

This section provides information on the XBAR interface.

9.5.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate in parallel with multiple slaves. In order to maximize data throughput it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls the masters or inserts bubbles on the slave side.

9.5.2 Master ports

Master accesses receive one of four responses from the XBAR. They can be ignored, terminated, taken, stalled or responded to with an error.

9.5.2.1 Ignored accesses

A master access is ignored if the **hsel** input of the XBAR is not asserted. The XBAR responds to IDLE transfers when the **hsel** input is asserted but do not allow the access to pass through the XBAR.

9.5.2.2 Terminated accesses

A master access is terminated if the **hsel** input of the XBAR is asserted and the transfer type is IDLE. The XBAR terminates the access and it is not allowed to pass through the XBAR.

9.5.2.3 Taken accesses

A master access is taken if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely transparent and the master's access is immediately seen on the slave bus, and no arbitration delays are incurred.

9.5.2.4 Stalled accesses

A master access is stalled if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a slave port that is busy serving another master, parked on another master, or is in low-power park mode. The XBAR indicates to the master that the address phase of the access has been taken, but then queues the access to the appropriate slave port to enter into arbitration for access to that slave port.

If the slave port is currently parked on another master or is in low-power park mode and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed (burst and locked transfers excluded). If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port once the other master releases control of the slave port, if no other higher priority master is also waiting for the slave port.

9.5.2.5 Error response terminated accesses

A master access is responded to with an error if the **hsel** input of the XBAR is asserted and the transfer type is non IDLE and the access decodes to a location not occupied by a slave port. This is the only time the XBAR responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

9.5.3 Slave ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. In order to do this, the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

The only instance when the XBAR forces a bubble onto the slave bus is when a master is actively making a request. This occurs when a higher priority master has control of the slave port and is running single clock (zero wait state) accesses while a lower priority master is stalled waiting for control of the slave port. When the higher priority master either leaves the slave port or runs an IDLE cycle to the slave port, the XBAR takes control of the slave bus and runs a single IDLE cycle before giving the slave port to the lower priority master that was waiting for control of the slave port.

The only other times the XBAR has control of the slave port are when the XBAR is halting, or when no masters are making access requests to the slave port and the XBAR is forced to either park the slave port on a specific master or put the slave port into low-power park mode.

In most instances when the XBAR has control of the slave port, it indicates IDLE for the transfer type, negates all control signals, and indicates ownership of the slave bus via the **hmaster** encoding of 4’b0000. One exception to this rule is when a master running locked cycles has left the slave port but continues to run locked cycles. In this case, the XBAR controls the slave port and indicates IDLE for the transfer type, but it does not affect any other signals.

NOTE

When a master runs a locked cycle through the XBAR, the master remains owning all slave ports it accesses while running locked cycles for one cycle beyond when the master finishes running locked cycles.

This page is intentionally left blank.

Chapter 10
Peripheral Bridge (PBRIDGE)

10.1 Introduction

The peripheral bridge (PBRIDGE) is the interface between the system bus and on-chip peripherals. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

The PBRIDGE automatically manages peripheral access requests from software. In other words, you can access a peripheral using its memory-mapped interface without having to configure the PBRIDGE. However, the PBRIDGE also supports special peripheral access, such as memory protection, that is available for you to use if you need it.

This device includes two instantiations of the PBRIDGE. These are called PBRIDGE_0 and PBRIDGE_1. In LSM, the base address for PBRIDGE_0 and PBRIDGE_1 is the same. In DPM, the base addresses are different. See Table 10-1.

10.2 Block interface

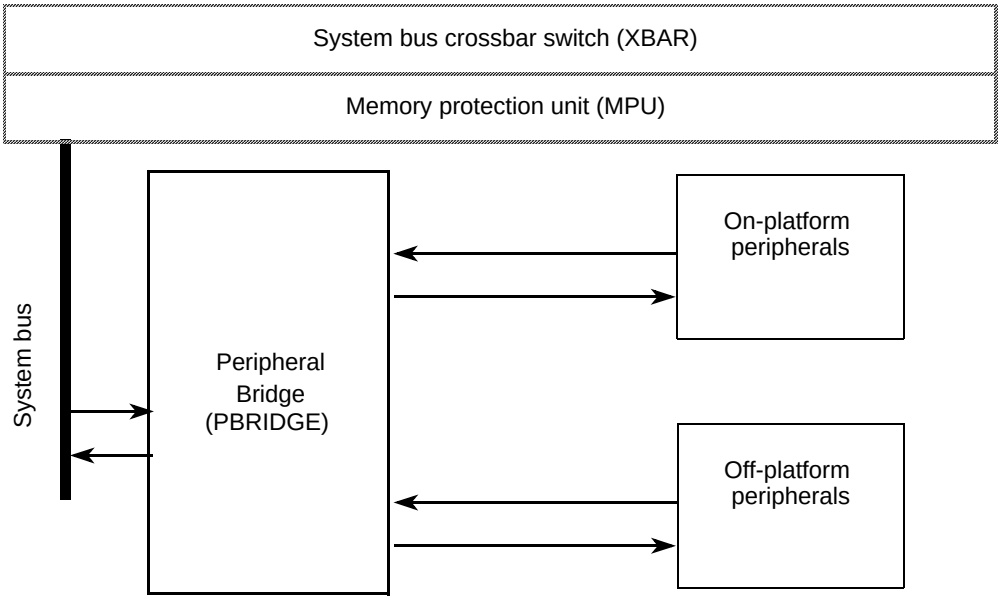


Figure 10-1. PBRIDGE interface

10.3 Features

The following list summarizes the key features of the PBRIDGE:

- Includes a 32-bit address bus and 64-bit data bus
- Supports 32-bit slave peripherals (byte, halfword, and word reads and writes are supported by the bridge)

- Provides configurable per-master access protections
- Provides configurable per-peripheral access protections for both on-platform and off-platform peripherals

10.4 Memory map and register description

10.4.1 Register access

All registers are 32-bit registers and can only be accessed in supervisor mode by trusted bus masters. Additionally, these registers must only be read from or written to by a 32-bit aligned access.

Two system clock cycles are required for read accesses and three system clock cycles are required for write accesses to the PBRIDGE registers.

10.4.2 Memory map

Each register in the PBRIDGE module has a size of 32 bits. The registers are listed in Table 10-2. The module organization is shown in Table 10-3. The organizational hierarchy is as follows:

- The module has multiple registers with the same register name (MPROT, PACR, OPACR), each at a different address offset.
- Each register has multiple similarly named fields, each with a different number.
- Each field has subfields as defined elsewhere in this section.

Accesses to registers or register fields marked as reserved return zeros on reads, and are ignored on writes.

Table 10-1. PBRIDGE module organization

Mode	Module	Module base address	Peripheral connections
LSM	PBRIDGE_0	0xFFF0_0000	All peripherals that communicate via the peripheral bridge
	PBRIDGE_1		
DPM	PBRIDGE_0	0xFFF0_0000 (same as LSM)	Peripherals with PCTL number less than 64[1]
	PBRIDGE_1	0x8FF0_0000	Peripherals with PCTL number greater than 63[1]

1 See Table 2-2 for a list of PCTL numbers.

Table 10-2. PBRIDGE registers

Offset from PBRIDGE_BASE	Register	Access [1]	Reset Value[2]	Location
0x0000–0x0007 [3]	Master Privilege Registers (MPROT*n*)[4]	R/W	—[5]	on page 256
0x0008–0x001F	Reserved			
0x0020–0x003F [3]	Peripheral Access Control Registers (PACR*n*)	R/W	—[5]	on page 257

Table 10-2. PBRIDGE registers (continued)

Offset from PBRIDGE_BASE	Register	Access [1]	Reset Value [2]	Location
0x0040–0x006F [3]	Off-Platform Peripheral Access Control Registers (OPACR*n*)	R/W	—[5]	on page 259
0x0070–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where "H" is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 This memory range contains reserved areas. See Table 10-3.

4 MPROT*n* mapping varies depending on LS/DP mode. See Table 10-3.

5 See the associated register description for more information.

Table 10-3. PBRIDGE memory map

Address offset	Register name	Bit numbers							
		0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
0x0000	MPROT[0:15] in LSM	MPROT0	Reserved	MPROT2	MPROT3	MPROT4	MPROT5	Reserved	Reserved
0x0004		MPROT8	Reserved						
0x0000	MPROT[0:15] in DPM	MPROT0	MPROT1	MPROT2	MPROT3	MPROT4	MPROT5	MPROT6	Reserved
0x0004		MPROT8	MPROT9	Reserved					
0x0008	Reserved								
0x000C									
0x0010									
0x0014									
0x0018									
0x001C									
0x0020	PACR	PACR0	PACR1	Reserved		PACR4	Reserved		
0x0024		Reserved	PACR9	Reserved				PACR14	PACR15
0x0028		PACR16	PACR17	PACR18	Reserved				
0x002C		Reserved							
0x0030	Reserved								
0x0034									
0x0038									
0x003C									
0x0040	OPACR	Reserved				OPACR4	OPACR5	OPACR6	Reserved
0x0044		Reserved							
0x0048		OPACR16	OPACR17	OPACR18	OPACR19	Reserved			OPACR23

Table 10-3. PBRIDGE memory map (continued)

Address offset	Register name	Bit numbers							
		0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
0x004C		OPACR24	Reserved						OPACR31
0x0050		OPACR32	OPACR33	Reserved	OPACR35	Reserved		OPACR38	OPACR39
0x0054		OPACR40	OPACR41	OPACR42	Reserved	OPACR44	OPACR45	OPACR46	Reserved
0x0058		OPACR48	OPACR49	OPACR50	OPACR51	Reserved			
0x005C		Reserved		OPACR58	OPACR59	Reserved			
0x0060		Reserved	OPACR65	OPACR66	OPACR67	OPACR68	OPACR69	OPACR70	Reserved
0x0064		Reserved			OPACR75 [1]	OPACR76	Reserved		
0x0068		Reserved						OPACR86	OPACR87
0x006C		OPACR88	OPACR89	OPACR90	Reserved	OPACR92	OPACR93	Reserved	
0x0070		Reserved							
0x0074		Reserved							
0x0078		OPACR112	Reserved			OPACR116	OPACR117	Reserved	OPACR119
0x007C		OPACR120	Reserved				OPACR125	Reserved	

[1] DPM only.

10.4.3 Register descriptions

10.4.3.1 Master privilege registers (MPROT)

Each MPROT register contains one or more 4-bit fields, called MPROT*n*, as shown in Table 10-3. Each of these fields defines the access privilege level associated with bus master *n* in the platform, as well as specifying whether write accesses from this master are bufferable. The registers provide one field per bus master. See the “Logical master IDs” section in the XBAR chapter for a list of master numbers and names.

Each MPROT*n* field has the structure described in Figure 10-2 and Table 10-4.

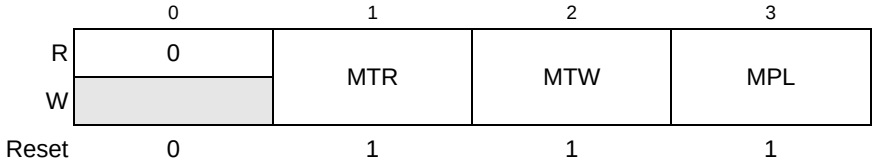


Figure 10-2. MPROT*n* field structure

Table 10-4. MPROT*n* field structure descriptions

Subfield	Description
MTR	Master Trusted for Reads. This bit determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
MTW	Master Trusted for Writes. This bit determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
MPL	Master Privilege Level. This bit determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

Table 10-5. MPROT*n* mapping

MPROT*n*	Mapping	
	LSM	DPM
MPROT0	e200z7_0 core complex Instruction port e200z7_0 core complex Load/Store port	e200z7_0 core complex Instruction port e200z7_0 core complex Load/Store port
	e200z7_1 core complex Instruction port e200z7_1 core complex Load/Store port	
MPROT1	—	e200z7_1 core complex Instruction port e200z7_1 core complex Load/Store port
MPROT2	DMA_0 DMA_1	DMA_0
MPROT3	FlexRay	FlexRay
MPROT4	Ethernet (FEC)	Ethernet (FEC)
MPROT5	PDI	PDI
MPROT6	—	DMA_1
MPROT7	—	—
MPROT8	e200z7_0 core Nexus e200z7_1 core Nexus	e200z7_0 core Nexus
MPROT9	—	e200z7_1 core Nexus

10.4.3.2 Peripheral access control registers (PACR)

Each PACR register contains one or more 4-bit fields, called PACR*n*, as shown in Table 10-3. Each of these fields defines the access levels supported by the associated module. The lists of modules and their corresponding numbers are shown in Table 10-7.

Each PACR*n* field has the structure described in Figure 10-3 and Table 10-6.

	0	1	2	3
R	0	SP	WP	TP
W				
Reset	0	1	0	0[1]

[1] The reset state of PACR0[TP] is 1, not 0. It is possible to write to PACR0 SP and TP and read back the value written, but it does not change the functionality of the PACR0 TP and SP bits. Functionally, they are tied to a '1' even though you can write a '0' to them.

Figure 10-3. PACR*n* field structure

Table 10-6. PACR*n* field structure descriptions

Subfield	Description
SP	Supervisor Protect. This bit determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The MPROTx[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
WP	Write Protect. This bit determines whether the peripheral allows write accesses. 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
TP	Trusted Protect. This bit determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.

Table 10-7. On-platform peripherals and PACR numbers

Sorted by PACR Number		Sorted Alphabetically	
PACR number	Peripheral	Peripheral	PACR number
0	PBRIDGE	ECSM	16
1	XBAR	eDMA	17
4	MPU	INTC	18
9	SEMA4	MPU	4
14	SWT	PBRIDGE	0
15	STM	SEMA4	9
16	ECSM	STM	15
17	eDMA	SWT	14
18	INTC	XBAR	1

10.4.3.3 Off-platform peripheral access control registers (OPACR)

The OPACR defines the access levels supported by the associated module. Each OPACR has a format identical to the PACR described in Section 10.4.3.2, Peripheral access control registers (PACR).

The lists of off-platform modules and their corresponding numbers are listed in Table 10-8. The OPACR number mirrors the PCTL column of Table 2-2 in Chapter 2, Memory Map.

Table 10-8. Off-platform peripherals and OPACR numbers

Sorted by OPACR number		Sorted alphabetically	
OPACR number	**Peripheral**	**Peripheral**	**OPACR number**
4	DSPI_0	32	ADC_0
5	DSPI_1	33	ADC_1
6	DSPI_2	116	ADC_2
16	FlexCAN_0	117	ADC_3
17	FlexCAN_1	31	BAM
18	FlexCAN_2	66	CFLASH_0
19	FlexCAN_3	76	CFLASH_1
23	DMA_MUX	58	CRC
24	FlexRay controller	120	CRCU_1
31	BAM	35	CTU_0
32	ADC_0	119	CTU_1
33	ADC_1	67	DFLASH_0
35	CTU_0	23	DMA_MUX
38	eTimer_0	75	DMACHMUX_1
39	eTimer_1	4	DSPI_0
40	eTimer_2	5	DSPI_1
41	FlexPWM_0	6	DSPI_2
42	FlexPWM_1	65	EBI
44	I^2C_0	38	eTimer_0
45	I^2C_1	39	eTimer _1
46	I^2C_2	40	eTimer__2
48	LINFlexD_0	59	FCCU
49	LINFlexD_1	16	FlexCAN_0
50	LINFlexD_2	17	FlexCAN_1
51	LINFlexD_3	18	FlexCAN_2
58	CRC	19	FlexCAN_3

Table 10-8. Off-platform peripherals and OPACR numbers (continued)

Sorted by OPACR number		Sorted alphabetically	
OPACR number	**Peripheral**	**Peripheral**	**OPACR number**
59	FCCU	41	FlexPWM_0
65	EBI	42	FlexPWM_1
66	CFLASH_0	125	FlexPWM_2
67	DFLASH_0	24	FlexRay controller
68	SIUL	44	I2C_0
69	WKPU	45	I2C_1
70	MDDRC	46	I2C_2
75[1]	DMACHMUX_1	48	LINFlexD_0
76	CFLASH_1	49	LINFlexD_1
86	SSCM	50	LINFlexD_2
87	MC_ME	51	LINFlexD_3
88	MC_CGM	88	MC_CGM
89	MC_RGM	87	MC_ME
90	MC_PCU	90	MC_PCU
92	PIT	89	MC_RGM
93	STCU	70	MDDRC
112	PDI	112	PDI
116	ADC_2	92	PIT
117	ADC_3	68	SIUL
119	CTU_1	86	SSCM
120	CRCU_1	93	STCU
125	FlexPWM_2	69	WKPU

1 DPM only

10.5 Functional description

The PBRIDGE serves as an interface between a system bus and the peripheral (slave) bus. It functions as a protocol translator. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

There is no need to configure the PBRIDGE registers unless the default master and/or peripheral access privileges need to be changed.

10.5.1 Access support

Aligned 32-bit word accesses, halfword accesses, and byte accesses are supported for the peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the PBRIDGE.

NOTE

Data accesses that cross a 32-bit boundary are not supported.

10.5.1.1 Peripheral write buffering

Buffered writes are not supported by the PBRIDGE.

10.5.1.2 Read cycles

Two-clock read accesses are possible with the PBRIDGE when the requested access size is 32 bits or smaller, and is not misaligned across a 32-bit boundary.

10.5.1.3 Write cycles

Three clock write accesses are possible with the PBRIDGE when the requested access size is 32 bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported.

10.5.2 General operation

Slave peripherals are modules that contain readable/writable control and status registers. The system bus master reads and writes these registers through the PBRIDGE. The PBRIDGE generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the slave peripherals. The PBRIDGE captures read data from the slave interface and drives it on the system bus.

The PBRIDGE occupies a 64 MB portion of the address space. The register maps of the slave peripherals are located on 16 KB boundaries. Each slave peripheral is allocated one 16 KB block of the memory map, and is activated by one of the module enables from the PBRIDGE.

The PBRIDGE is responsible for indicating to slave peripherals if an access is in supervisor or user mode. All eDMA and FlexRay transfers are done in supervisor mode.

This page is intentionally left blank.

Chapter 11
Analog-to-Digital Converter (ADC)

11.1 Introduction

The ADC module provides accurate and fast conversions for a wide range of applications. The ADC provides features for normal or injected conversion. A conversion can be triggered by software.

The mask registers present within the ADC can be programmed to configure which channel is to be converted.

A conversion timing register for configuring different sampling and conversion times is associated with each channel type.

Analog watchdogs allow continuous hardware monitoring.

Each member of the MPC5675K family has four ADC instantiations: ADC_0, ADC_1, ADC_2, and ADC_3.

11.2 Features

- 12-bit A/D resolution
- 16 input analog channels per ADC
- 2 different sampling and conversion time registers
- One-Shot/Scan modes
- Chain injection mode
- Triggered injection
- Presampling
- Abort single-channel or chain conversions
- 16 data registers for storing converted data
- Configurable analog watchdog channels
- Auto Clock off feature
- Configurable clock prescaler (ipg_clk divided by 2)
- Two different control modes (CPU Control Mode and CTU Control Mode)
- Self-testing feature
- 4 independent ADCs with 12-bit A/D resolution
- Conversion range of 0–5 V or 0–3.3 V (dependent on reference voltage)
- DMA and interrupt request support

11.2.1 External connections

The MPC5675K ADC modules provide the following external inputs:

- 9 external channels on ADC_0

- 9 external channels on ADC_1
- 4 external channels shared between ADC_0 and ADC_1
- 4 external channels on ADC_2
- 4 external channels on ADC_3
- 4 external channels shared between ADC_2 and ADC_3

11.2.2 Internal connections

The MPC5675K ADC modules provide the following internal connections:

Table 11-1. ADC internal connections

ADC Module	Connection	ADC channel
ADC_0	PMU	adc0 AN[9]
	VDD_LV_COR	adc0 AN[10]
	TSENS	adc0 AN[15]
ADC_1	PMU	adc1 AN[9]
	VDD_HV_PDI	adc1 AN[10]
	VDD_HV_DRAM	adc1 AN[15]
ADC_2	PMU	adc2 AN[9]
ADC_3	PMU	adc3 AN[9]

For more information on internal connections, see Figure 11-44, Figure 11-45, and Section 11.4.1, ADC channel muxing.

11.2.3 Inter-module communication

Figure 11-1 shows the modules that communicate with the ADC.

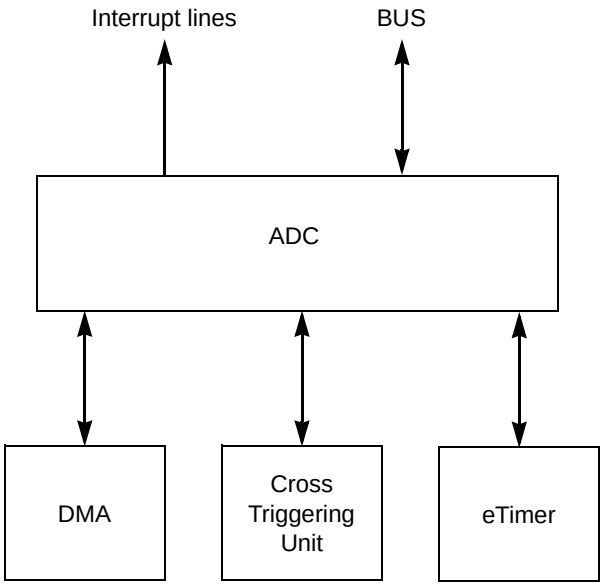


Figure 11-1. ADC interaction with other modules

Figure 11-2 shows the CTU / eTimer / ADC interface. Each ADC can be controlled by the CPU (CPU Control Mode) or by the CTU (CTU Control Mode). The CTU can control the ADC sending an ADC command only when the ADC is in CTU Control Mode. The control mode is selected via the MCU[CTUEN] configuration bit. During the CTU Control Mode, the CPU is able to write in the ADC registers but it cannot start a new conversion.

eTimer1 channel 5 is connected to the injection trigger inputs of ADC_0 and ADC_1. eTimer2 channel 5 is connected to the injection trigger inputs of ADC_2 and ADC_3.

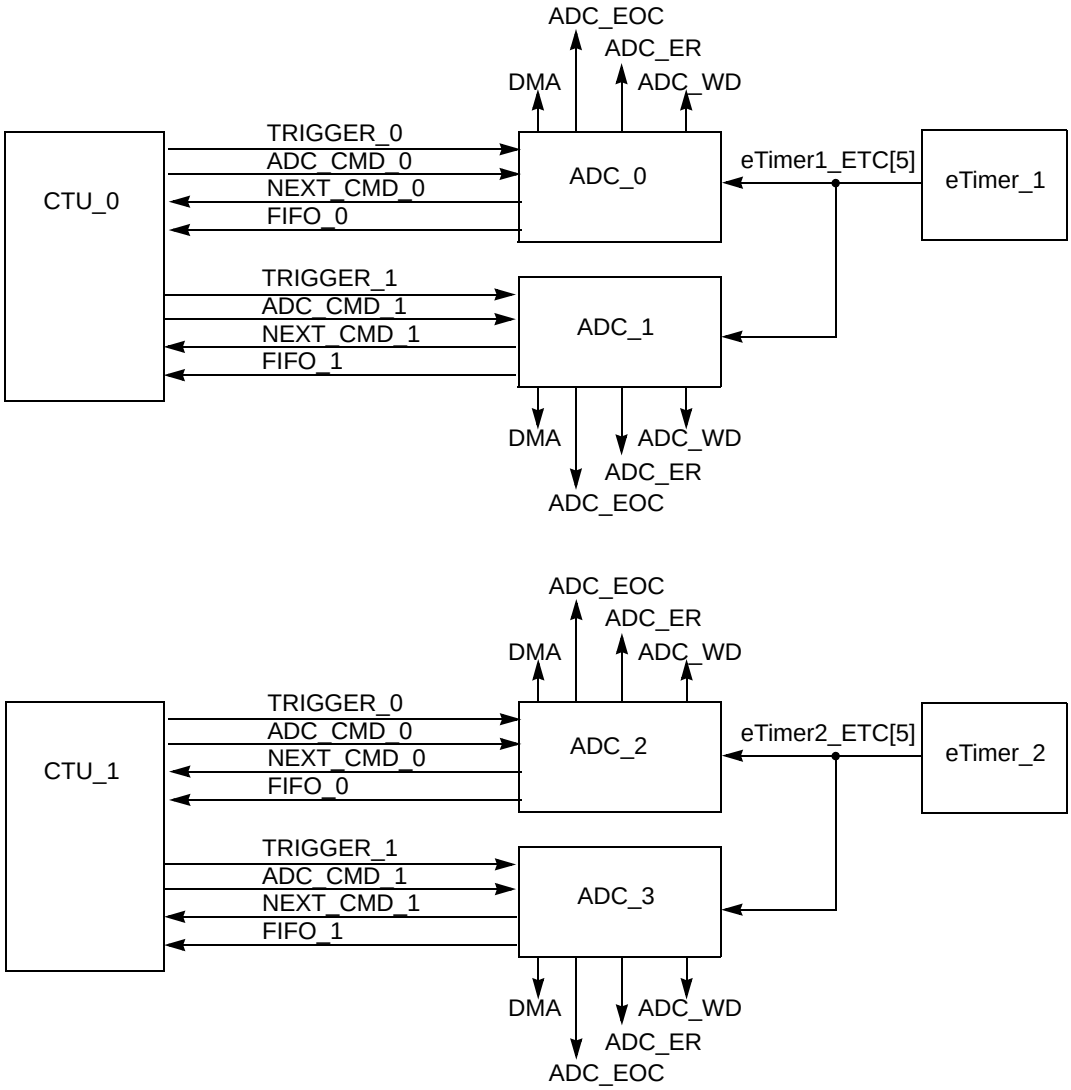


Figure 11-2. CTU / eTimer / ADC interface

11.2.4 Debug mode

When the MCU is in debug mode, the ADC behavior is unaffected and remains identical to its operation in normal mode.

11.3 Memory map and register descriptions

Table 11-2 shows the base addresses for the ADC modules. Addresses are the same for Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). Table 11-3 shows the memory map for the ADC program-visible registers.

Table 11-2. ADC module base addresses

Mode	Module	Module base address
LSM and DPM	ADC_0	0xFFE0_0000
	ADC_1	0xFFE0_4000
	ADC_2	0xC3E5_0000
	ADC_3	0xC3E5_4000

Table 11-3. ADC memory map

Offset from ADC_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Main Configuration Register (MCR)	R/W	0x0000_0001	on page 270
0x0004	Main Status Register (MSR)	R	0x0000_0001	on page 272
0x0008–0x000F	Reserved			
0x0010	Interrupt Status Register (ISR)	R/W	0x0000_0000	on page 273
0x0014	Channel Pending Register 0 (CEOCFR0)	R/W	0x0000_0000	on page 274
0x0018–0x001F	Reserved			
0x0020	Interrupt Mask Register (IMR)	R/W	0x0000_0000	on page 274
0x0024	Channel Interrupt Mask Register 0 (CIMR0)	R/W	0x0000_0000	on page 275
0x0028–0x002F	Reserved			
0x0030	Watchdog Threshold Interrupt Register (WTISR)	R/W	0x0000_0000	on page 276
0x0034	Watchdog Threshold Interrupt Mask Reg (WTIMR)	R/W	0x0000_0000	on page 277
0x0038–0x003F	Reserved			
0x0040	DMA Enable Register (DMAE)	R/W	0x0000_0000	on page 277
0x0044	DMA Channel Select Register 0 (DMAR0)	R/W	0x0000_0000	on page 278
0x0048–0x005F	Reserved			
0x0060	Threshold Register 0 (THRHLR0)	R/W	0x0FFF_0000	on page 278
0x0064	Threshold Register 1 (THRHLR1)	R/W	0x0FFF_0000	on page 278
0x0068	Threshold Register 2 (THRHLR2)	R/W	0x0FFF_0000	on page 278
0x006C	Threshold Register 3 (THRHLR3)	R/W	0x0FFF_0000	on page 278
0x0070–0x007F	Reserved			
0x0080	Presampling Control Register (PSCR)	R/W	0x0000_0000	on page 279
0x0084	Presampling Register 0 (PSR0)	R/W	0x0000_0000	on page 280
0x0088–0x0093	Reserved			

Table 11-3. ADC memory map (continued)

Offset from ADC_BASE	Register	Access[1]	Reset Value[2]	Location
0x0094	Conversion Timing Register 0 (CTR0)	R/W	0x0000_0203	on page 280
0x0098	Conversion Timing Register 1 (CTR1)	R/W	0x0000_0203	on page 282
0x009C–0x00A3	Reserved			
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	R/W	0x0000_0000	on page 282
0x00A8–0x00B3	Reserved			
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	R/W	0x0000_0000	on page 283
0x00B8–0x00C7	Reserved			
0x00C8	Power-down Exit Delay Register (PDEDR)	R/W	0x0000_0000	on page 284
0x00CC–0x00FF	Reserved			
0x0100	Channel 0 Data Register (CDR0)	R/W	0x0000_0000	on page 284
0x0104	Channel 1 Data Register (CDR1)	R/W	0x0000_0000	on page 284
0x0108	Channel 2 Data Register (CDR2)	R/W	0x0000_0000	on page 284
0x010C	Channel 3 Data Register (CDR3)	R/W	0x0000_0000	on page 284
0x0110	Channel 4 Data Register (CDR4)	R/W	0x0000_0000	on page 284
0x0114	Channel 5 Data Register (CDR5)	R/W	0x0000_0000	on page 284
0x0118	Channel 6 Data Register (CDR6)	R/W	0x0000_0000	on page 284
0x011C	Channel 7 Data Register (CDR7)	R/W	0x0000_0000	on page 284
0x0120	Channel 8 Data Register (CDR8)	R/W	0x0000_0000	on page 284
0x0124	Channel 9 Data Register (CDR9)	R/W	0x0000_0000	on page 284
0x0128	Channel 10 Data Register (CDR10)	R/W	0x0000_0000	on page 284
0x012C	Channel 11 Data Register (CDR11)	R/W	0x0000_0000	on page 284
0x0130	Channel 12 Data Register (CDR12)	R/W	0x0000_0000	on page 284
0x0134	Channel 13 Data Register (CDR13)	R/W	0x0000_0000	on page 284
0x0138	Channel 14 Data Register (CDR14)	R/W	0x0000_0000	on page 284
0x013C	Channel 15 Data Register (CDR15)	R/W	0x0000_0000	on page 284
0x0140–0x027F	Reserved			
0x0280	Threshold Register 4 (THRHLR4)	R/W	0x0FFF_0000	on page 278
0x0284	Threshold Register 5 (THRHLR5)	R/W	0x0FFF_0000	on page 278
0x0288	Threshold Register 6 (THRHLR6)	R/W	0x0FFF_0000	on page 278
0x028C	Threshold Register 7 (THRHLR7)	R/W	0x0FFF_0000	on page 278

Table 11-3. ADC memory map (continued)

Offset from ADC_BASE	Register	Access[1]	Reset Value[2]	Location
0x0290	Threshold Register 8 (THRHLR8)	R/W	0x0FFF_0000	on page 278
0x0294	Threshold Register 9 (THRHLR9)	R/W	0x0FFF_0000	on page 278
0x0298	Threshold Register 10 (THRHLR10)	R/W	0x0FFF_0000	on page 278
0x029C	Threshold Register 11 (THRHLR11)	R/W	0x0FFF_0000	on page 278
0x02A0	Threshold Register 12 (THRHLR12)	R/W	0x0FFF_0000	on page 278
0x02A4	Threshold Register 13 (THRHLR13)	R/W	0x0FFF_0000	on page 278
0x02A8	Threshold Register 14 (THRHLR14)	R/W	0x0FFF_0000	on page 278
0x02AC	Threshold Register 15 (THRHLR15)	R/W	0x0FFF_0000	on page 278
0x02B0	Channel Watchdog Selection Register 0 (CWSEL0)	R/W	0x0000_0000	on page 285
0x02B4	Channel Watchdog Selection Register 1 (CWSEL1)	R/W	0x0000_0000	on page 285
0x02B8–0x02DF	Reserved			
0x02E0	Channel Watchdog Enable Register 0(CWENR0)	R/W	0x0000_0000	on page 286
0x02E4–0x02EF	Reserved			
0x02F0	Analog Watchdog Out of Range Register 0 (AWORR0)	R/W	0x0000_0000	on page 287
0x02F4–0x033F	Reserved			
0x0340	Self-Test Configuration Register 1 (STCR1)	R/W	0x1818_2507	on page 287
0x0344	Self-Test Configuration Register 2 (STCR2)	R/W	0x0000_0005	on page 288
0x0348	Self-Test Configuration Register 3 (STCR3)	R/W	0x0000_0300	on page 290
0x034C	Self-Test Baud Rate Register (STBRR)	R/W	0x0005_0000	on page 291
0x0350	Self-Test Status Register 1 (STSR1)	R/W	0x0000_0000	on page 292
0x0354	Self-Test Status Register 2 (STSR2)	R	0x0000_0000	on page 294
0x03358	Self-Test Status Register 3 (STSR3)	R	0x0000_0000	on page 294
0x035C	Self-Test Status Register 4 (STSR4)	R	0x0000_0000	on page 295
0x0360–0x036F	Reserved			
0x0370	Self-Test Data Register 1 (STDR1)	R	0x0000_0000	on page 296
0x0374	Self-Test Data Register 2 (STDR2)	R	0x0000_0000	on page 296
0x0378–0x037F	Reserved			

Table 11-3. ADC memory map (continued)

Offset from ADC_BASE	Register	Access[1]	Reset Value[2]	Location
0x0380	Self-Test Analog Watchdog Register 0 (STAW0R)	R/W	0x0727_04C5	on page 297
0x0384	Self-Test Analog Watchdog Register 1A (STAW1AR)	R/W	0x0003_0001	on page 298
0x0388	Self-Test Analog Watchdog Register 1B (STAW1BR)	R/W	0x03E8_0ED0	on page 298
0x038C	Self-Test Analog Watchdog Register 2 (STAW2R)	R/W	0x0000_0FF9	on page 299
0x0390	Self-Test Analog Watchdog Register 3 (STAW3R)	R/W	0x086D_0793	on page 299
0x0394	Self-Test Analog Watchdog Register 4 (STAW4R)	R/W	0x0828_07D8	on page 300
0x0398	Self-Test Analog Watchdog Register 5 (STAW5R)	R/W	0x0010_0010	on page 301
0x039C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

11.3.1 Register descriptions

11.3.1.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R				0	0	0	0		0					0		0
W	OWREN	WLSIDE	MODE					NSTART		JTRGEN	JEDGE	JSTART	REF_RANGE_EXP		CTUEN	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R		0	0	0	0	0	0					0	0	0	0	
W	STCL							ADCLKSEL	ABORTCHAIN	ABORT	ACKO					PWDN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 11-3. Main Configuration Register (MCR)

Table 11-4. MCR field descriptions

Bit	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded. 1 Enables converted data to be overwritten by a new conversion.
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 16 to 27). The WLSIDE bit affects all the CDR registers simultaneously. See Figure 11-22.
MODE	One-Shot/Scan 0 One-Shot Mode—Configures the normal conversion of one chain. 1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.
NSTART	Normal Start conversion Setting this bit starts the chain or scan conversion. Resetting this bit during Scan mode causes the current chain conversion to finish, then stops the operation. 0 Causes the current chain conversion to finish and stops the operation. 1 Starts the chain or scan conversion.
JTRGEN	Injection external trigger enable 0 External trigger disabled for channel injection (injected conversion cannot be started using an external signal). 1 External trigger enabled for channel injection.
JEDGE	Injection trigger edge selection Edge selection for external trigger, if JTRGEN = 1. 0 Selects falling edge for the external trigger. 1 Selects rising edge for the external trigger.
JSTART	Injection start Setting this bit starts the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.
REF_RANGE_EXP	Expected reference range This control bit specifies the expected value of the MSR[REF_RANGE] bit. If the expected value does not match with the actual value, the ISR[REF_RANGE] bit is set.
CTUEN	Cross Trigger Unit Enable 0 The cross triggering unit is disabled and the triggered injected conversion cannot take place. 1 The cross triggering unit is enabled and the triggered injected conversion can take place.
STCL	Self-Testing Configuration Lock 0 No lock. 1 The self-testing configuration is locked. The following bits are write-protected: STCR1, STCR2, STCR3, STBRR, STAW0R, STAW1AR, STAW1BR, STAW2R, STAW3R, STAW4R, and STAW5R. This bit can be used only in CPU and SCAN mode, and is cleared only by a peripheral reset.
ADCLKSEL	Analog clock frequency selector 0 The AD_clk frequency is half of the ipg_clk frequency. 1 The AD_clk frequency equals the ipg_clk frequency. This bit can be written in power-down only.

Table 11-4. MCR field descriptions (continued)

Bit	Description
ABORTCHAIN	Abort Chain When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested. 0 Conversion is not affected. 1 Aborts the ongoing chain conversion.
ABORT	Abort Conversion When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. 0 Conversion is not affected. 1 Aborts the ongoing conversion.
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off is disabled. 1 Auto clock off is enabled.
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode. 1 ADC has been requested to power down.

NOTE

When CTU Trigger mode is used, do not use injected conversion.

11.3.1.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	N START	J ABORT	0	0	J START	REF_RANGE	SELF_TEST_S	0	CTUSTART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHADDR							0	0	0	ACKO	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 11-4. Main Status Register (MSR)

Table 11-5. MSR field descriptions

Field	Description
NSTART	This status bit signals that a normal conversion is ongoing. This bit stays high while the conversion is ongoing (or pending during injection mode).
JABORT	This status bit signals that an injected conversion has been aborted. This bit is reset when a new injected conversion starts.
JSTART	This status bit signals that an injected conversion is ongoing.
REF_RANGE	This bit defines the voltage range for operation of the ADC. It is provided as an output by the ADC along with data after every conversion.
SELF_TEST_S	This status bit signals that self-test conversion is ongoing.
CTUSTART	This status bit signals that a CTU conversion is ongoing. This bit is set when a CTU trigger pulse is received and the CTU conversion starts. When CTU trigger mode is enabled this bit is automatically reset when the conversion is completed. Otherwise, if Control Mode is enabled this bit is reset when the CTU is disabled (CTUEN = 0).
CHADDR	Channel under measure address. This field signals which channel is under measure.
ACKO	Auto-clock-off enable. This status bit signals if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 IDLE 001 Power-down 010 Wait state 011 — 100 Sample 101 — 110 Conversion 111 —

11.3.1.3 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REF_RANGE	0	0	0	0	0	0	0	0	0	0	EOCTU	JEOC	JECH	EOC	ECH
W	w1c[1]											w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-5. Interrupt Status Register (ISR)

[1] Write 1 to clear: indicates that writing a 1 to this field clears it.

Table 11-6. ISR field descriptions

Field	Description
REF_RANGE	This bit is set if the REF_RANGE output from the ADC does not match with expected value programmed in the MCR register.
EOCTU	End of CTU Conversion interrupt (EOCTU) flag. It is the interrupt of the digital end of conversion for the CTU channel; active when set. When this bit is set, an EOCTU interrupt has occurred. Writing a 1 to this bit clears it. Writing a 0 has no effect.
JEOC	End of Injected Channel Conversion interrupt (JEOC) flag. It is the interrupt of the digital end of conversion for the injected channel; active when set. When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt (JECH) flag. It is the interrupt of the digital end of chain conversion for the injected channel; active when set. When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt (EOC) flag. It is the interrupt of the digital end of conversion. When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt (ECH) flag. It is the interrupt of the digital end of chain conversion. When this bit is set, an ECH interrupt has occurred.

11.3.1.4 Channel Pending Register 0 (CEOCFR0)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EOC_CH15	EOC_CH14	EOC_CH13	EOC_CH12	EOC_CH11	EOC_CH10	EOC_CH9	EOC_CH8	EOC_CH7	EOC_CH6	EOC_CH5	EOC_CH4	EOC_CH3	EOC_CH2	EOC_CH1	EOC_CH0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-6. Channel Pending Register 0 (CEOCFR0)

Table 11-7. CEOFR*n* field descriptions

Field	Description
EOC_CH*n*	This field indicates the end of conversion. 0 The measure of channel *n* is not complete. 1 The measure of channel *n* is complete.

11.3.1.5 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MSK_REF_RANGE	0	0	0	0	0	0	0	0	0	0	MSKEOCTU	MSKJEOC	MSKJECH	MSKEOC	MSKECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-7. Interrupt Mask Register (IMR)

Table 11-8. IMR field descriptions

Field	Description
MSK_REF_RANGE	When this bit is set, an interrupt corresponding to the REF_RANGE bit in ISR is enabled.
MSKEOCTU	Mask bit for EOCTU. When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask bit for JEOC. When set, the JEOC interrupt is enabled.
MSKJECH	Mask bit for JECH. When set, the JECH interrupt is enabled.
MSKEOC	Mask bit for EOC. When set, the EOC interrupt is enabled.
MSKECH	Mask bit for ECH. When set, the ECH interrupt is enabled.

11.3.1.6 Channel Interrupt Mask Register 0 (CIMR0)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIM15	CIM14	CIM13	CIM12	CIM11	CIM10	CIM9	CIM8	CIM7	CIM6	CIM5	CIM4	CIM3	CIM2	CIM1	CIM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-8. Channel Interrupt Mask Register 0 (CIMR0)

Table 11-9. CIMR*n* field descriptions

Field	Description
CIM*n*	This field enables the interrupt for channel *n*. 0 Interrupt for channel *n* is disabled. 1 Interrupt for channel *n* is enabled.

11.3.1.7 Watchdog Threshold Interrupt Status Register (WTISR)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDG15H	WDG15L	WDG14H	WDG14L	WDG13H	WDG13L	WDG12H	WDG12L	WDG11H	WDG11L	WDG10H	WDG10L	WDG9H	WDG9L	WDG8H	WDG8L
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WDG7H	WDG7L	WDG6H	WDG6L	WDG5H	WDG5L	WDG4H	WDG4L	WDG3H	WDG3L	WDG2H	WDG2L	WDG1H	WDG1L	WDG0H	WDG0L
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-9. Watchdog Threshold Interrupt Status Register (WTISR)

Table 11-10. WTISR field descriptions

Field	Description
WDG*x*H	This field indicates whether an interrupt has been generated because the converted value is higher than the programmed higher threshold. 0 Converted value is lower or equal to the programmed higher threshold (no interrupt generated). 1 Converted value is higher than the programmed higher threshold (interrupt is generated).
WDG*x*L	This field indicates whether an interrupt has been generated because the converted value is lower than the programmed lower threshold. 0 Converted value is higher or equal to the programmed lower threshold (no interrupt generated). 1 Converted value is lower than the programmed lower threshold (interrupt is generated).

11.3.1.8 Watchdog Threshold Interrupt Mask Register (WTIMR)

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MSKWDG 15H	MSKWDG15L	MSKWDG 14H	MSKWDG14L	MSKWDG13H	MSKWDG 13L	MSKWDG 12H	MSKWDG 12L	MSKWDG 11H	MSKWDG 11L	MSKWDG10H	MSKWDG10L	MSKWDG 9H	MSKWDG9L	MSKWDG8H	MSKWDG8L
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MSKWDG 7H	MSKWDG 7L	MSKWDG6H	MSKWDG6L	MSKWDG5H	MSKWDG 5L	MSKWDG 4H	MSKWDG4L	MSKWDG 3H	MSKWDG 3L	MSKWDG2H	MSKWDG2L	MSKWDG 1H	MSKWDG1L	MSKWDG0H	MSKWDG0L
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-10. Watchdog Threshold Interrupt Mask Register (WTIMR)

Table 11-11. WTIMR field descriptions

Field	Description
MSKWDG*x*H	Mask bit for the interrupt generated because the converted value is higher than the programmed higher threshold. 0 Interrupt is not enabled. 1 Interrupt is enabled.
MSKWDG*x*L	Mask bit for the interrupt generated because the converted value is lower than the programmed lower threshold. 0 Interrupt is not enabled. 1 Interrupt is enabled.

11.3.1.9 DMA Enable Register (DMAE)

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DCLR	DMAEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-11. DMA Enable Register (DMAE)

Table 11-12. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller. 1 DMA request is cleared when a read of data registers occurs.
DMAEN	DMA global enable 0 DMA feature is disabled. 1 DMA feature is enabled.

11.3.1.10 DMA Channel Select Register 0 (DMAR0)

Address: Base + 0x044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA15	DMA14	DMA13	DMA12	DMA11	DMA10	DMA9	DMA8	DMA7	DMA6	DMA5	DMA4	DMA3	DMA2	DMA1	DMA0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-12. DMA Channel Select Register 0 (DMAR0)

Table 11-13. DMAR0 field descriptions

Field	Description
DMA*n*	DMA enable 0 DMA transfer for channel *n* is disabled. 1 Channel *n* is enabled to transfer data in DMA mode.

11.3.1.11 Threshold Registers (THRHLR*n*)

Address: See Table 11-15. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	THRH											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-13. Threshold Registers (THRHLR*n*)

Table 11-14. THRHLR*n* field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

Table 11-15. THRHLR*n* addresses

THRHLR*n*	Address (offset from base)	THRHLR*n*	Address (offset from base)
Threshold Register 0 (THRHLR0)	0x0060	Threshold Register 8 (THRHLR8)	0x0290
Threshold Register 1 (THRHLR1)	0x0064	Threshold Register 9 (THRHLR9)	0x0294
Threshold Register 2 (THRHLR2)	0x0068	Threshold Register 10 (THRHLR10)	0x0298
Threshold Register 3 (THRHLR3)	0x006C	Threshold Register 11 (THRHLR11)	0x029C
Threshold Register 4 (THRHLR4)	0x0280	Threshold Register 12 (THRHLR12)	0x02A0
Threshold Register 5 (THRHLR5)	0x0284	Threshold Register 13 (THRHLR13)	0x02A4
Threshold Register 6 (THRHLR6)	0x0288	Threshold Register 14 (THRHLR14)	0x02A8
Threshold Register 7 (THRHLR7)	0x028C	Threshold Register 15 (THRHLR15)	0x02AC

NOTE

The THRHLR*n* address space is not contiguous. A gap (containing other ADC registers) exists between THRHLR3 and THRHLR4.

11.3.1.12 Presampling Control Register (PSCR)

Address: Base + 0x080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	PREVAL1		PREVAL0		PRECONV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-14. Presampling Control Register (PSCR)

Table 11-16. PSCR field descriptions

Field	Description
PREVAL1	Internal voltage selection for presampling of internal channels. 00 Selects the VDD_HV_ADR presampling voltage for this ADC. 01 Selects the VSS_HV_ADR presampling voltage for this ADC. 10 Reserved. 11 Reserved.
PREVAL0	Internal voltage selection for presampling of external channels. 00 Selects the VDD_HV_ADR presampling voltage for this ADC. 01 Selects the VSS_HV_ADR presampling voltage for this ADC. 10 Reserved. 11 Reserved.
PRECONV	Convert presampled value 0 The ADC performs a sampling followed by a conversion. The order of events is: Pre-sample, Sample, Convert. 1 The ADC performs a presampling followed by a conversion. The order of events is: Pre-sample, Convert. The sampling is bypassed, so the conversion result is that of the presampled value. **Note:** Regardless of this bit's setting, presampling is performed.

11.3.1.13 Presampling Register 0 (PSR0)

Address: Base + 0x084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRES15	PRES14	PRES13	PRES12	PRES11	PRES10	PRES9	PRES8	PRES7	PRES6	PRES5	PRES4	PRES3	PRES2	PRES1	PRES0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-15. Presampling Register 0 (PSR0)

Table 11-17. PSR0 field descriptions

Field	Description
PRES*n*	Presampling enable 0 Presampling for channel *n* is disabled. 1 Presampling for channel *n* is enabled.

11.3.1.14 Conversion Timing Register 0 (CTR0)

The Conversion Timing Register 0 (CTR0) configures the conversion timing for external channels. Timings for internal channels are configured using CTR1 (see Section 11.3.1.15, Conversion Timing Register 1 (CTR1)).

Address: Base + 0x094 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Figure 11-16. Conversion Timing Register 0 (CTR0)

Table 11-18. CTR0 field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration (see Figure 11-17).
OFFSHIFT	Configuration for offset shift characteristic. 00 No shift (that is the transition between codes 000h and 001h) is reached when the A_{VIN} (analog input voltage) is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the A_{VIN} is equal to1/2 LSB. 10 Transition between code 000h and 001h is reached when the A_{VIN} is equal to 0. 11 Reserved.
INPCMP	Configuration bits for comparison phase duration (see Table 11-19).
INPSAMP	Configuration bits for sampling phase duration (see Figure 11-17).

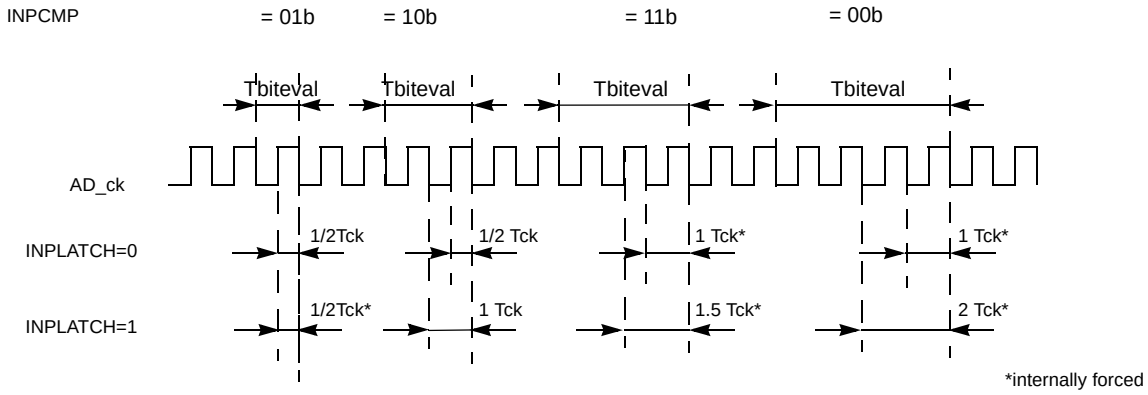


Figure 11-17. Conversion timing

Table 11-19. xMinimum AD_ck frequency

INPCMP	AD_ck min. freq (MHz)	$T_{biteval}$ in Eqn. 11-3
00	12	$4*T_{CK}$
01	3	$1*T_{CK}$

Table 11-19. xMinimum AD_ck frequency (continued)

INPCMP	AD_ck min. freq (MHz)	$T_{biteval}$ in Eqn. 11-3
10	6	$2*T_{CK}$
11	9	$3*T_{CK}$

11.3.1.15 Conversion Timing Register 1 (CTR1)

The Conversion Timing Register 1 (CTR1) configures the conversion timings for internal channels.

Address: Base + 0x098 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	0	0	0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1

Figure 11-18. Conversion Timing Register 1 (CTR1)

Table 11-20. CTR1 field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration (see Figure 11-17).
INPCMP	Configuration bits for comparison phase duration (see Table 11-19).
INPSAMP	Configuration bits for sampling phase duration (see Figure 11-17). When converting the TSENS internal channel 15 using ADC0, the LSB of this field CTR1[31] selects the operating mode of the TSENS module. 0 Selects PTAT mode. 1 Selects CTAT mode.

11.3.1.16 Normal Conversion Mask Register 0 (NCMR0)

The Normal Conversion Mask Register 0 (NCMR0) programs which of the input channels are converted during normal conversion.

Address: Base + 0x0A4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-19. Normal Conversion Mask Register 0 (NCMR0)

Table 11-21. NCMR0 field descriptions

Field	Description
CH*n*	Sampling enable 0 Sampling is disabled for channel *n.* 1 Sampling is enabled for channel *n*.

11.3.1.17 Injected Conversion Mask Register 0 (JCMR0)

The Injected Conversion Mask Register 0 (JCMR0) programs which of the input channels are converted during injected conversion.

Address: Base + 0x0B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-20. Injected Conversion Mask Register 0 (JCMR0)

Table 11-22. JCMR0 field descriptions

Field	Description
CH*n*	Sampling enable. 0 Sampling is disabled for channel *n*. 1 Sampling is enabled for channel *n*.

11.3.1.18 Power-down Exit Delay Register (PDEDR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PDED							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-21. Power-down Exit Delay Register (PDEDR)

Table 11-23. PDEDR field descriptions

Field	Description
PDED	The delay between the power-down bit reset and the start of conversion. The power-down delay is calculated as: PDED × 1/frequency of ADC clock.

11.3.1.19 Channel Data Register *n* (CDR*n*)

ADC conversion results are stored in data registers, with one register per channel. Each data register also gives information regarding the corresponding result as described below.

Address: See Table 11-3. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	CDATA (MCR[WLSIDE] = 0)											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDATA (MCR[WLSIDE] = 1)												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-22. Channel Data Register *n* (CDR*n*)

Table 11-24. CDR*n* field descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data. This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]: • When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read. • When OWREN = 1, then OVERW flags the CDATA field overwrite status. 0 Converted data has not been overwritten. 1 Previous converted data has been overwritten before having been read.
RESULT	This field reflects the mode of conversion for the corresponding channel. 00 Data is a result of normal conversion mode. 01 Data is a result of injected conversion mode. 10 Data is a result of CTU conversion mode. 11 Reserved.
CDATA	Channel *n* converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this field can be changed as shown in Figure 11-22.

11.3.1.20 Channel Watchdog Selection Registers (CWSEL*n*)

Address: Base + 0x2B0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WSEL_CH7				WSEL_CH6				WSEL_CH5				WSEL_CH4			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WSEL_CH3				WSEL_CH2				WSEL_CH1				WSEL_CH0			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-23. Channel Watchdog Selection Register 0 (CWSEL0)

Address: Base + 0x2B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WSEL_CH15				WSEL_CH14				WSEL_CH13				WSEL_CH12			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WSEL_CH11				WSEL_CH10				WSEL_CH9				WSEL_CH8			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-24. Channel Watchdog Selection Register 1 (CWSEL1)

Table 11-25. CWSEL*n* field descriptions

Field	Description
WSEL_CH*n*	Selects the threshold register that provides the values to be used for upper and lower thresholds for channel *n*. 0000 THRHLR0 register is selected. 0001 THRHLR1 register is selected. . . . 1110 THRHLR14 register is selected. 1111 THRHLR15 register is selected.

11.3.1.21 Channel Watchdog Enable Register 0 (CWENR0)

Address: Base + 0x2E0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CWEN15	CWEN14	CWEN13	CWEN12	CWEN11	CWEN10	CWEN9	CWEN8	CWEN7	CWEN6	CWEN5	CWEN4	CWEN3	CWEN2	CWEN1	CWEN0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-25. Channel Watchdog Enable Register 0 (CWENR0)

Table 11-26. CWENR0 field descriptions

Field	Description
CWEN*n*	Enables the watchdog feature for channel *n* 0 The watchdog feature for channel *n* is disabled. 1 The watchdog feature for channel *n* is enabled.

11.3.1.22 Analog Watchdog Out of Range Register 0 (AWORR0)

Address: Base + 0x2F0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AWOR_CH15	AWOR_CH14	AWOR_CH13	AWOR_CH12	AWOR_CH11	AWOR_CH10	AWOR_CH9	AWOR_CH8	AWOR_CH7	AWOR_CH6	AWOR_CH5	AWOR_CH4	AWOR_CH3	AWOR_CH2	AWOR_CH1	AWOR_CH0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-26. Analog Watchdog Out of Range Register 0 (AWORR0)

Table 11-27. AWORR0 field descriptions

Field	Description
AWOR_CH*n*	Out of range indicator 0 Channel *n* converted data is in range. 1 Channel *n* converted data is out of range.

11.3.1.23 Self-Test Configuration Register 1 (STCR1)

Address: Base + 0x340 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INPSAMP_C								INPSAMP_RC							
Reset	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPSAMP_S								0	0	0	0	0	ST_INPCMP		ST_INPLATCH
W																
Reset	0	0	1	0	0	1	0	1	0	0	0	0	0	1	1	1

Figure 11-27. Self-Test Configuration Register 1 (STCR1)

Table 11-28. STCR1 field descriptions

Field	Description
INPSAMP_C	Sampling phase duration for the test conversions related to Algorithm C. 0x00 Reserved . . . 0x17 Reserved 0x18 Minimum value . . . 0xFF Maximum value
INPSAMP_RC	Sampling phase duration for the test conversions related to Algorithm RC. 0x00 Reserved . . . 0x5F Reserved 0x60 Minimum value . . . 0xFF Maximum value
INPSAMP_S	Sampling phase duration for the test conversions related to Algorithm S. 0x00 Reserved . . . 0xFE Reserved 0xFF Sampling phase duration value
ST_INPCMP	Configuration bits for comparison phase duration for self-test channel (as in Table 11-19 for normal conversion).
ST_INPLATCH	Configuration bits for latching phase duration for self-test channel (as in Figure 11-17 for normal conversion).

11.3.1.24 Self-Test Configuration Register 2 (STCR2)

Address: Base + 0x344 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	MSKWDSERR	0	MSKWDTERR	0	MSKST_EOC	0	0	0	0	MSKWDG_EOA_C	MSKWDG_EOA_RC	MSKWDG_EOA_S
W						SERR										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MSKERR_C	MSKERR_RC	MSKERR_S2	MSKERR_S1	MSKERR_S0	0	0	0	EN	0	0	FMA_WDSERR	FMA_WDTERR	FMA_C	FMA_RC	FMA_S
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 11-28. Self-Test Configuration Register 2 (STCR2)

Table 11-29. STCR2 field descriptions

Field	Description
MSKWDSERR	Interrupt enable (STSR1[WDSERR] status bit) 0 Interrupt disabled. 1 Enables the STSR1[WDSERR] status bit to generate an interrupt.
SERR	Error fault injection control. Setting this bit causes the STSR1[ERR*n*] status bits to be set.
MSKWDTERR	Interrupt enable (STSR1[WDTERR] status bit) 0 Interrupt disabled. 1 Enables the STSR1[WDTERR] status bit to generate an interrupt.
MSKST_EOC	Interrupt Enable bit for STSR2[ST_EOC] 0 Interrupt disabled. 1 If IMR[MSKEOC] = 1, enables the STSR1[ST_EOC] status bit to generate an interrupt indication.
MSKWDG_EOA_C	Interrupt enable (STSR1[WDG_EOA_C] status bit) 0 Interrupt disabled. 1 Enables the STSR1[WDG_EOA_C] status bit to generate an interrupt.
MSKWDG_EOA_RC	Interrupt enable (STSR1[WDG_EOA_RC] status bit) 0 Interrupt disabled. 1 Enables the STSR1[WDG_EOA_RC] status bit to generate an interrupt.
MSKWDG_EOA_S	Interrupt enable (STSR1[WDG_EOA_S] status bit) 0 Interrupt disabled. 1 Enables the STSR1[WDG_EOA_S] status bit to generate an interrupt.
MSKERR_C	Interrupt enable (STSR1[ERR_C] status bit) 0 Interrupt disabled. 1 Enables the STSR1[ERR_C] status bit to generate an interrupt.
MSKERR_RC	Interrupt enable (STSR1[ERR_RC] status bit) 0 Interrupt disabled. 1 Enables the STSR1[ERR_RC] status bit to generate an interrupt.
MSKERR_S2	Interrupt enable (STSR1[ERR_S2] status bit) 0 Interrupt disabled. 1 Enables the STSR1[ERR_S2] status bit to generate an interrupt.
MSKERR_S1	Interrupt enable (STSR1[ERR_S1] status bit) 0 Interrupt disabled. 1 Enables the STSR1[ERR_S1] status bit to generate an interrupt.
MSKERR_S0	Interrupt enable (STSR1[ERR_S0] status bit) 0 Interrupt disabled. 1 Enables the STSR1[ERR_S0] status bit to generate an interrupt.
EN	Self-test channel enable bit. This bit enables the test channel only in CPU mode. In CTU trigger/control mode the enable is provided directly by CTU. This bit should be set before starting the normal conversion and should not be changed while conversion is ongoing. This bit should be cleared only after end of conversion for the last self-test channel has been received. 0 Test conversions are disabled. 1 Test conversions are enabled.
FMA_WDSERR	Fault mapping for the Watchdog Sequence error. 0 Non-critical fault (NCF) mapping. 1 Critical fault (CF) mapping.

Table 11-29. STCR2 field descriptions (continued)

Field	Description
FMA_WDTERR	Fault mapping for the Watchdog Timer error. 0 NCF mapping. 1 CF mapping.
FMA_C	Fault mapping for Algorithm C. 0 NCF mapping. 1 CF mapping.
FMA_RC	Fault mapping for Algorithm RC. 0 NCF mapping. 1 CF mapping.
FMA_S	Fault mapping for Algorithm S. 0 NCF mapping. 1 CF mapping.

11.3.1.25 Self-Test Configuration Register 3 (STCR3)

Address: Base + 0x348 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ALG		0	0	0	MSTEP				
W																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

Figure 11-29. Self-Test Configuration Register 3 (STCR3)

Table 11-30. STCR3 field descriptions

Field	Description
ALG	Algorithm scheduling. This field has different functionality depending on the ADC mode (One-Shot or Scan). **For One-Shot mode:** 00 Algorithm S (single step = MSTEP). 01 Algorithm RC (single step = MSTEP). 10 Algorithm C (single step = MSTEP). 11 Algorithm S. For test/debug purposes. **For Scan mode:** 00 Algorithm S. 01 Algorithm RC. 10 Algorithm C. 11 Algorithm S + Algorithm RC + Algorithm C. The baud rate for the execution of the selected algorithm is defined by the STBRR register.

Table 11-30. STCR3 field descriptions (continued)

Field	Description
MSTEP	For One-Shot mode, this field defines the current step for Algorithms S, RC, and C as follows: • For Algorithm S: MSTEP = 0 to 2 • For Algorithm C: MSTEP = 0 to 16 • For Algorithm RC: MSTEP = 0 to 18 For Scan mode, this field is not used and should be programmed to 0x00. This is because in Scan mode, only single step execution (interleaved mode) is performed.

11.3.1.26 Self-Test Baud Rate Register (STBRR)

Address: Base + 0x34C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	WDT		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	BR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-30. Self-Test Baud Rate Register (STBRR)

Table 11-31. STBRR field descriptions

Field	Description
WDT	Watchdog timer value. This value is used to monitor that the algorithm sequence is correctly executed within the respective process safety time or fault tolerant time interval. The self-testing watchdog is enabled by setting the STAW*n*R[WDTE] control bits. A fixed pre-scaler runs on the ADC clock (120 MHz). 000 0.1 ms 001 0.5 ms 010 1 ms 011 2 ms 100 5 ms 101 10 ms 110 20 ms 111 50 ms
BR	Baud rate for the selected algorithm in Scan mode (MCR[MODE] = 1). This field must be programmed before enabling the self-test channel. 0x00 Maximum scheduling rate (nominal rate) . . . 0xFF Minimum scheduling rate (nominal rate scaled by 255)

11.3.1.27 Self-Test Status Register 1 (STSR1)

Address: Base + 0x350 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	WDSERR	0	WDTERR	OVERWR	ST_EOC	0	0	0	0	WDG_EOA_C	WDG_EOA_RC	WDG_EOA_S
W					w1c		w1c	w1c	w1c					w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR_C	ERR_RC	ERR_S2	ERR_S1	ERR_S0	0	STEP_C					STEP_RC				
W	w1c	w1c	w1c	w1c	w1c											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-31. Self-Test Status Register 1 (STSR1)

Table 11-32. STSR1 field descriptions

Field	Description
WDSERR	Watchdog sequence error of the ADC sub-system (check for algorithm step sequence). It generates an interrupt if enabled (STCR2[MSKWDSERR] = 1). It provides the fault indication to the FCCU, asserting CF or NCF according to the STCR2[FMA_WDSERR] mapping. 0 No failure. 1 Failure occurred.
WDTERR	Watchdog timer error of the ADC sub-system (algorithm check for completion within respective process safety time or fault tolerant time interval). It generates an interrupt if enabled (STCR2[MSKWDTERR] = 1). It provides the fault indication to the FCCU, asserting CF or NCF according to the STCR2[FMA_WDTERR] mapping. 0 No failure. 1 Failure occurred.
OVERWR	Overwrite error. Used to notify when the STSR1[ERR*n*] bit is overwritten by a newer one. The new error status is written or discarded according to the MCR[OWREN] bit value. To avoid OVERWR indication, the ERR*n* status bit must be cleared (via software).
ST_EOC	Self-Test EOC Bit. If IMR[MSKEOC] = 1, this bit is set along with EOC bit when end_of_conversion signal is received from ADC analog for self-test channel. It generates an interrupt if enabled by STCR2[MSKST_EOC].
WDG_EOA_C	This bit indicates that Algorithm C has been completed. This bit is set after the last step of the algorithm is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_C] = 1). This bit is set only if STAW4R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.

Table 11-32. STSR1 field descriptions (continued)

Field	Description
WDG_EOA_RC	This bit indicates that Algorithm RC has been completed. This bit is set after the last step of the algorithm is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_RC] = 1). This bit is set only if STAW3R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.
WDG_EOA_S	This bit indicates that Algorithm S has been completed. This bit is set after the last step of Algorithm S is executed. It generates an interrupt if enabled (STCR2[MSKWDG_EOA_S] = 1). This bit is set only if STAW0R[WDTE] = 1. For CTU conversions, this bit is significant only for Burst mode of operation.
ERR_C	Indicates an error on the self-testing channel (Algorithm C). It generates an interrupt if enabled (STCR2[MSKERR_C] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]) You can also set the ERR_C bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No Algorithm C error has occurred. 1 An Algorithm C error has occurred.
ERR_RC	Indicates an error on the self-testing channel (algorithm RC). It generates an interrupt if enabled (STCR2[MSKERR_RC] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]). You can also set the ERR_RC bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No Algorithm RC error has occurred. 1 An Algorithm RC error has occurred.
ERR_S2	Indicates an error on the self-testing channel (algorithm SUPPLY, Step2). It generates an interrupt if enabled (STCR2[MSKERR_S2] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]). You can also set the ERR_S2 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No error has occurred on the sampled signal. 1 An error has occurred on the sampled signal.
ERR_S1	Indicates an error on the self-testing channel (algorithm SUPPLY, Step1). It generates an interrupt if enabled (STCR2[MSKERR_S1] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]). You can also set the ERR_S1 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No VDD error has occurred. 1 A VDD error has occurred.
ERR_S0	Indicates an error on the self-testing channel (algorithm SUPPLY, Step0). It generates an interrupt if enabled (STCR2[MSKERR_S0] = 1). It provides the fault indication to the FCCU, asserting the programmed fault line (STCR2[FMA*n*]). You can also set the ERR_S0 bit (fault injection) by setting the STCR2[SERR] bit. In this case the CF or NCF lines are asserted according to the STCR2[FMA*n*] mapping. 0 No VREF error has occurred. 1 A VREF error has occurred.
STEP_C	Step of Algorithm C when an ERR_C has occurred. 0.. (NUM_C_STEPS – 1) → Algorithm C.
STEP_RC	Step of Algorithm RC when an ERR_RC has occurred. 0.. (NUM_RC_STEPS – 1) → Algorithm RC.

11.3.1.28 Self-Test Status Register 2 (STSR2)

Address: Base + 0x354 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVFL	0	0	0	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-32. Self-Test Status Register 2 (STSR2)

Table 11-33. STSR2 field descriptions

Field	Description
OVFL	Overflow bit. This bit is set when the divisor is zero. If this happens, the STSR1[ERR_S1] bit is also set.
DATA1	Test channel converted data when the ERR_S1 has occurred. — Algorithm S (step1)→ fractional part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP.
DATA0	Test channel converted data when the ERR_S1 has occurred. — Algorithm S (step1) → integer part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP.

NOTE

If the MCR[OVERWR] bit is set, the STSR2-4 registers are overwritten if another error occurs.

11.3.1.29 Self-Test Status Register 3 (STSR3)

Address: Base + 0x358 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-33. Self-Test Status Register 3 (STSR3)

Table 11-34. STSR3 field descriptions

Field	Description
DATA1	Test channel converted data when the ERR_S2 has occurred. — Algorithm S (step2) → test channel data = VREF/VREF.
DATA0	Test channel converted data when the ERR_S0 has occurred. — Algorithm S (step0) → test channel data = VBGAP/VREF.

NOTE

If the MCR[OVERWR] bit is set, the STSR2-4 registers are overwritten if another error occurs.

11.3.1.30 Self-Test Status Register 4 (STSR4)

Address: Base + 0x35C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	DATA1											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DATA0											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-34. Self-Test Status Register 4 (STSR4)

Table 11-35. STSR4 field descriptions

Field	Description
DATA1	Test channel converted data when the ERR_C has occurred. — Algorithm C → test channel data.
DATA0	Test channel converted data when the ERR_RC has occurred. — Algorithm RC → test channel data.

NOTE

If the MCR[OVERWR] bit is set, the STSR2-4 registers are overwritten if another error occurs.

11.3.1.31 Self-Test Data Register 1 (STDR1)

Address: Base + 0x370 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERWR	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	TCDATA											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-35. Self-Test Data Register 1 (STDR1)

Table 11-36. STDR1 field descriptions

Field	Description
VALID	Valid data. Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERWR	Overwrite data. Used to notify when a conversion data is overwritten by a newer result. The new data is written or discarded according to the MCR[OWREN] bit value.
TCDATA	Test channel converted data.

11.3.1.32 Self-Test Data Register 2 (STDR2)

Address: Base + 0x374 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FDATA												VALID	OVERWR	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	IDATA											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-36. Self-Test Data Register 2 (STDR2)

Table 11-37. STDR2 field descriptions

Field	Description
FDATA	Fractional part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP for Algorithm S. It takes 26 ADC clock cycles from the time the EOC bit is set until valid data can be read from this field.
VALID	Valid data. Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERWR	Overwrite data. Used to notify when a conversion data is overwritten by a newer result. The new data is written or discarded according to the MCR[OWREN] bit value.
IDATA	Integer part of the ratio TEST(step1)/TEST (step0) = VDD/VBGAP for Algorithm S. It takes 26 ADC clock cycles from the time the EOC bit is set until valid data can be read from this field.

11.3.1.33 Self-Test Analog Watchdog Register 0 (STAW0R)

Address: Base + 0x380 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWDE	WDTE	0	0	THRH											
W																
Reset	0	0	0	0	0	1	1	1	0	0	1	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	1

Figure 11-37. Self-Test Analog Watchdog Register 0 (STAW0R)

Table 11-38. STAW0R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to Algorithm S (Step0) is disabled. 1 The analog watchdog related to Algorithm S (Step0) is enabled.
WDTE	Watchdog timer enable. The watchdog timer verifies: • Correct sequence of the algorithm (step sequence) • Execution of the algorithm within the respective process safety time or fault tolerant time interval as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the respective process safety time or fault tolerant time interval. The watchdog timer is reset each time the Algorithm S (Step0) restarts. **Note:** This bit should be set only in Scan mode. 0 The watchdog timer related to Algorithm S is disabled. 1 The watchdog timer related to Algorithm S is enabled.
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_S0] status bit is set if STDR1[TCDATA] > THRH.

Table 11-38. STAW0R field descriptions (continued)

Field	Description
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_S0] status bit is set if STDR1[TCDATA] < THRL.

11.3.1.34 Self-Test Analog Watchdog Register 1A (STAW1AR)

Address: Base + 0x384 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWDE	0	0	0	THRH											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 11-38. Self-Test Analog Watchdog Register 1A (STAW1AR)

Table 11-39. STAW1AR field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to Algorithm S (Step1) is disabled. 1 The analog watchdog related to Algorithm S (Step1) is enabled.
THRH	High threshold value (integer part) for test channel for Algorithm S (Step1) (unsigned coding).
THRL	Low threshold value (integer part) for test channel for Algorithm S (Step1) (unsigned coding).

11.3.1.35 Self-Test Analog Watchdog Register 1B (STAW1BR)

Address: Base + 0x388 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	THRH											
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	1	1	1	0	1	1	0	1	0	0	0	0

Figure 11-39. Self-Test Analog Watchdog Register 1B (STAW1BR)

Table 11-40. STAW1BR field descriptions

Field	Description
THRH	High threshold value (fractional part) for test channel for Algorithm S (Step1) (unsigned coding).
THRL	Low threshold value (fractional part) for test channel for Algorithm S (Step1) (unsigned coding).

11.3.1.36 Self-Test Analog Watchdog Register 2 (STAW2R)

Address: Base + 0x38C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWDE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1

Figure 11-40. Self-Test Analog Watchdog Register 2 (STAW2R)

Table 11-41. STAW2R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to Algorithm S (Step2) is disabled. 1 The analog watchdog related to Algorithm S (Step2) is enabled.
THRL	Low threshold value for channel *n* (unsigned coding). If the analog watchdog is enabled, the STSR1[ERR_S2] status bit is set if STDR1[TCDATA] < THRL.

11.3.1.37 Self-Test Analog Watchdog Register 3 (STAW3R)

Address: Base + 0x390 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWDE	WDTE	0	0	THRH											
W																
Reset	0	0	0	0	1	0	0	0	0	1	1	0	1	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	1	1	1	1	0	0	1	0	0	1	1

Figure 11-41. Self-Test Analog Watchdog Register 3 (STAW3R)

Table 11-42. STAW3R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to Algorithm RC is disabled. 1 The analog watchdog related to Algorithm RC is enabled.
WDTE	Watchdog timer enable. The watchdog timer verifies: • Correct sequence of the algorithm (step sequence) • Execution of the algorithm within the respective process safety time or fault tolerant time interval as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the respective process safety time or fault tolerant time interval. The watchdog timer is reset each time the algorithm restarts. **Note:** This bit should be set only in Scan mode. 0 The watchdog timer related to Algorithm RC is disabled. 1 The watchdog timer related to Algorithm RC is enabled.
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_RC] status bit is set if STDR1[TCDATA] > THRH.
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_RC] status bit is set if STDR1[TCDATA] < THRL.

11.3.1.38 Self-Test Analog Watchdog Register 4 (STAW4R)

Address: Base + 0x394 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AWDE	WDTE	0	0	THRH											
W																
Reset	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0

Figure 11-42. Self-Test Analog Watchdog Register 4 (STAW4R)

Table 11-43. STAW4R field descriptions

Field	Description
AWDE	Analog watchdog enable 0 The analog watchdog related to Algorithm C is disabled. 1 The analog watchdog related to Algorithm C is enabled.

Table 11-43. STAW4R field descriptions (continued)

Field	Description
WDTE	Watchdog timer enable. The watchdog timer verifies: • Correct sequence of the algorithm (step sequence) • Execution of the algorithm within the respective process safety time or fault tolerant time interval as defined by STBRR[WDT] As soon as the watchdog timer is enabled the algorithm starting must be detected within the respective process safety time or fault tolerant time interval. The watchdog timer is reset each time the algorithm restarts. **Note:** This bit should be set only in Scan mode. 0 The watchdog timer related to Algorithm C is disabled. 1 The watchdog timer related to Algorithm C is enabled.
THRH	High threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_C] status bit is set if STDR1[TCDATA] > THRH. **Note:** For Algorithm C, this value is valid only for the Step0 (refer to the STAW5R register for the other Algorithm C steps).
THRL	Low threshold value for channel *n*. If the analog watchdog is enabled, the STSR1[ERR_C] status bit is set if STDR1[TCDATA] < THRL. **Note:** For Algorithm C, this value is valid only for the Step0 (refer to the STAW5R register for the other Algorithm C steps).

11.3.1.39 Self-Test Analog Watchdog Register 5 (STAW5R)

Address: Base + 0x398 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	THRH											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	THRL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 11-43. Self-Test Analog Watchdog Register 5 (STAW5R)

Table 11-44. STAW5R field descriptions

Field	Description
THRH	High threshold value (unsigned coding) for Algorithm C (step1 to step CS – 1). If the analog watchdog is enabled (STAW4R[AWDE] = 1), the STSR1[ERR_C] status bit is set if STDR1[TCDATA{Stepn}] – STDR1[TCDATA {algC-step0}] > THRH.
THRL	Low threshold value (unsigned coding) for Algorithm C (step1 to step CS – 1). If the analog watchdog is enabled (STAW4R[AWDE] = 1), the STSR1[ERR_C] status bit is set if STDR1[TCDATA {algC-step0}] – STDR1[TCDATA{Stepn}] > THRL.

11.4 Functional description

11.4.1 ADC channel muxing

Table 11-45 defines the channel muxing for ADC_0 and ADC_1.

Table 11-45. ADC_0 and ADC_1 channel muxing[1]

ADC		Int/ Ext	Signal source[2]
Module name	Channel number		
Shared external channels between ADC_0 and ADC_1			
ADC_0	11	EXT	adc0_adc1_AN[11][3]
ADC_1	11		
ADC_0	12	EXT	adc0_adc1_AN[12][3]
ADC_1	12		
ADC_0	13	EXT	adc0_adc1_AN[13][3]
ADC_1	13		
ADC_0	14	EXT	adc0_adc1_AN[14][3]
ADC_1	14		
ADC_0 external channels			
ADC_0	0	EXT	adc0_AN[0]
ADC_0	1	EXT	adc0_AN[1]
ADC_0	2	EXT	adc0_AN[2]
ADC_0	3	EXT	adc0_AN[3]
ADC_0	4	EXT	adc0_AN[4]
ADC_0	5	EXT	adc0_AN[5]
ADC_0	6	EXT	adc0_AN[6]
ADC_0	7	EXT	adc0_AN[7]
ADC_0	8	EXT	adc0_AN[8]
ADC_1 external channels			
ADC_1	0	EXT	adc1_AN[0]
ADC_1	1	EXT	adc1_AN[1]
ADC_1	2	EXT	adc1_AN[2]
ADC_1	3	EXT	adc1_AN[3]
ADC_1	4	EXT	adc1_AN[4]
ADC_1	5	EXT	adc1_AN[5]
ADC_1	6	EXT	adc1_AN[6]

Table 11-45. ADC_0 and ADC_1 channel muxing[1] (continued)

ADC		Int/ Ext	Signal source[2]
Module name	Channel number		
ADC_1	7	EXT	adc1_AN[7]
ADC_1	8	EXT	adc1_AN[8]
ADC_0 internal channels			
ADC_0	9	INT	PMU analog mux output
ADC_0	10	INT	VDD_LV_COR Supply
ADC_0	15	INT	TSENS
ADC_1 internal channels			
ADC_1	9	INT	PMU analog mux output
ADC_1	10	INT	VDD_HV_PDI Supply
ADC_1	15	INT	VDD_HV_DRAM Supply
ADC_0 and ADC_1 Presampling			
ADC_0	PRES0	INT	VDD_HV_ADR_0
ADC_0	PRES1	INT	VSS_HV_ADR_0
ADC_1	PRES0	INT	VDD_HV_ADR_1
ADC_1	PRES1	INT	VSS_HV_ADR_1

1 When performing conversions of the internal PMC channels, the minimum sample time set by software must be at least $t_{ADC\_S\_PMC}$. Please consult the Data Sheet for this value.

2 See Chapter 3, Signal Description.

3 Accurate results cannot be guaranteed if both ADCs are configured to simultaneously sample the same shared channel.

Table 11-46 defines the channel muxing for ADC_2 and ADC_3.

Table 11-46. ADC_2 and ADC_3 channel muxing[1]

ADC		Int/ Ext	Signal source[2]
Module name	Channel number		
Shared external channels between ADC_2 and ADC_3			
ADC_2	11	EXT	adc2_adc3_AN[11][3]
ADC_3	11		
ADC_2	12	EXT	adc2_adc3_AN[12][3]
ADC_3	12		
ADC_2	13	EXT	adc2_adc3_AN[13][3]
ADC_3	13		

Table 11-46. ADC_2 and ADC_3 channel muxing[1] (continued)

ADC		Int/ Ext	Signal source[2]
Module name	Channel number		
ADC_2	14	EXT	adc2_adc3_AN[14][3]
ADC_3	14		
ADC_2 external channels			
ADC_2	0	EXT	adc2_AN[0]
ADC_2	1	EXT	adc2_AN[1]
ADC_2	2	EXT	adc2_AN[2]
ADC_2	3	EXT	adc2_AN[3]
ADC_3 external channels			
ADC_3	0	EXT	adc3_AN[0]
ADC_3	1	EXT	adc3_AN[1]
ADC_3	2	EXT	adc3_AN[2]
ADC_3	3	EXT	adc3_AN[3]
ADC_2 internal channels			
ADC_2	9	INT	PMU analog mux output
ADC_2	10	INT	not used
ADC_2	15	INT	not used
ADC_3 internal channels			
ADC_3	9	INT	PMU analog mux output
ADC_3	10	INT	not used
ADC_3	15	INT	not used
ADC_2 and ADC_3 Presampling			
ADC_2	PRES0	INT	VDD_HV_ADR_23
ADC_2	PRES1	INT	VSS_HV_ADR_23
ADC_3	PRES0	INT	VDD_HV_ADR_23
ADC_3	PRES1	INT	VSS_HV_ADR_23

1 When performing conversions of the internal PMC channels, the minimum sample time set by software must be at least $t_{ADC\_S\_PMC}$. Please consult the Data Sheet for this value.

2 See Chapter 3, Signal Description.

3 Accurate results cannot be guaranteed if both ADCs are configured to simultaneously sample the same shared channel.

Figure 11-44 and Figure 11-45 show the ADC channel muxing. Each ADC can select one out of 16 channels for conversion. Channels 11 to 14 are shared between the two ADCs. Channel 7 and 8 of each ADC are not implemented on all MPC5675K derivatives. Refer to Table 11-45 and Table 11-46 for more

details. Channel 9, 10, and 15 are not connected to external pins but to internal voltage sources as shown in Figure 11-44, Figure 11-45, Table 11-45, and Table 11-46.

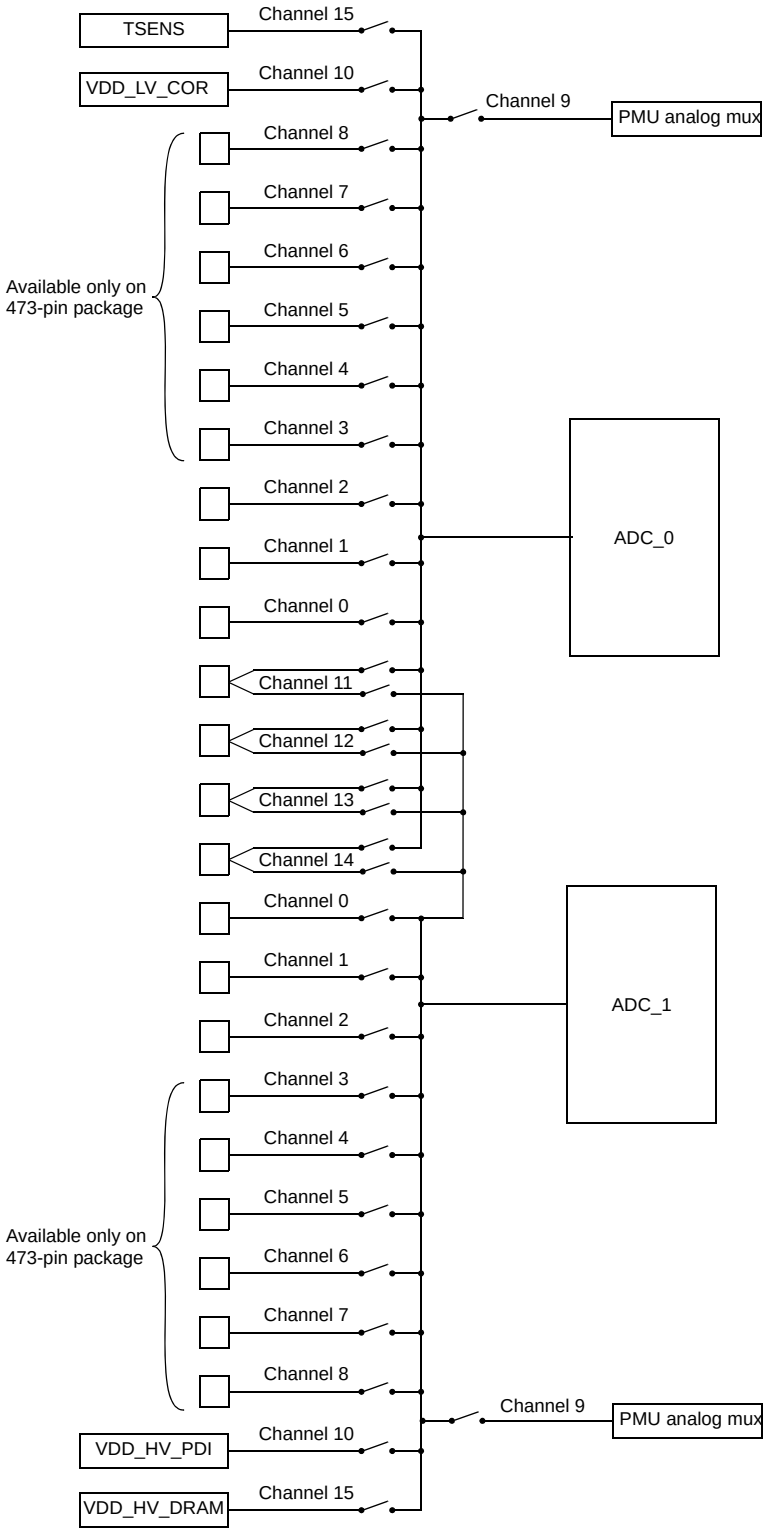


Figure 11-44. ADC_0 and ADC_1 channel muxing

As shown in Figure 11-44 and Table 11-46, ADC_2 and ADC_3 implement a slightly different number of channels compared to ADC_0 and ADC_1.

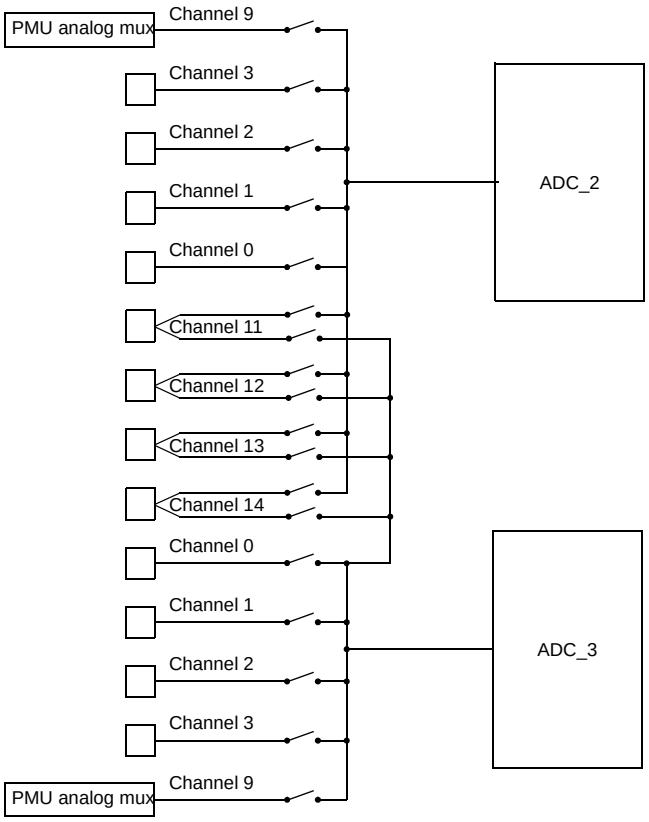


Figure 11-45. ADC_2 and ADC_3 channel muxing

11.4.2 Analog channel conversion

- 12-bit resolution
- Reference voltage from 3.3 V to 5.5 V (nominal)
- Supply voltage from 3.3 V (nominal)
- Minimum conversion time (compound of sampling + bit evaluation): See the specifications for t_{ADC_S} and t_{ADC_E} in the MPC5675K data sheet
- Minimum sampling time: See the specification for t_{ADC_S} in the MPC5675K data sheet
- Software controlled power-down

Two conversion modes are available within the ADC module:

- Normal conversion
- Injected conversion

11.4.2.1 Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCMR). Each channel can be individually enabled by setting ‘1’ in the corresponding NCMR field. Mask

registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (MSR[NSTART] is cleared).

11.4.2.2 Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One-Shot mode
- Scan mode

To enter one of these modes, the MCR[MODE] bit must be programmed. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in Figure 11-46.

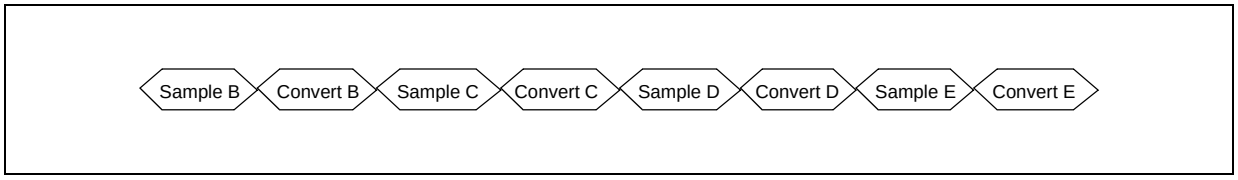


Figure 11-46. Normal conversion flow

In One-Shot mode (MCR[MODE] = 0), a sequential conversion specified in the NCMRs is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

Example 11-1. One-Shot mode (MCR[MODE] = 0)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One-Shot mode. MCR[MODE] = 0 is set for One-Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time, the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case, the new requested conversion starts after the running conversion is completed.

In Scan mode (MCR[MODE] = 1), a sequential conversion of N channels specified in the NCMR registers is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. Unlike One-Shot mode, MCR[NSTART] is not reset. It can be reset by software when the user needs to stop Scan mode. In that case, the ADC completes the current scan conversion and, after the last conversion, also resets the MCR[NSTART] bit.

Example 11-2. Scan mode (MCR[MODE] = 1)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan mode. MCR[MODE] = 1 is set for Scan mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning

of channel B starts followed by conversion of the channels D-E. This sequence repeats itself until the MCR[NSTART] bit is reset by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit).

If the content of all the normal conversion mask registers is zero (that is, no channel is selected), the conversion operation is considered completed and the ECH interrupt (see Section 11.4.7, Interrupts) is issued immediately after the start of conversion.

11.4.2.3 Injected channel conversion

A conversion chain can be injected into the ongoing normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As normal conversion, each channel can be individually selected. This injected conversion can only occur in One-Shot mode and interrupts the normal conversion. When an injected conversion is inserted, ongoing channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was stopped as shown in Figure 11-47.

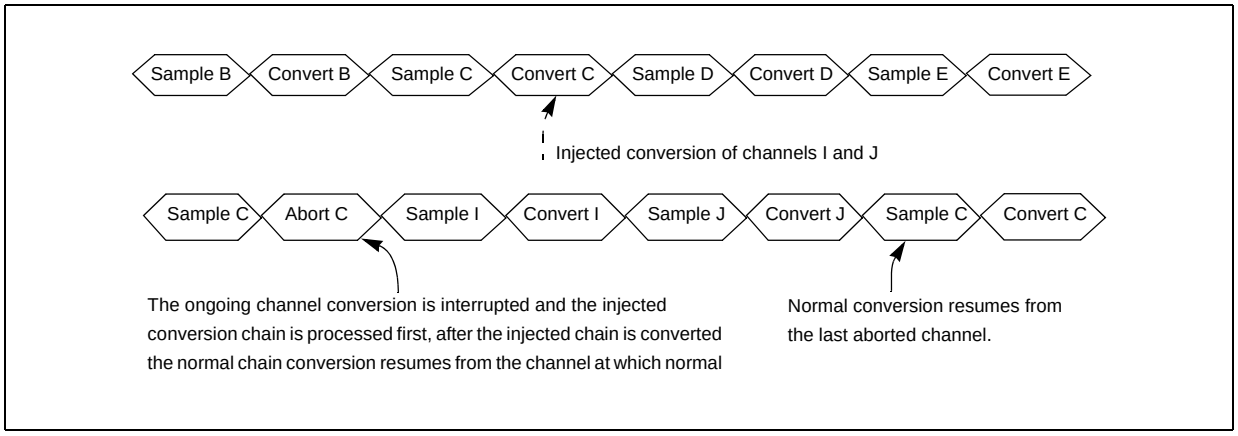


Figure 11-47. Injected sample/conversion sequence

The MSR[JSTART] status bit is automatically set when the injected conversion starts. At the same time the MCR[JSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the corresponding mask bit) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the corresponding mask bit).

If the content of all the injected conversion mask registers is zero (that is, no channel is selected) the interrupt JECH is immediately issued after the start of conversion.

Once started, injected chain conversion cannot be interrupted by any other conversion type (it can, however, be aborted; see Section 11.4.2.4, Abort conversion).

NOTE

When triggered conversions interrupt the ADC, it is possible that the aborted conversion does not get restored to the ADC and is not converted during the chain. Vulnerable configurations are:

* Injected chain over a normal chain

* CTU trigger over a normal chain

When any of these triggers arrive whilst the ADC is in the conversion stage of the sample and conversion, the sample is discarded and is not restored. This means that the channel data register will not show the channel as being valid and the CEOCFRx field will not indicate a pending conversion. The sample that was aborted is lost.

When the trigger arrives during the final channel in a normal or injected chain, the same failure mode can cause two ECH interrupts to be raised.

If the trigger arrives during the sampling phase of the last channel in the chain, an ECH is triggered immediately, the trigger is processed and the channel is restored and after sampling/conversion, a second ECH interrupt occurs.

In scan mode, the second ECH does not occur if the trigger arrives during the conversion phase. In one-shot mode, the trigger arriving during the conversion phase of the last channel restarts the whole conversion chain and the next ECH occurs at completion of that chain.

To manage this behavior, the application must check for valid data using the CDR status bits or the CEOCFRx registers to ensure all expected channels have converted. This can be tested by running a bitwise AND and an XOR with either the NCMRx register and the CEOCFRx registers during the ECH handler. Any non-zero value for (xCMRx & (xCMRx $\oplus$ CEOCFRx)) indicates that a channel has been missed and conversion should be requested again.

Spurious ECH interrupts can be detected by checking the NSTART flags in the ADC Module Status Registers. If the flag remains set during an ECH interrupt then another interrupt will follow after the restored channel or chain has been sampled and converted.

The spurious ECH recommended sequence above applies to single-shot conversions. In single-shot mode, NSTART changes from 1 to 0. Therefore, the user can rely on checking the NSTART bit to confirm if a spurious ECH has occurred. However, for scan mode, the NSTART bit will remain set during normal operation, so it cannot be relied upon to check for the spurious ECH issue. Consequently, if CTU is being used in trigger mode, the conversions must be single-shot and not scan mode.

11.4.2.4 Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is immediately started (generating a new start pulse to the Analog ADC). In the case of an abort operation, the NSTART/JSTART bit remains set and the MCR[ABORT] bit is reset after the conversion of the next channel starts. The EOC corresponding to the aborted channel is not generated. This behavior is true for normal or triggered/injected conversion modes. If the last channel of a chain is aborted, the end of chain is reported generating an ECH interrupt.
- It is also possible to abort the current chain conversion by setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MCR[MODE] bit. In fact, if Scan mode is disabled (MCR[MODE] is cleared), the MCR[NSTART] bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the MCR[MODE] bit is set, a new chain conversion is started. The EOC of the current aborted conversion is not generated but an ECH interrupt is generated to signal the end of the chain.

  When a chain conversion abort is requested (the MCR[ABORTCHAIN] bit is set) while an injected conversion is running over a suspended normal conversion, both injected chain and normal conversion chain are aborted (both the MCR[NSTART] and MCR[JSTART] bits are also reset).

NOTE

If an Injected chain (jch1,jch2,jch3) is injected over a Normal chain (nch1,nch2,nch3,nch4) the Abort switch does not behave as expected.

Expected behavior:

* Correct Case (without SW Abort on jch3): Nch1→ Nch2(aborted)→Jch1 → Jch2 → Jch3 →Nch2(restored) → Nch3→Nch4

* Correct Case (with SW Abort on jch3): Nch1→ Nch2(aborted)→Jch1 → Jch2 → Jch3(aborted) →Nch2(restored) → Nch3→Nch4

Observed unexpected behavior:

* Fault1 (without SW abort on jch3): Nch1→ Nch2(aborted) → Jch1 → Jch2 → Jch3 → Nch3 → Nch4 (Nch2 not restored)

* Fault2 (with SW abort on jch3): Nch1→ Nch2 (aborted)→Jch1 → Jch2 → Jch3(aborted) → Nch4 (Nch2 not restored & Nch3 conversion skipped).

To manage this behavior, it is possible to detect the unexpected behavior by using the CEOCFRx register. The CEOCFRx fields will not be set for a not restored or skipped channel, which indicates this issue has occurred. The CEOCFRx fields need to be checked before the next Normal chain execution (in scan mode). The CEOCFRx fields should be read by every ECH interrupt at the end of every chain execution.

NOTE

When ABORTCHAIN is programmed and an injected chain conversion is programmed afterwards, the injected chain is aborted, but neither JECH is set, nor ABORTCHAIN is reset. In case a CTU conversion is demanded, the CTU conversion is aborted by the ADC digital logic but the CTU awaits ADC analogue acknowledgement that never arrives, stopping all CTU operation until next reset.

When ABORT is programmed and normal/injected chain conversion comes afterwards, the ABORT bit is reset and chain conversion runs without a channel abort.

If ABORT, or ABORTCHAIN, feature is programmed after the start of the chain conversion, it works properly.

To manage this behavior, do not program ABORT/ABORTCHAIN before starting the execution of the chain conversion.

If the CTU is being used in trigger mode, there will always be a small vulnerable window preventing reliably reading the ADC status and using ADC.MCR[ABORT] or ADC.MCR[ABORTCHAIN]. If these functions are required, disable all CTU triggers to that ADC first and do not start the CTU if the ABORT or ABORTCHAIN flags have been set whilst the ADC was IDLE. If the CTU cannot be disabled, an optimised ABORT should be implemented in software that de-configures further channel conversions using the ADC.NMCRx registers before waiting for ADC.MSR[NSTART] == 0. At this point, the ADC should be IDLE and the the NMCRx registers can be reinstated for next use.

11.4.3 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL] bit. When this bit is set, the ADC clock has the same frequency as the system clock. Otherwise, the ADC clock is half of the system clock frequency. The MCR[ADCLKSEL] bit can be written only in power-down mode.

When the internal divider is not enabled (ADCCLKSEL = 1), it is important that the associated clock divider in the clock generation module is ‘1’. This is needed to ensure 50% clock duty cycle.

The direct clock should be used only in low power mode when the device is using only the 16 MHz fast internal RC oscillator, but the conversion still requires a 16 MHz clock. An 8 MHz clock is not fast enough.

In all other cases, the ADC should use the clock divided by two internally.

11.4.4 ADC sampling and conversion timing

In order to support different loading and switching times, several different Conversion Timing registers (CTR*x*) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitance to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the fields of the CTR*x* register. Fields INPLATCH, INPCMP, and INPSAMPLE are used to define the total conversion duration (T_{conv}) and in particular the partition between sampling phase duration (T_{sample}) and total evaluation phase duration (T_{eval}).

NOTE

T_{eval} is equivalent to datasheet parameter tADC_E. T_{sample} is equivalent to datasheet parameter tADC_S.

In the following equation, the unit T_{ck} refers to the reciprocal of the motor control clock which is then modified by value of the MCR.ADCCLKSEL field:

$$T_{CK} = (2 - MCR.ADCCLKSEL) * T_{MOTC\_CLK} \quad \textit{Eqn. 11-1}$$

The sampling phase duration is:

$$T_{SAMPLE} = (INSAMPLES - 1) * T_{ck} \quad \textit{Eqn. 11-2}$$

where INPSAMPLES must be greater than or equal to 8 (hardware requirement). If INPSAMPLES is < 8, the sampling time remains as if INPSAMPLES = 8.

The total evaluation phase duration is:

$$T_{eval} = 12 * T_{biteval} \quad \textit{Eqn. 11-3}$$

INPCMP	$T_{biteval}$
00	$4 * T_{ck}$
01	$1 * T_{ck}$
10	$2 * T_{ck}$
11	$3 * T_{ck}$

1. The Tsample and Teval must respect the minimum value specified in the Data Sheet to allow ADC to reach TUE performance.

2. For Tbiteval/INPCMP explanation, please refer to figure.

The total conversion duration is (T_{CONV}) is (not including external multiplexing) the time to perform the total evaluation:

$$T_{CONV} = T_{SAMPLE} + T_{EVAL} + T_{CK} + T_{DIGSYNC} + T_{CTUSYNC} \quad \textit{Eqn. 11-4}$$

where $T_{DIGSYNC}$ = (1 + **MCR.ADCCLKSEL**) * T_{CK}

MCR.CTUEN	MCR.ADCCLKSEL	$T_{CTUSYNC}$
0	-	0
1	0	0.5 * T_{ck}
1	1	2 * T_{ck}

The maximum clock frequency is specified in the MPC5675K data sheet.

Presampling allows precharging or discharging the ADC internal capacitor before it starts sampling/conversion of the analog input. This is useful for resetting information (offset cancellation) regarding the last converted data. During presampling, the analog ADC samples the internally generated voltage, while in the sampling the analog ADC samples analog input coming from pads.

Presampling can be enabled or disabled on a per-channel basis by setting the corresponding bits in the Presampling Register 0 (PSR0).

After enabling the presampling for a channel, the normal sequence of operation is Presampling+Sampling+Conversion for that channel. Sampling of the channel can be bypassed by setting the PSCR[PRECONV] bit. When sampling of a channel is bypassed, the sampled data of internal voltage in the Presampling State is converted (see Figure 11-48 and Figure 11-49).

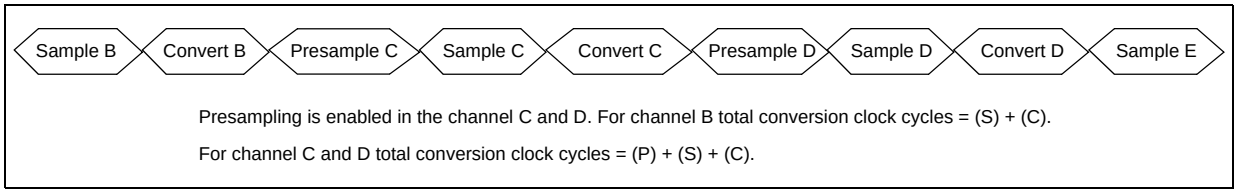


Figure 11-48. Presampling sequence

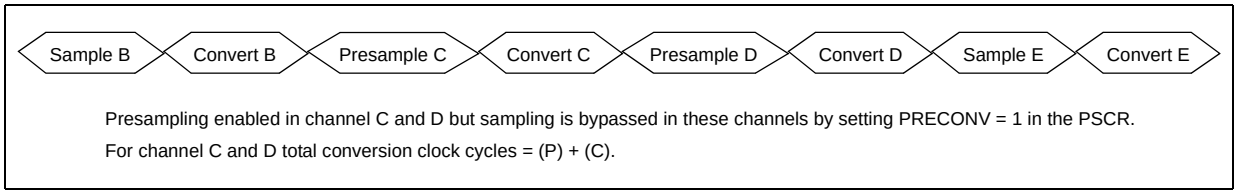


Figure 11-49. Presampling sequence with PRECONV = 1

11.4.5 Programmable analog watchdog

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in Figure 11-50) specified by an upper and a lower threshold value named THRH and THRL, respectively.

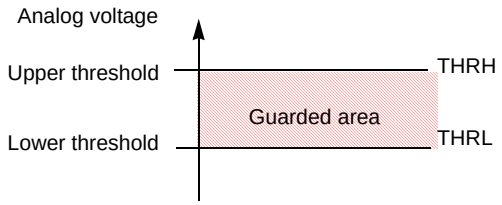


Figure 11-50. Guarded area

After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value lies outside that guarded area then corresponding threshold violation interrupts are generated. The comparison result is stored as WDG*x*H and WDG*x*L bits in the WTISR as explained in Table 11-47. Depending on the mask bits MSKWDG*x*L and MSKWDG*x*H in the WTIMR, an interrupt is generated on threshold violation.

Table 11-47. Values of WDG*x*H and WDG*x*L fields

WDG*x*H	WDG*x*L	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRL ≤ converted data ≤ THRH

NOTE

Avoid the situation where THRH < THRL. In this case, a WDG*x*H or WDG*x*L interrupt is always generated, and this could lead to misinterpretation of the watchdog interrupts.

11.4.6 Mode of operation

It is possible for the four ADCs to operate in parallel. The ADCs are able to sample and convert four physically different channels at the same time.

11.4.7 Interrupts

The ADC generates the following maskable interrupt signals:

- EOC (end of conversion) interrupt request
- ECH (end of chain) interrupt request
- JEOC (end of injected conversion) interrupt request
- JECH (end of injected chain) interrupt request
- EOCTU (end of CTU conversion) interrupt request
- WDG*x*L and WDG*x*H (watchdog threshold) interrupt requests
- REF_RANGE (reference voltage comparison) interrupt request
- Self-test interrupts

Table 11-48 shows the interrupts generated by each instantiation of the ADC module. The combined interrupts are routed to both interrupt controllers INTC_0 and INTC_1. The table applies to all MPC5675K family members.

Table 11-48. ADC interrupts

ADC Interrupt Signals	Combined Interrupts (ORed) routed to INTC	Interrupt Function
EOC	ADC_EOC	End of Conversion interrupt request
ECH		End of Chain interrupt request
JEOC		End of Injected Conversion interrupt request
JECH		End of Injected Chain interrupt request
EOCTU		End of CTU Conversion interrupt request
WDG_EOA_S		End of Algorithm S interrupt request
WDG_EOA_RC		End of Algorithm RC interrupt request
WDG_EOA_C		End of Algorithm C interrupt request
EOFFSET	ADC_ER	Error in offset cancellation interrupt request
OFFCANCOVR		Offset cancellation phase over interrupt request
REF_RANGE		REF_RANGE interrupt request
WDSERR		Watchdog Sequence error interrupt request
WDTERR		Watchdog Timer error interrupt request
ERR_S0		Error on Self-Testing Channel Algorithm Supply Step0 interrupt request
ERR_S1		Error on Self-Testing Channel Algorithm Supply Step1 interrupt request
ERR_S2		Error on Self-Testing Channel Algorithm Supply Step2 interrupt request
ERR_RC		Error on Self-Testing Channel Algorithm RC interrupt request
ERR_C		Error on Self-Testing Channel Algorithm C interrupt request
WDG*x*L (*x* = 0 to 15)	ADC_WD	Watchdog *x* Low Threshold Error interrupt request
WDG*x*H (*x* = 0 to 15)		Watchdog *x* High Threshold Error interrupt request

NOTE

If the ADC is in CTU Control Mode, only the sources WDG*x*L and WDG*x*H can generate an interrupt request.

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in Section 11.3.1.4, Channel Pending Register 0 (CEOCFR0). The Channel Pending Register 0 (CEOCFR0) and Interrupt Mask Register (IMR) registers are provided in order to check and enable the interrupt requests.

Interrupts can be individually enabled on a channel-by-channel basis by programming the Channel Interrupt Mask Register 0 (CIMR0).

Several Channel Interrupt Pending registers are also provided in order to signal which of the channels' measurement has been completed.

The analog watchdog interrupts are handled by two 32-bit registers WTISR (Watchdog Threshold Interrupt Status Register) and WTIMR (Watchdog Threshold Interrupt Mask Register) in order to check and enable the interrupt request to the EIC module. The Watchdog interrupt source sets two pending bits WDG*x*H and WDG*x*L in the WTISR for each of the four channels being monitored.

The CEOFR0 contains the interrupt pending request status. If the user wants to clear a particular interrupt event status, then writing a '1' to the corresponding status bit clears the pending interrupt flag (at this write operation all the other bits of the CEOFR0 must be maintained at '0').

End of conversion interrupts for the self-test channel are similar to the interrupts for the normal conversion channel. The EOC and ECH bits for normal conversion are set for self-test channel also. Similar to other channels, self-test channel has Channel interrupt pending bit (ST_EOC) present in STSR1 and channel mask bit (MSKST_EOC) in STCR2 (corresponding to bits for every channel in the CEOFR0 and CIMR0 registers). In addition, End of Algorithm interrupts are also present, which are handled by the STSR1 and STCR2 registers.

Error interrupts related to self-testing are handled by status bits in STSR1 and mask bits in STCR2. If an error is generated due to analog watchdog monitoring or sequence checking of algorithm or internal watchdog timer timeout, the corresponding error bit is set in STSR1. The mask bits for these error bits are present in STCR2.

11.4.8 DMA functionality

For each ADC module, a DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR register. The DMA masking bits must be programmed before starting any conversion.

The DMA transfers can be enabled using the DMAE[DMAEN] bit. When the DMAE[CLR] bit is set, the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

For more details on DMA channel assignments, please refer to Chapter 19, Enhanced Direct Memory Access (eDMA).

11.4.9 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN] bit. After releasing the reset signal, the ADC analog module is kept in power-down mode by default, so this state must be exited before starting any operation by clearing the MCR[PWDN] bit.

The power-down mode can be requested at any time by setting the PWDN bit in the MCR. If a conversion is ongoing, the ADC module cannot immediately enter the power-down mode. In fact, the ADC enters power-down mode only after completing the ongoing conversion. Otherwise, the ongoing operation should be aborted manually by resetting the NSTART bit and using the MCR[ABORTCHAIN] bit.

When the ADC enters power-down mode, the MSR[ADCSTATUS] field is set to 0b001.

After the power-down phase is completed, the process ongoing before the power-down phase must be restarted manually by setting the NSTART or the JSTART bit.

After an exit from power-down mode, the first conversion can be started after 5 µs, otherwise the result can be affected by the improper setting of the ADC analog operating point.

Clearing the PWDN bit and setting the NSTART or the JSTART bit during the same cycle is not permitted.

11.4.10 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an “auto-clock-off” feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

11.4.11 Self-testing

11.4.11.1 General operation

For safety devices used for very critical applications, it may be necessary to check at regular intervals that the ADC is functioning correctly. For this purpose, the self-testing feature has been incorporated inside the ADC.

The self-tests use analog watchdogs to verify the result of self-test conversions. The threshold of these watchdogs is saved in the test flash. Before running the self-test, you must copy these values from the test flash to the STAW*x*R registers.

Three types of self-testing algorithms have been implemented inside ADC analog.

- Supply self-test: Algorithm S. It includes the conversion of the ADC internal bandgap voltage, ADC supply voltage, and ADC reference voltage. It includes a sequence of 3 test conversions (steps). The supply test conversions must be an atomic operation (no functional conversions interleaved).
- Resistive-Capacitive self-test: Algorithm RC. It includes a sequence of 19 test conversions (steps) by setting the ADC internal resistive digital-to-analog converter (DAC).
- Capacitive self-test: Algorithm C. It includes a sequence of 17 test conversions (steps) by setting the capacitive elements comprising the sampling capacitor/ capacitive DAC.

The ADC implements an additional test channel dedicated for self-testing. It also provides signals to schedule self-testing algorithms using Configuration registers, monitors the converted data using analog watchdog registers, and flags the error to the Fault Collection and Control Unit (FCCU) in case some failure occurs in any of the algorithms.

The test channel can be activated in CPU or CTU mode as described in Table 11-49.

Table 11-49. Test channel activation

ADC mode	Test channel (ADC)	Test channel (CTU)
CPU mode (MCR[CTUEN] = 0)	Yes (One-Shot mode and Scan mode)	No
CTU control mode	No	Yes

11.4.11.2 CPU mode

In this case, the test channel works similar to normal conversion. In CPU mode, the test channel is enabled by setting STCR2[EN].

The self-testing channel conversions are carried along with the functional conversions. The sequencing of steps of the selected algorithm for the test channel depends on the operating mode of normal conversions interleaved, selected by the MCR[MODE] bit.

In One-Shot mode, if the test channel is enabled, only one step of the selected self-testing algorithm is executed at the end of the chain. The step number and algorithm to be executed is programmed in STCR3 register. So, in One-Shot mode, the sequence is as follows:

- Program NCMR0 to select channels to be converted for normal conversion.
- Program MCR[MODE] = 0 to select One-Shot mode.
- Program sampling duration values in STCR1[INPSAMP*n*] field.
- Select the self-testing algorithm in STCR3[ALG]. Default is Algorithm S.
- Enable self-testing channel by setting the STCR2[EN] bit.
- Start the normal conversion by setting the MCR[NSTART] bit.
- All normal conversions are executed as usual.
- At the end of all the normal conversions, the step number programmed in the STCR3[MSTEP] field of self-testing algorithm selected by STCR3[ALG] is executed similar to a normal functional channel.
- On receiving the End of Conversion for the test channel, the digital result is written in STDR1.TCDATA and STDR1[VALID] bit is set. Also, the ISR[EOC], ISR[ECH], are set and STSR1[ST_EOC] bits are set.
- State Machine returns to IDLE state.

Example 11-3.

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One-Shot mode. MODE=0 is set for One-Shot mode. The result is shown in Figure 11-51. Conversion for channels B-D-E are done. After channel E, the test channel conversion is done and ISR[ECH] and ISR[EOC] are set.

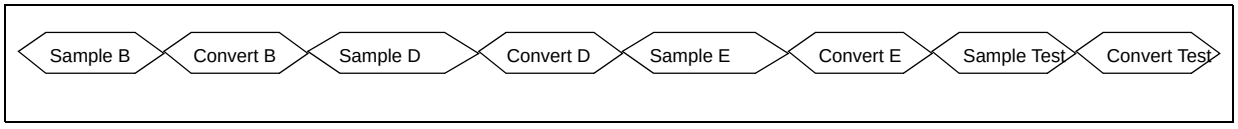


Figure 11-51. Test channel conversion example

The MSR[NSTART] status bit is automatically set when the normal conversion starts and is reset at the end of conversion for the test channel.

In Scan mode, consecutive steps of selected self-test algorithm are converted continuously at the end of each chain of normal conversions. The number of channels converted at the end of each chain is 1 (except for Algorithm S, in which all the steps are performed at once without any functional conversion interleaved). So, in Scan mode the sequence is as follows:

- Program NCMR0 to select the channels to be converted for normal conversion.
- Program MCR[MODE] = 1 to select Scan mode.
- Program sampling duration values in the STCR1[INPSAMP] field.
- Select the self-testing algorithm in STCR3[ALG]. By default, all three algorithms are selected; i.e., all algorithms are executed step-by-step, one after the other.
- Enable self-testing channel by setting the STCR2[EN] bit.
- Start the normal conversion by setting MCR[NSTART].
- All normal conversions are executed as usual.
- At the end of chain of the normal conversions, (assuming default value of STCR3[ALG]) all steps of Algorithm S are performed (as Algorithm S is always atomic). MSR[SELF_TEST_S] is set.
- On receiving end of conversion of test channel for last step of Algorithm S, the digital result is written in STDR1[TCDATA] and the STDR1[VALID] bit is set. At the same time, MSR[SELF_TEST_S] is reset. For Step1, the integral part and fractional part are written in STDR2[IDATA] and STDR2[FDATA]. Also, the ISR[EOC], ISR[ECH], and STSR2[ST_EOC] bits are set.
- Then the next chain of normal conversion starts.
- At end of the normal conversion chain, Step0 of Algorithm RC is executed.
- On the receiving end of the conversion of test channel for Step0 of Algorithm RC, the digital result is stored in STDR1[TCDATA], and the STDR1[VALID] bit is set (if the MCR[OVERWR] bit is set). Also, the ISR[EOC] and ISR[ECH] bits are set, and STSR2[ST_EOC] is set.
- Then the next chain of normal conversion starts.
- At end of the normal conversion chain, Step1 of Algorithm RC is executed.
- This process continues for all the steps of all three algorithms.
- State Machine returns to IDLE state when MCR[NSTART] is cleared.

Instead of starting normal conversion by software (by setting MCR[NSTART]), if it is started by external trigger, the test channel behavior remains the same.

In case of injected conversions, test channel conversion is not performed. It is performed only during normal conversions.

If during a test channel conversion, injection conversion arrives, then the test channel is aborted (just as a normal functional channel) and injected conversions are done. After injected conversions are completed,

the test channel resumes from the step at which it was aborted. In this case, the MSR[SELF_TEST_S] remains high during the injected conversion.

For self-testing, the Mode bit should be programmed (at least one cycle) *before* setting the MCR[NSTART] bit and should not be changed thereafter until conversion is ongoing.

11.4.11.3 CTU mode

The CTU mode is enabled by setting MCR[CTUEN]. With this bit set, the CTU operates in control mode.

If MCR[CTUEN] is set, the test channel conversion can be started only by the CTU interface and software cannot start it. The STCR2[EN] bit does not have any effect in CTU mode. The MCR[CTUEN] bit should not be changed while normal conversion is ongoing.

The interface between CTU and ADC (for self-test) is shown in Figure 11-52.

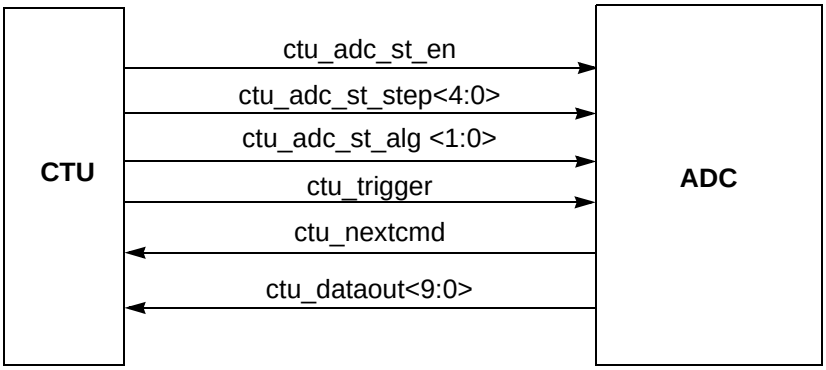


Figure 11-52. Interface between ADC and CTU to manage self-test

For self-testing conversions in CTU mode, the CTU asserts the ctu_adc_st_en signal along with the ctu_trigger signal. The algorithm and the step number to be executed are put on the ctu_adc_st_alg and ctu_adc_st_step signals, respectively. The other signals ctu_nextcmd, ctu_trigger, and ctu_dataout have the same meaning as for normal CTU functional conversions.

For Algorithm S, the three steps must be atomic. In CTU mode, this is managed by CTU itself; i.e., the CTU has to send three triggers (one for each step) asserting ctu_adc_st_en and updating ctu_adc_st_step for each step.

The Channel Conversion Command Registers (CLR*x*) in the CTU (see Chapter 18, Cross-Triggering Unit (CTU)") allow setting up a self-test command and choosing among the algorithms available. See Table 11-50 and Table 11-51.

NOTE

If the CTU trigger arrives while normal chain execution was ongoing and the current normal channel is in evaluation phase of conversion, then the current channel might not be restored back after CTU injected conversion finishes.

For example, if conversions ch0 → ch1 → ch2 → ch3 → ch4 are ongoing and the CTU trigger arrives at ch2 (evaluation phase), then ch2 would be aborted and would not be restored back after the injected CTU channel conversion is over; converted data of ch2 is lost for the current executing chain.

Table 11-50. CTU–CLR*x*-ST1, ST0 meaning

ST1 bit	ST0 bit	Command meaning
0	0	No self-test Single conversion
0	1	Self-test command
1	0	No self-test command Dual conversion
1	1	No self-test Dual conversion

Table 11-51. CTU–CLR*x*-ALG[1:0] meaning

ALG[1:0]	Algorithm meaning
00	Algorithm S
01	Algorithm RC
10	Algorithm C
11	Algorithm FULL (S + RC + C)

Algorithm S must be executed as an atomic test without interleaved user conversions.

- Step0: VBGAP/VREF test
- Step1: VDD/VBGAP test
- Step2: VREF/VREF test

The CTU sends three triggers, one for each step, asserting ctu_adc_st_en and updating ctu_adc_st_step for each step.

Algorithms RC and C can be executed, programming the CLR*x* registers, in one of the following schemes:

- Burst mode: when the CTU schedules the execution of the CLR*x* register configured for the self-testing, both the Algorithms RC and C are executed in burst mode (step0—algRC, step1—algRC, ...stepN—algRC, step0—algC, step—-algC, ... stepM—algC).
- Interleaved mode: when the CTU schedules the execution of the CLR*x* register configured for the self-testing, a single step of the Algorithm RC or C is executed according to an internal counter. In this mode, the ADC self-testing procedure is distributed and the functional conversions are not stalled for a long time.

The burst mode and interleaved mode are not applicable to Algorithm S, which can be executed only as an atomic conversion.

When Algorithm FULL is enabled, the CTU executes Algorithm S followed by Algorithm RC and Algorithm C.

The enabling of the trigger configured for the ADC self-testing can be performed according to the following schemes:

- Triggered mode: according to the current CTU implementation
- Sequential mode: according to the current CTU implementation

11.4.11.4 Abort and abort chain for self-testing channel

Setting MCR[ABORT] during self-test channel has no effect.

In One-Shot Mode, if MCR[ABORTCHAIN] is set when test channel conversion is ongoing, the test channel is aborted and ECH is set. In this case, EOC for the test channel is not generated.

For zero baud rate in Scan mode, if MCR[ABORTCHAIN] bit is set when the test channel STEP N is ongoing, the test channel STEP N is aborted and next chain conversion starts. At the end of this chain, STEP N conversion is performed again. (In case of Algorithm S, the FULL Algorithm is executed again).

The case of non-zero baud rate is described in Section 11.4.11.7.1, Abort chain when baud rate is non-zero.

11.4.11.5 Self-test analog watchdog

The ADC also provides a monitor (watchdog) for the values returned by its analog portion for self-test algorithms. The analog watchdogs are used to determine whether the result of conversion for self-test algorithms lie in a particular guard area. For this purpose, separate self-test analog watchdog registers have been provided for each algorithm.

After the conversion of each step of an algorithm, a comparison is performed between the converted value and the threshold values if Analog watchdog feature is enabled by setting the STAW*x*R[AWDE] bit. If the converted value does not lie between the upper and lower threshold values specified by the Analog Watchdog Register of the particular algorithm, the corresponding error bit STSR1[ERR_*x*] is set and the step number in which the error occurred is updated in the STSR1[STEP_*x*] bit (in case of Algorithm C or Algorithm RC). Also, erroneous data is written in the STSR4[DATA*x*] field. The STSR1[ERR_*x*] bits generate an interrupt (if enabled by corresponding Mask bit in the STCR2 register). The fault indication is also given to the FCCU via CF and NCF, so that necessary action can be taken at the device level. For more information, please see Chapter 25, Fault Collection and Control Unit (FCCU).

The Analog Watchdog feature works differently in case of Algorithm S. As already mentioned, Algorithm S is always an atomic operation. So, separate error bits are provided in STSR1 for each step of Algorithm S to avoid overwrite in case error occurs in more than one step. Hence, there are separate Mask bits for each step in STCR1. For the same reason, separate fields exist in the Status registers (STSR2 and STSR3) to store erroneous data for each step.

For Algorithm S Step1, a fixed-point division has to be performed outside the ADC (it takes around 26 ADC clock cycles after EOC for Step1 to get the divided value) and it is the divided value on which the analog watchdog checks are applied. So, the value to be compared for Step1 contains integer as well as fractional parts. Thus, two registers (STAW1AR and STAW1BR) are provided. The comparison is done first for integer part using the threshold values programmed in STAW1AR. If integer part lies outside the

range, the fractional part comparison is skipped. Otherwise, it is compared with the values programmed in STAW1BR. Table 11-52 summarizes this feature for Step1.

Table 11-52. Algorithm S (Step1) threshold comparison

STDR2[IDATA] (integer part)	STDR2[FDATA] (fractional part)	STSR1[ERR_S1]
> STAW1AR[THRH]	Any value	Set
< STAW1AR[THRL]	Any value	Set
== STAW1AR[THRH]	> STAW1BR[THRH]	Set
== STAW1AR[THRL]	< STAW1BR[THRL]	Set

For Algorithm S Step2, (VREF/VREF) is measured in order to check the integrity of sampling signal. For this particular conversion, no higher threshold value is required as the ideal value is 0xFFF. Only lower threshold value is programmed in STAW2R.

For Algorithm C, a separate register is provided for Step0. In Step0, an offset for other steps is measured. The converted data is compared with the threshold values provided by STAW5R if STAW4R[AWDE] is set. For other steps, this offset is subtracted from converted data before performing watchdog checks.

11.4.11.6 Watchdog timer

The watchdog timer is an additional check that monitors the sequence of the self-testing algorithm implemented and also that the algorithm is completed within a respective process safety time or fault tolerant time interval. The watchdog timers can be enabled for CPU as well CTU conversions. Each algorithm has a different watchdog timer that runs independently of the other. The watchdog timer for a particular algorithm can be enabled by setting the STAW*n*R[WDTE] bit. The respective process safety time or fault tolerant time interval can be programmed in STBRR[WDT] field (the default value is 10 ms, assuming a 120 MHz clock).

The respective process safety time or fault tolerant time interval is measured starting from Step0 of the algorithm (including all normal chain conversions in between) to the point where Step0 of the same algorithm starts again.

The sequence is as follows:

- Program NCMR0 to select channels to be converted for normal conversion in Scan mode (MODE = 1).
- Program the STCR1 with the required INPSAMP/INPCMP/INLATCH durations for the desired test conversion algorithms.
- Select the self-testing algorithm in STCR3[ALG]. By default, all three algorithms are selected (i.e., all algorithms are executed step by step, one after the other). For scan mode the default is Algorithm (S+RC+C) and for one shot mode the default is Algorithm S. Also, program STCR3[MSTEP] in case of One-Shot mode.
- Enable the required interrupts from one of many possible interrupt events by writing STCR2 register (events are related to watchdog sequence error, time error, error bits for individual steps errors, etc.) based on the diagnostic requirement of the chosen algorithm.

- Enable self-testing channel by setting the STCR2[EN] bit.
- Program respective process safety time or fault tolerant time interval in the STBRR[WDT] field.

NOTE

This value is dependent on the ADC_clk frequency and total number of interleaved conversion steps, including normal conversion steps.

- Program the STAW*x*R registers bits (except the WDTE bit) to associate a watchdog (WDG) timer to a particular algorithm to monitor correct sequence (order) of algorithm steps and/or respective process safety time or fault tolerant time interval of the chosen algorithm and threshold of the converted value at every step. For this purpose, a number of WDG timers are available and are programmable through 7 STAW*x*R registers (*x*:0,3,4,1A,1B,2,5).
- Enable watchdog timer by setting STAW*x*R[WDTE] bit. Assume the setting of WDTE bit to be ‘t0’. (It is important to do all the programming first, and then enable WDTE bit as the respective process safety time or fault tolerant time interval check is also performed between setting of WDTE bit and start of Step0 to check that the algorithm has started within the respective process safety time or fault tolerant time interval).
- Freeze the above setting writing MCR[STCL] (config lock). Start the normal conversion by setting MCR[NSTART].
- After first chain conversion ends, STEP0 of Algorithm S is executed. Lets assume the start of STEP0 to be ‘t1’.
- After this STEP1 and STEP2 of Algorithm S are executed.
- Then, next chain conversions are performed.
- When chain conversion completes, STEP0 of Algorithm RC is performed.
- After each chain conversion, consecutive STEP of Algorithm RC is performed. A similar sequence follows for Algorithm C.
- After the last step of Algorithm C is performed, another chain conversion is executed. At the end of this chain conversion, STEP0 of Algorithm S is started repeating the whole sequence. Assume this time (starting of STEP0) to be ‘t2’.
- For Algorithm S, if (t1 – t0) > Safe Period or (t2 – t1) > Safe Period, the watchdog timer flags an error and the STSR1[WDTERR] bit is set. Critical fault is asserted and an interrupt is also generated if enabled by the STCR2[.MSKWDTERR] bit. Otherwise, watchdog timer counter is reset and starts again to monitor the same for the next sequence.
- A similar sequence is followed for watchdog timers for Algorithm RC and Algorithm C.
- To stop the above algorithm(s):
  - Disable the watchdog timer for all algorithms (STAW*x*R[WDTE] bits).
  - One-Shot mode: algorithm automatically stops on every step unless the next step is initiated setting MCR[NSTART].
  - Scan mode: clearing of MCR[START] stops the scan mode conversion and thus stops the test conversion (algorithm).
- Check the converted data in STDR1–2 and error conditions in STSR1–4.

As the CTU does not incorporate any safe period checking mechanism, the watchdog timers can be enabled for CTU conversions also.

NOTE

You must not enable the watchdog timer for an algorithm that is not to be executed.

11.4.11.6.1 Watchdog sequence checking

The watchdog timer also incorporates sequence checking features that check that the steps of a particular algorithm are in the correct order. If steps are not in order, then an error is flagged by setting STSR1[WDSERR]. If enabled by STCR2[MSKWDSERR], Critical fault is asserted and an interrupt is also generated.

Watchdog sequence error is flagged in following cases:

- If steps of any algorithm are not executed in proper order.
- If abort chain occurs during test channel conversion, that step has to be repeated at the end of the next chain. This gives a sequence error as soon as test channel conversion starts again.

  Exception: If abort chain occurs during last step of Algorithm S, then sequence error is not flagged as the whole algorithm has to be repeated again.
- If, for CTU conversions, step numbers provided by CTU are not in order. Watchdog sequence checking is significant for CTU burst mode only.

If injected conversion occurs during the test channel, the watchdog sequence error is NOT flagged although the ongoing step number is aborted and is repeated again.

NOTE

The watchdog timer feature is applicable only for Scan mode, not for One-Shot mode. The STBRR[BR] should be set to zero for One-Shot mode.

11.4.11.7 Baud rate control for test channel

It defines the scheduling of test channel between the normal conversions. The scheduling rate is specified by the STBRR[BR] field.

By default, if the test channel is enabled, one step of the selected algorithm is executed after every chain of normal conversion. The bandwidth consumed by the test channel depends on the number of channels in normal chain. For example, if we have 100 normal conversions in a chain, then test channel consumes only 1% of total bandwidth, which is very small. But in case the number decreases to just 4 channels, then the bandwidth consumed by the test channel is 25%, which is significant (and may not be desirable as it slows down the normal conversion rate).

The STBRR[BR] field provides flexibility by scheduling the test channel conversion to be performed not at the end of every chain but at the end of STBRR[BR] + 1 number of chains. For example, if STBRR[BR] = 5, a single step of selected algorithm for the test channel is performed after 6 chain conversions, then the next step is performed at end of the next 6 chain conversions, and so on. By default, the value of STBRR[BR] is 0.

To use the baud rate feature in Scan mode, the NCMR should have a non-zero value.

NOTE

This feature is applicable only for Scan mode, not for One-Shot mode. The STBRR[BR] should be set to zero for One-Shot mode.

11.4.11.7.1 Abort chain when baud rate is non-zero

As already described, for a non-zero value of the STBRR[BR] field, the test channel conversion is performed at the end of (BR + 1) number of chains. If an abort chain occurs during the chain in which the test channel is scheduled to be converted, then the test channel is converted after the next (BR + 1) number of chains.

For example, if the STBRR[BR] field is programmed to 0x2, the sequence is two normal chains (without any test channel conversion), followed by a chain with the test channel converted at the end. Now, if an abort chain occurs during the first two chains, it is treated as a normal chain abort and the test channel is converted at the end of the 3rd chain only (as the case without any abort chain). But if an abort chain occurs during the 3rd chain, in which the test channel is scheduled to be converted, then the test channel is converted after the next three chains, i.e., at the end of the 6th chain (counting from the beginning).

Chapter 12
Boot Assist Module (BAM)

12.1 Overview

The BAM is a block of read-only memory containing VLE code that is executed according to the boot mode of the device.

The BAM downloads code into internal SRAM through the following serial protocols and executes it afterwards:

- FlexCAN (with and without autobaud) — see Chapter 27, FlexCAN Module
- LINFlexD-UART (with and without autobaud) — see Chapter 34, LIN Controller (LINFlexD)

12.2 Features

The BAM serial boot provides the following features:

- Puts BAM in Safe mode if internal flash is not initialized or invalid
- Programmable 64-bit password protection for serial boot mode
- Serial boot loads the application boot code from a FlexCAN or LINFlex bus into internal SRAM
- Censorship protection for the internal flash module

NOTE

The BAM initializes the SRAM used for BAM operation and also the area in which the code is downloaded with 64-bit data so that no ECC errors are reported. The rest of the SRAM area remains uninitialized, and the application code must initialize the rest of the SRAM if it to be used.

12.3 Safety aspects

The MPC5675K device accesses the BAM module only in a safe environment condition.

As a consequence of this, all the fault management mechanisms (NMI and interrupts generated by the FCCU) are ignored during BAM code execution whenever possible. The software timer (SWT), which is enabled on reset, is also disabled to prevent any resets.

The aim of the BAM code is to download code from a serial interface and place it in SRAM. At the end of the download, a checksum is performed to enhance the integrity of the downloaded data.

12.4 Boot modes

The MPC5675K supports the following boot modes:

- Single Chip (SC)—the device boots from the first bootable section of the flash main array.
- Serial Boot (SBL)—the device downloads boot code from either the FlexCAN or LINFlex interface and then executes it.

If booting is not possible with the selected configuration (for example, if no valid Boot ID is found in any of the possible boot locations) then the device enters the static mode. “Static mode” is defined as entering the “LP SAFE” mode by programming the MC_ME accordingly and executing the Power Architecture “wait” instruction.

12.5 Memory map

The BAM code resides in 8 KB of ROM mapped from address 0xFFFF_C000. The address space and memory used by the BAM is shown in Table 12-1.

Table 12-1. BAM memory organization

Entity	Address
BAM entry point	0xFFFF_C000
Downloaded code base address	0x4000_0100. This address is valid only for GPIO-zero-data mode (JTAG). All other modes use a protocol that embeds the destination address.

The RAM location from which to download the code can be any 8-byte-aligned location in the SRAM, starting from the address 0x4000_0100.

12.6 Functional description

12.6.1 Entering boot modes

The MPC5675K detects the boot mode based on external pins and device status. The following sequence applies (see Figure 12-1):

1. To boot from the FlexCAN or LINFlex, the device must be forced into an Alternate Boot Loader Mode via the FAB (Force Alternate Boot Mode) pin, which must be asserted before initiating the reset sequence. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see Table 12-2).
2. If FAB is not asserted, the device boots from the first flash-memory sector that contains a valid boot signature.
3. If no flash memory sector contains a valid boot signature, the device goes into static mode.

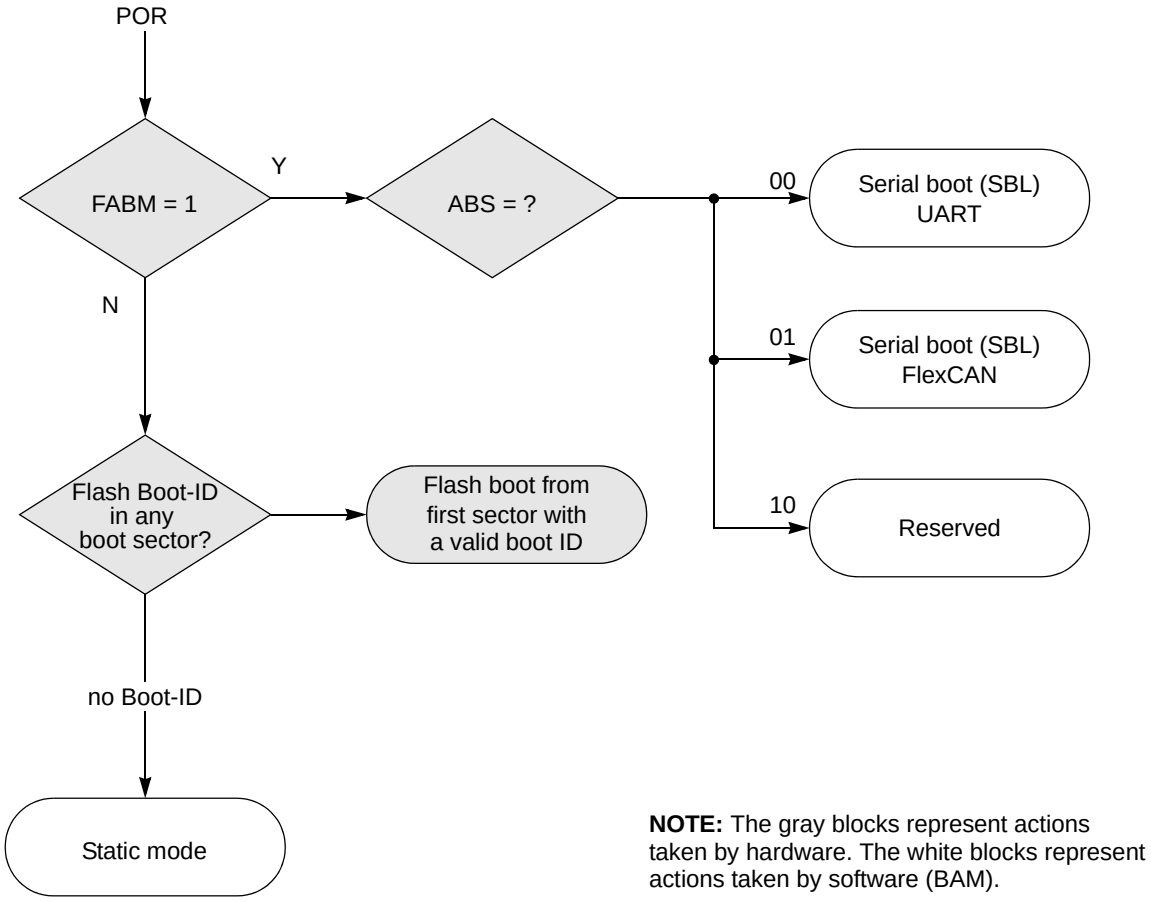


Figure 12-1. Boot mode selection

Table 12-2. Hardware configuration to select boot mode

FAB	ABS[2]	ABS[0]	Autobaud	BAM Action
0	x	x	—	No BAM code is executed. Normal boot from internal flash controlled by SSCM.
1	0	0	No	Serial boot via LINFlex_0.
1	0	1	No	Serial boot via FlexCAN_0.
1	1	0	Yes	Serial boot via LINFlex_0 or FlexCAN_0 with autobaud support for each peripheral. The serial boot loading is executed by the peripheral answering to a boot load request.
1	1	1	—	Reserved. BAM places the device in Safe Mode.

12.6.2 Reset Configuration Halfword Source (RCHW)

The MPC5675K flash memory is partitioned into boot sectors as shown in Table 12-4. Each boot sector contains the Reset Configuration Halfword (RCHW) at offset 0x00.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	VLE	BOOT_ID							
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 12-2. Reset Configuration Halfword (RCHW)

Table 12-3. RCHW field descriptions

Field	Description
VLE	Selects the VLE or Book E instruction set. 0 Selects the classic PowerPC instruction set. 1 Selects the VLE instruction set.
BOOT_ID	Valid boot identifier is 0x5A.

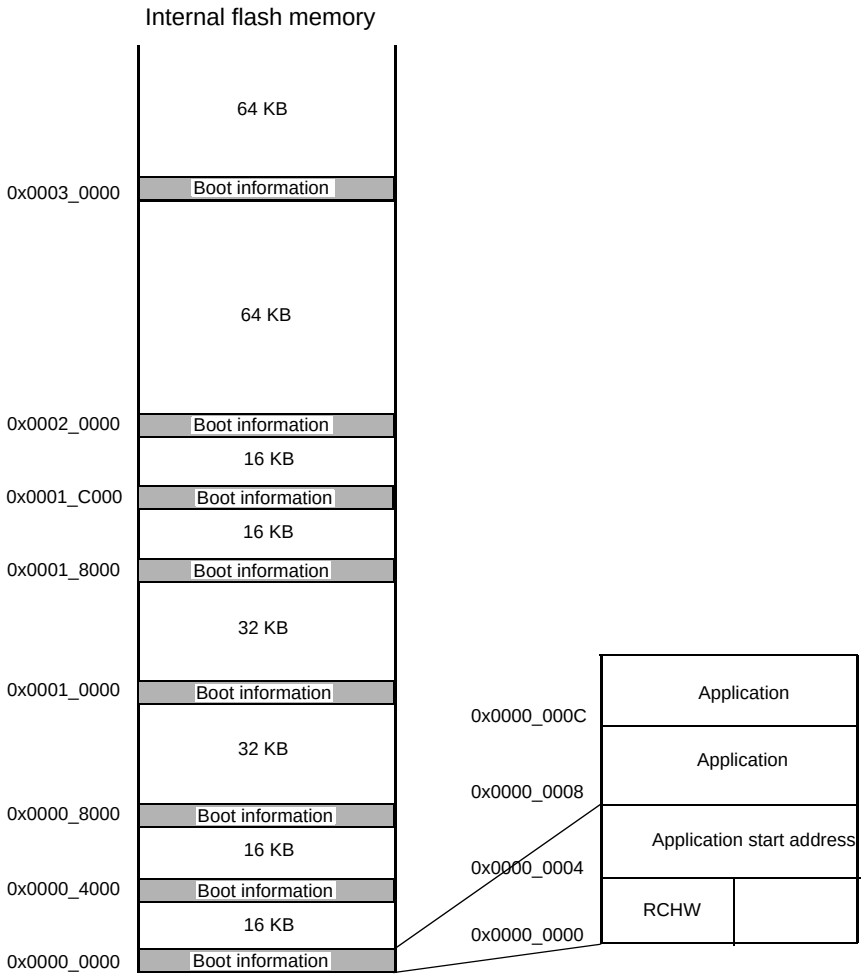


Figure 12-3. MPC5675K flash memory partitioning and RCHW search

Table 12-4. Flash memory boot sector locations

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0000_8000
3	0x0001_0000
4	0x0001_8000
5	0x0001_C000
6	0x0002_0000
7	0x0003_0000

12.6.3 Single chip boot mode

In single chip mode, the hardware searches all valid flash boot sector locations for a valid boot ID. As soon the device detects a bootable sector, it jumps within this sector and reads the 32-bit word at offset 0x4. This word is the address where the startup code is located (reset boot vector).

Then the device executes this startup code. The user application must have a valid instruction at the reset boot vector address.

If a valid RCHW is not found, the BAM code is executed. In this case, the BAM moves the MPC5675K into static mode.

12.6.3.1 Boot and alternate boot

Some applications require an alternate boot sector so that the main boot can be erased and reprogrammed in the field.

When an alternate boot is needed, the user can create two bootable sectors. The lowest sector is the main boot sector, and the highest is the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This scheme ensures that there is always one active boot sector by erasing one of the boot sectors only:

- Sector is activated (that is, program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector is deactivated writing to 0 some of the bits BOOT-ID bit field (bit 1 and/or bit 3, and/or bit 4, and/or bit 6).

12.6.4 Boot through BAM

12.6.4.1 Executing BAM

Single chip boot mode is managed by hardware and BAM does not participate in it.

BAM is executed only in the following cases:

- Serial boot mode has been selected by FABM pin
- Hardware has not found a valid Boot ID in any flash memory boot location

If one of these conditions is true, the device fetches code at location 0xFFFF_C000 and the BAM application starts.

12.6.4.2 BAM software flow

Figure 12-4 describes the BAM logic flow.

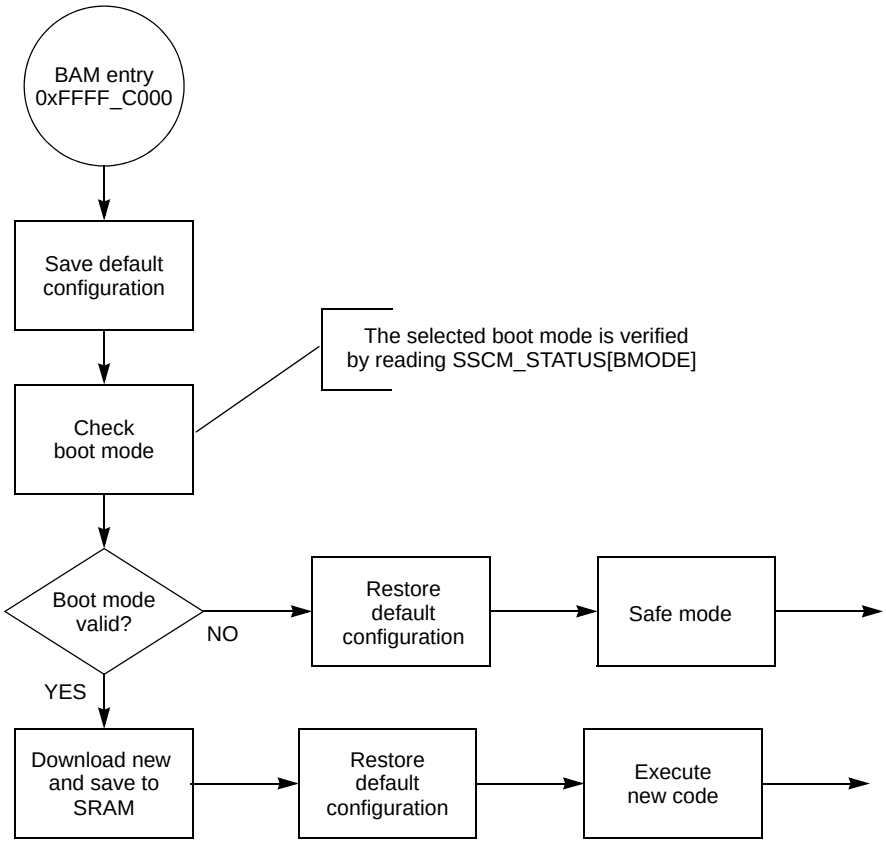


Figure 12-4. BAM logic flow

The first action is to save the initial device configuration. In this way is possible to restore the initial configuration after downloading the new code but before executing it. This allows the new code being executed as the device is just coming out of reset.

The SSCM_STATUS[BMODE] field (see Section 49.3.1.1, System Status Register (SSCM_STATUS)) indicates which boot is to be executed (see Table 12-5).

If the BMODE bitfield shows either a single chip value (011) or the reserved values, the boot mode is not considered valid and the BAM pushes the device into Safe mode.

In all other cases, the code of the relative boot is called. Data is downloaded and saved into the correct SRAM location.

Table 12-5. Fields of SSCM_STATUS register used by BAM

Field	Description
BMODE	Device Boot Mode. 000 Reserved 001 FlexCAN Serial Boot Loader 010 LINFlexD Serial Boot Loader 011 Single Chip Other values are reserved.

Then, the initial device configuration is restored and the code jumps to the address of downloaded code. At this point BAM has just finished its task.

If any error is detected, such as a communication error, wrong boot selected, etc., BAM restores the default configuration and puts the device into static mode. Static mode means the device enters the low power Safe mode and the processor executes a wait instruction. It is needed if the device cannot boot in the selected mode. During BAM execution and after, the mode reported by the field S_CURRENT_MODE of the register ME_GS in the module MC_ME Module is “DRUN.”

12.6.4.3 BAM resources

BAM uses/initializes the following MCU resources:

- MC_ME and MC_CGM modules to initialize mode and clock sources
- CAN_A, LINFlexD_A, and their pads when performing serial boot mode
- SWT is disabled in case of a serial boot mode or when entering static mode
- SSCM to check the boot mode and during password check (see Table 12-5 and Figure 12-5)
- External oscillator

The following hardware resources are used only when autobaud feature is selected:

- System Timer Module (STM) to measure the baud rate
- Clock Monitor Unit (CMU) to measure the external clock frequency related to the internal RC clock source
- FMPLL to work with the system clock near the maximum allowed frequency (for higher resolution during baud rate measurement)

The initial configuration is restored before executing the downloaded code.

When the autobaud feature is disabled, the system clock is selected directly from the external oscillator. Thus the oscillator frequency defines baud rates for serial interfaces used to download the user application (see Table 12-6).

For FlexCAN, the system clock is the FMPLL clock with IRC being the input to the FMPLL clock. The FMPLL is configured to be 96 MHz (with 16 MHz IRC). Because of the divide-by-2 divider on system clock, the FlexCAN module clock is > 40 MHz. The protocol clock of FlexCAN is driven from XOSC.

For LINFlexD, the FMPLL is selected as the system clock. The FMPLL is configured based on the XOSC. The CMU detects the XOSC frequency and whether the FMPLL is programmed accordingly.

Table 12-6. Serial boot mode without autobaud—Baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/sec)
f_{extal}	(f_{extal} × 2) / 833	f_{extal} / 40
8	19200	200K
12	28800	300K
16	38400	400K

Table 12-6. Serial boot mode without autobaud—Baud rates (continued)

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/sec)
20	48000	500K
24	57600	600 K
32	76800	800 K
40	96000	1M

12.6.4.4 Download and execute the new code

From a high-level perspective, the download protocol follows steps:

1. Send message and receive acknowledge message for autobaud or autobit rate selection. (This step is optional for FlexCAN, but required for LINFlexD.)
2. Send 64-bit password.
3. Send start address, size of downloaded code in bytes, and VLE bit.
4. Download data.
5. Execute code from start address.

Each step must be complete before the next step starts.

These steps are correct if autobaud is disabled. Otherwise, to measure the baud rate some data from host to MCU are sent before the step 1 (see Section 12.6.7, Autobaud feature).

The communication is done in half duplex manner. Any transmission from the host is followed by the MCU transmission:

1. Host sends data to MCU and starts waiting
2. MCU echoes to host the data received
3. MCU verifies if echoes are correct
   - If data is correct, the host can continue to send data
   - If data is not correct, the host stops to transmit and MCU need to be reset

All multi-byte data structures are sent MSB first.

A more detailed description of these steps follows.

12.6.4.5 Download 64-bit password and password check

The first 64 bits received represent the password. This password is sent to the Password Check procedure, which verifies if it is correct.

Password check data flow is shown in Figure 12-5 where:

- SSCM_STATUS[SEC] = 1 means that the flash memory is secured
- SSCM_STATUS[PUB] = 1 means flash memory with public access

In case of flash with public access, the received password is compared with the public password 0xFEED_FACE_CAFE_BEEF.

If public access is not allowed but the flash is not secured, the received password is compared with the value saved on NVPWD0 and NVPWD1 registers.

In both of previous cases, comparison is done by BAM application. If it goes wrong, BAM pushes MCU into static mode.

In case of public access not allowed and flash secured, the password is written into SSCM_PWCMPH/L registers.

After a fixed time waiting, comparison is done by hardware. Then BAM re-verifies the SSCM_STATUS[SEC] flag:

- SEC = 0, flash is now unsecured and BAM continues its task
- SEC = 1, flash is still secured because password was wrong; BAM puts MCU to standby mode.

This fixed time depends on external crystal oscillator frequency (XOSC). With XOSC of 12 MHz, fixed time is 350 ms.

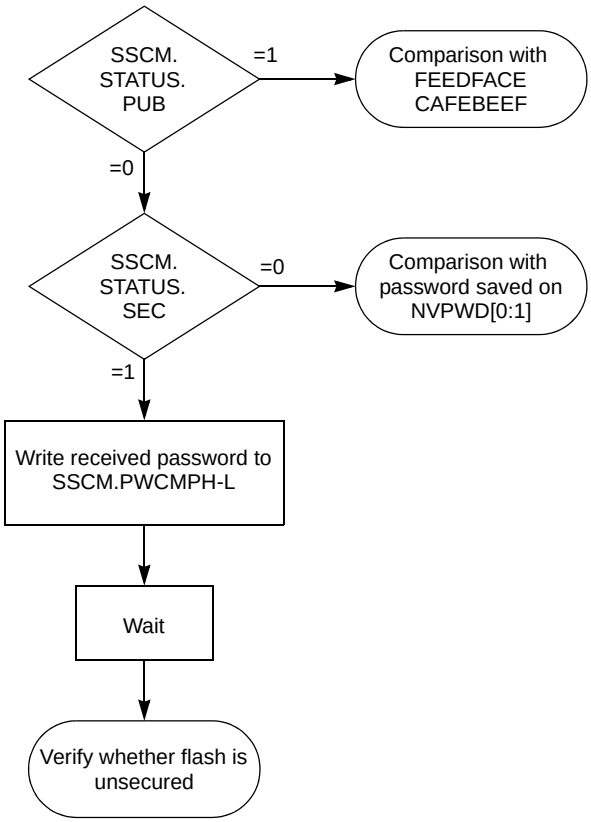


Figure 12-5. Password check flow

12.6.4.6 Download start address, VLE bit, and code size

The next 8 bytes received by the MCU contain a 32-bit start address, the VLE mode bit, and a 31-bit code length as shown in Figure 12-6.

The VLE bit (Variable Length Instruction) indicates instruction set for which the code has been compiled. Currently the BAM supports only VLE = 1. This bit is provided for backward compatibility.

The start address defines where the received data is stored and where the MCU branches after the download is finished. The two LSB bits of the start address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

The length defines how many data bytes are to be loaded.

NOTE

The maximum code size that can be provided starting from 0x4000_0100 is the SRAM memory size – 0x100 (for the BAM stack) – 0x10 (to prevent ECC error during pre-fetching). If code size is greater than this size, the BAM writes outside the SRAM memory, which causes an exception. Code execution will fail.

Observe that some members of the MPC5675K family have SRAM modules smaller than 512 KB.

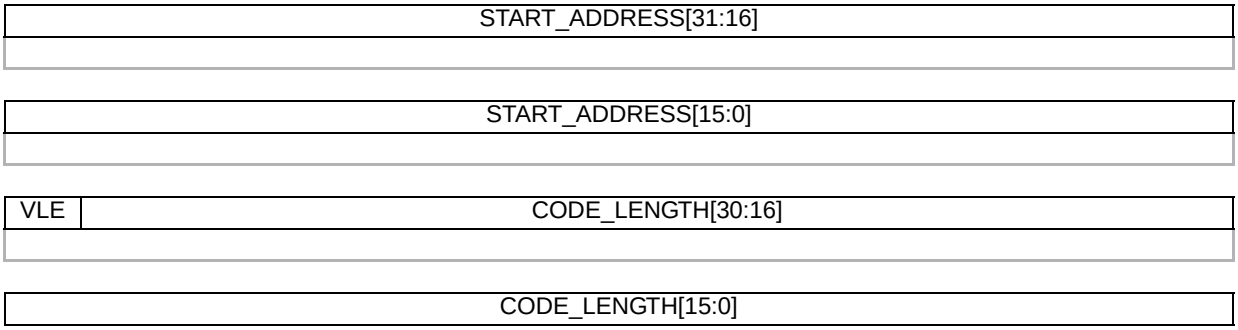


Figure 12-6. Start address, VLE bit, and download size in bytes

12.6.4.7 Download data

Each byte of data received is stored into device's SRAM, starting from the address specified in the previous protocol step.

The address increments until the number of bytes of data received matches the number of bytes specified in the previous protocol step.

The SRAM is protected by 64-bit wide Error Correction Code (ECC). Once the BAM has the start location and code size information, it makes 64-bit writes on the area so that when downloaded data is written, no ECC errors occur. BAM always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, the BAM fills it with 0 bytes.

Then a "dummy" word (0x0000_0000) is written to avoid ECC error during core prefetch.

As noted in the previous section, the maximum size of the data that can be downloaded is smaller than the size of the SRAM. In addition, since the SRAM performs 64-bit reads, it may be forced to write two

"dummy" words instead of just one, to account for the reads occurring on matching bitwise boundaries between the BAM and the SRAM.

12.6.4.8 Execute code

The BAM program waits for the completion of the last echo message transmission.

Then it restores the initial MCU configuration and jumps to the loaded code at Start Address, which was received in step 2 of the protocol.

At this point, the BAM has finished its tasks and the MCU is controlled by new code executing from SRAM.

12.6.5 Boot from UART—autobaud disabled

12.6.5.1 Configuration

Boot from UART protocol is implemented by LINFlexD_0 module. The following pins are used:

- LINFlex_TX corresponds to PCR[18]
- LINFlex_RX corresponds to PCR[19]

When the autobaud feature is disabled, the system clock is driven by the FMPLL clock.

The LINFlexD controller is configured to operate at a baud rate = (system clock frequency × 2) / 833 (see Table 12-6 for baud rate example), using 8-bit data frame without parity bit and 1 stop bit.

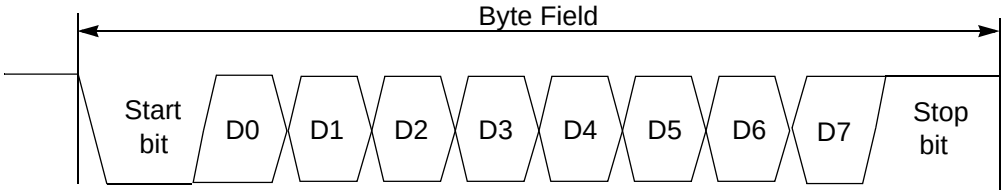


Figure 12-7. LINFlex bit timing in UART mode

12.6.5.2 Protocol

Table 12-7 summarizes the protocol and BAM action during this boot mode.

Table 12-7. UART boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	64-bit password (MSB first)	64-bit password	Password checked for validity and compared against stored password.
2	32-bit store address	32-bit store address	Load address is stored for future use.
3	VLE bit + 31-bit number of bytes (MSB first)	VLE bit + 31-bit number of bytes (MSB first)	Size of download are stored for future use. Verify if VLE bit is set to 1.

Table 12-7. UART boot mode download protocol (autobaud disabled) (continued)

Protocol step	Host sent message	BAM response message	Action
4	8 bits of raw binary data	8 bits of raw binary data	8 bits of data are packed into 32-bit words. This word is saved into SRAM starting from the “Load address”. “Load address” incremented until the number of data received and stored matches the size as specified in the previous step.
5	none	none	Branch to downloaded code.

12.6.6 Bootstrap with FlexCAN—autobaud disabled

12.6.6.1 Configuration

Boot from FlexCAN protocol is implemented by the FlexCAN_0 module. The following pins are used:

- CAN_TX
- CAN_RX

When the autobaud feature is disabled, the system clock is driven by the FMPLL clock, with IRC being the input to the FMPLL and the FlexCAN protocol clock sourced by the external oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency / 40 (see Table 12-6 for examples of baud rate).

It uses the standard 11-bit identifier format detailed in the FlexCAN 2.0A specification.

FlexCAN controller bit timing is programmed with 10 time quantas, and the sample point is 2 time quantas before the end, as shown in Figure 12-8.

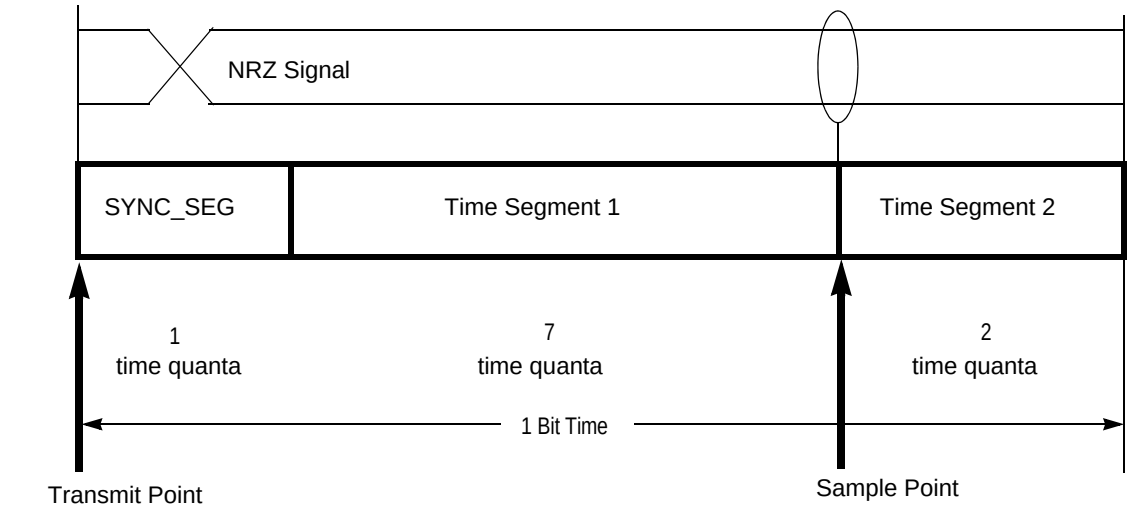


Figure 12-8. FlexCAN bit timing

12.6.6.2 Protocol

Table 12-8 summarizes the protocol and BAM action during this boot mode. All data are transmitted byte wise.

Table 12-8. FlexCAN boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	FlexCAN ID 0x011 + 64-bit password	FlexCAN ID 0x001 + 64-bit password	Password checked for validity and compared against stored password.
2	FlexCAN ID 0x012 + 32-bit store address + VLE bit + 31-bit number of bytes	FlexCAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address is stored for future use. Size of download are stored for future use. Verify if VLE bit is set to 1
3	FlexCAN ID 0x013 + 8 to 64 bits of raw binary data	FlexCAN ID 0x003 + 8 to 64 bits of raw binary data	8 bits of data are packed into 32-bit words. These words are saved into SRAM starting from the “Load address.” “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
4	none	none	Branch to dowloaded code

12.6.7 Autobaud feature

The purpose of Autobaud is to allow boot operation with a wide range of baud rates independently from the external oscillator frequency.

12.6.7.1 Configuration

The baud rate is measured by the BAM using the STM, which is driven by system clock. One of the main differences with the previous cases is that the system clock is not driven directly by the external oscillator, but it is driven by FMPLL output. The reason is that to have an optimum baud rate measurement resolution, the system clock needs to be nearer to the maximum allowed device frequency. This is achieved with the following two steps:

1. Using the Clock Monitor Unit (CMU) and the RC oscillator, the external frequency is roughly measured.
2. Using the FMPLL, the system clock is configured to be near to, but lower than the maximum allowed frequency.

The relation between system clock frequency and external clock frequency with FMPLL configuration value is shown on Table 12-9.

Table 12-9. System clock frequency related to external clock frequency

f_{osc} [MHz]	PHI Clock	f_{sys} [MHz]
8	32	16
12	48	24
16	64	32
20	80	40
24	96	48
32	128	64
40	160	80

12.6.7.2 Boot from UART with autobaud enabled

The only difference between booting from UART with autobaud enabled and booting from UART with autobaud disabled is that a further byte is sent from the host to the MCU when autobaud is enabled. The value of that byte is 0x00.

This first byte measures the time from falling edge and rising edge. The baud rate can be calculated from this time.

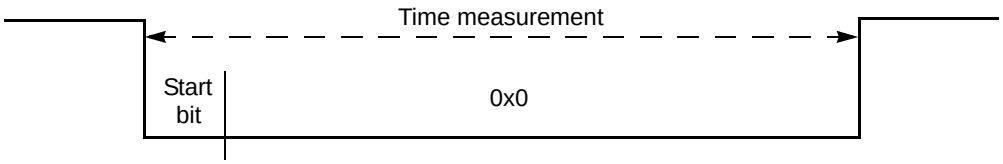


Figure 12-9. Baud measurement on UART boot

Initially the UART RX pin is configured as a GPIO input. It waits in polling for the first falling edge, when the STM starts. The UART RX pin waits again for the first rising. At this point, the STM stops and the baud rate is computed from this measurement.

The LINFlexD module is configured to work in UART mode with the calculated baud rate. Then an acknowledge byte (0x59, ASCII char “Y”) is sent.

From this point, the BAM follows the normal UART mode boot protocol (see Figure 12-10).

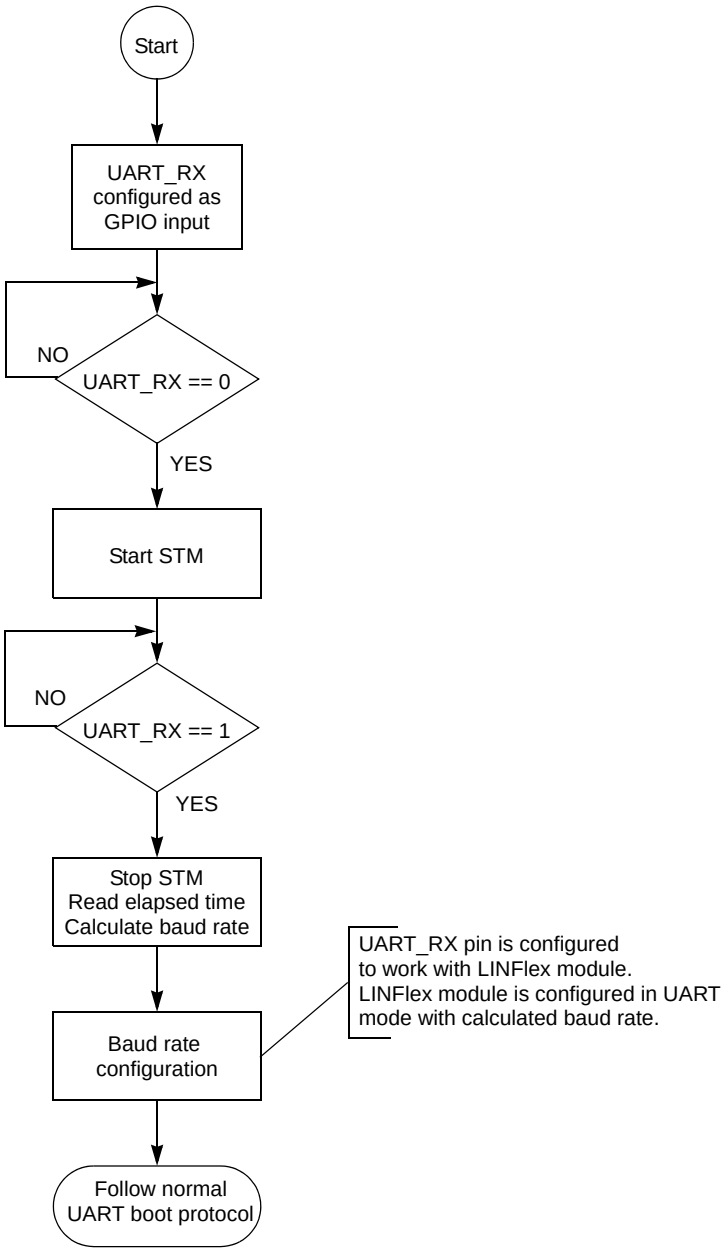


Figure 12-10. BAM rate measurement flow during UART boot

12.6.7.2.1 Choosing the host baud rate

The calculation of the UART baud rate from the length of the first zero byte that is received allows the operation of the boot loader with a wide range of baud rates. However, in order to ensure proper data transfer, the upper and lower limits have to be kept.

The LINFlexD autobaud rate feature operates by polling the LINFlexD_RX pin for a low signal. The length of time until the next low to high transition is measured using the STM time base. This

high-low-high transition is expected to be a zero byte: a start bit (low) followed by eight data bits (low) followed by a stop bit (high).

Upon reception of a zero byte and configuration of the baud rate, an acknowledge byte is returned to the host using the selected baud rate.

The time base is enabled at reception of first low bit, disabled and read at reception of next high bit. Error introduced due to polling will be small (typically <6 cycles).

The following equation gives the relation between baud rate and LINFlex register configuration:

Eqn. 12-1

$$LDIV = \frac{F_{cpu}}{16 \cdot baudrate}$$

LDIV is an unsigned fixed point number and its mantissa is coded into 13 bits of the LINIBRR register in the LINFlexD.

From this equation and considering that a single UART transmission contains 9 bits, it is possible to obtain the connection between time base measured by STM and LINIBB register:

Eqn. 12-2

$$LINIBRR = \frac{timebase}{144}$$

To minimize errors in baud rate, a large external oscillator frequency value and low baud rate signal are preferred.

Example 12-1. Baud rate calculation—24 Kbaud signal

Consider a 24 Kbaud signal and the device operating with 20 MHz external frequency.

— Over 9 bits the STM will measure: (9 × 20e6)/24e3 = 7497 cycles.

— Error expected to be approximately ±6 cycles due to polling.

— Thus, LINIBB is set to 52 (rounding required). This results in a baud rate of 24.038 Kbaud. Error of <0.2%.

To maintain the maximum deviation between host and calculated baud rate, recommendations for the user are listed Table 12-10.

Table 12-10. Maximum and minimum recommended baud rates

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for < 2.5% deviation	Min baud rate for < 2.5% deviation
4	13.4 Kbit/s (SBR = 19)	30 bit/s (SBR = 8192)
8	26.9 Kbit/s (SBR = 19)	60 bit/s (SBR = 8192)
12	38.4 Kbit/s (SBR = 20)	90 bit/s (SBR = 8192)

Table 12-10. Maximum and minimum recommended baud rates (continued)

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for < 2.5% deviation	Min baud rate for < 2.5% deviation
16	51.2 Kbit/s (SBR = 20)	120 bit/s (SBR = 8192)
20	64.0 Kbit/s (SBR = 20)	150 bit/s (SBR = 8192)

Higher baud rates high may be used, but software must ensure they fall within the acceptable error range. Figure 12-11 shows the effect of quantization error on the baud rate selection.

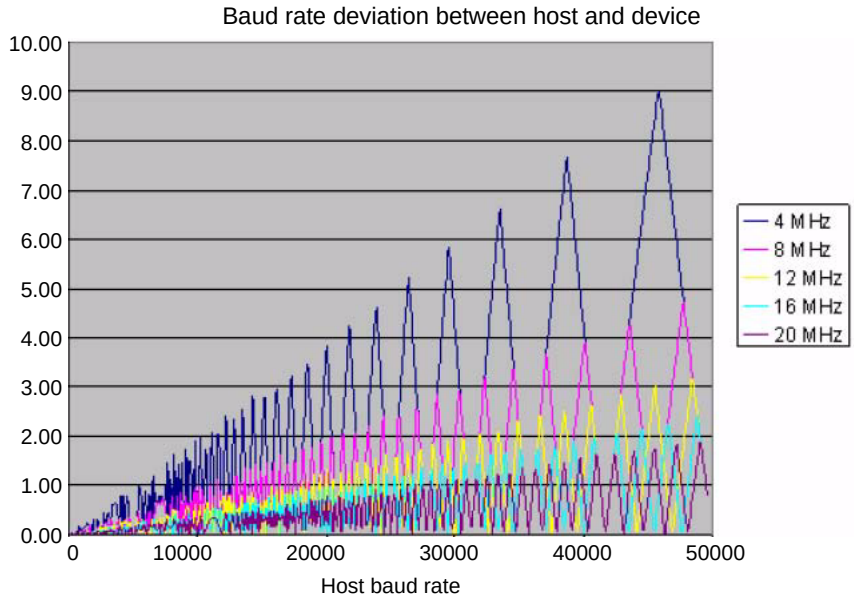


Figure 12-11. Baud rate deviation between host and MPC5675K

Example 12-2. Baud rate calculation—250 Kbaud signal

Consider the reception of a 250 Kbaud signal from the host and MPC5675K operating with a 4 MHz oscillator. Over 9 bits the time base measures: (9 × 4e6)/250e3 = 144 cycles.

— Thus, LINIBB is set to 144/144 = 1. This results in a baud rate of exactly 250 Kbaud.

However, a slower 225 Kbaud signal operating with 4 MHz XTAL would again result in LINIBB = 1, but this time with an 11.1% deviation.

12.6.7.3 Boot from FlexCAN with autobaud enabled

The only difference with the FlexCAN protocol used when autobaud is disabled is that the host sends to the MCU the following initialization FlexCAN frame for baud measurement purposes:

- Standard identifier = 0x0
- Data Length Code (DLC) = 0x0

As all the bits to be transmitted are dominant bits, there is a succession of 5 dominant bits and 1 stuff bit on the FlexCAN network (see Figure 12-12).

From the duration of this frame, the MCU calculates the corresponding baud rate factor with respect to the current CPU clock and initializes the FlexCAN interface accordingly.

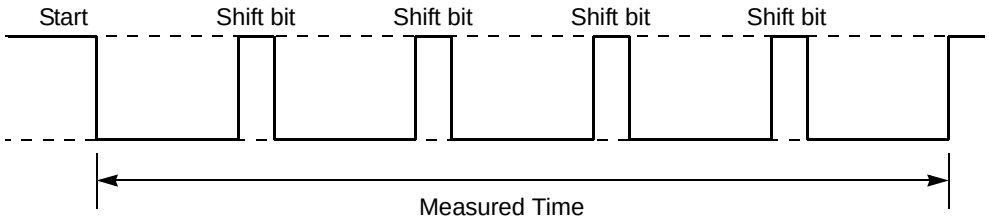


Figure 12-12. Bit time measure

As UART boot mode, the FlexCAN RX pin is first configured to work as a GPIO input. In the end of baud rate measurement, it switches to work with FlexCAN module.

The baud rate measurement flow is shown in Figure 12-13.

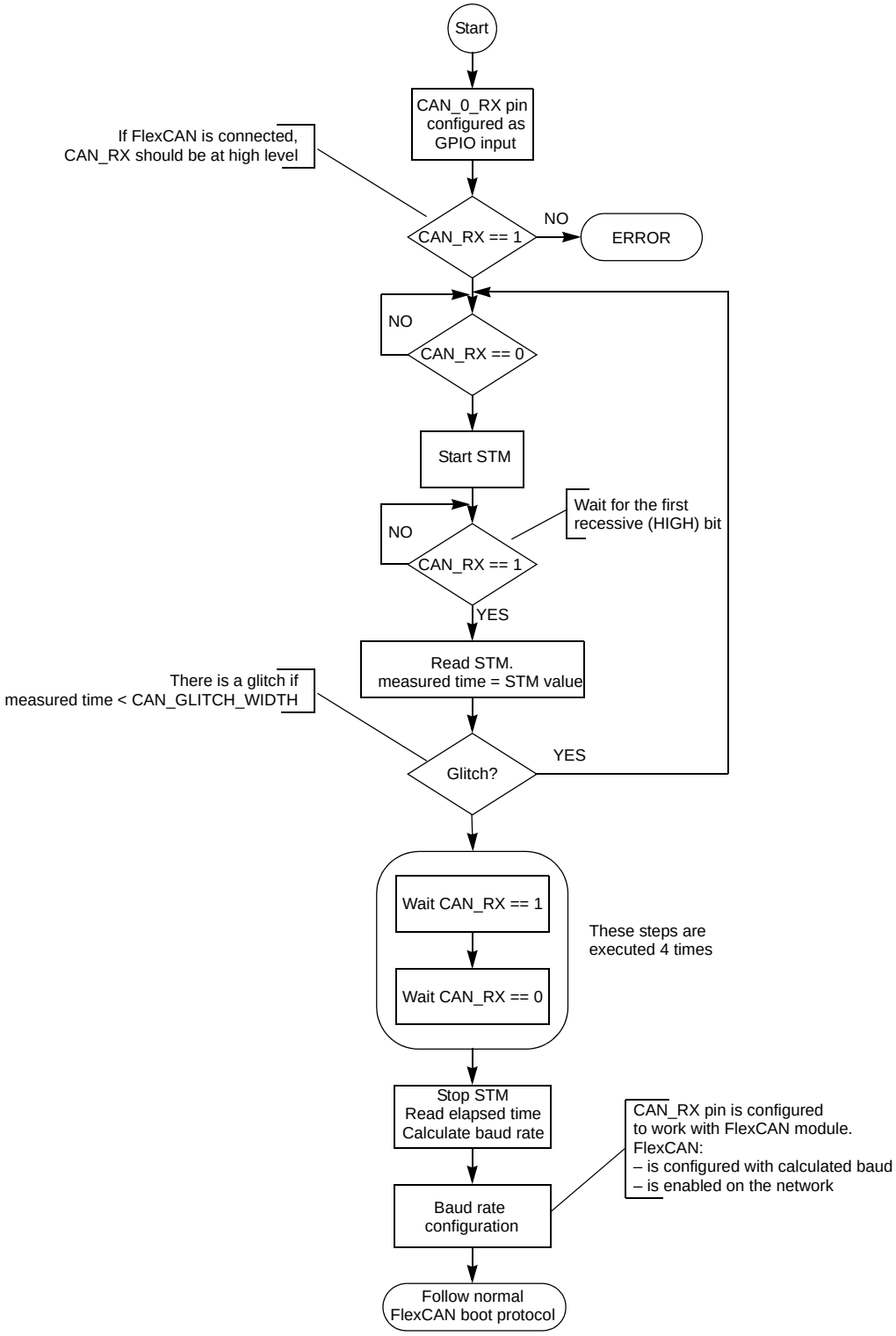


Figure 12-13. BAM rate measurement flow during FlexCAN boot

12.6.7.4 Choosing the host baud rate

The calculation of the FlexCAN baud rate allows the operation of the boot loader with a wide range of baud rates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

Pins are measured until reception of the 5th recessive bit. Thus a total of 29 bit-times are measured (see Figure 12-12).

To minimize errors when calculating bit times and prescalers, operate with the minimum system clock prescaler divider (CAN_CR[PRESDIV]) and the maximum number of time quanta possible.

After measuring the 29 bit times, the result stored in the STM time base is used to select PRESDIV. The number of time quanta in a FlexCAN bit time is given by:

Bit_time = SYNCSEG + TSEG1 + TESG2

SYNCSEG = Exactly one time quantum.

TSEG1 = PROGPSEG + PSEG1 + 2

TSEG2 = PSEG2 + 1

Time base result = 29 × (Presdiv + 1) × (SYNCSEG + TSEG1 + TSEG2)

The FlexCAN protocol specifies that the FlexCAN bit timing should comprise a minimum of 8 time quanta and a maximum of 25 time quanta. Therefore available range is:

$8 \leq 1 + TSEG1 + TESG2 \leq 25$

For 29 bit times, the possible range in which the result in the time base may reside, accounting for PRESDIV, is:

$(232 \times (1+PRESDIV)) \leq \text{time base} \leq (725 \times (1 + PRESDIV))$

Therefore, the available values of the time base can be divided into windows of 725 counts.

Table 12-11. Prescaler/divider and time base values

PRESDIV	Time base Minimum	Time base Maximum
0	232	725
1	726	1450
2	1451	2175
3	2176	2900

In the BAM, the time base is divided by 726, the remainder is discarded. The result provides the CAN_CR[PRESDIV] to be selected.

To help compensate for any error in the calculated baud rate, the resynchronization jump width is increased from its default value of 1 to a fixed value of 2 time quanta.This is the maximum value allowed that can accommodate all permissible FlexCAN baud rates. See Table 12-12.

Table 12-12. FlexCAN standard compliant bit timing segment settings

Time Segment 1	Time Segment 2	RJW
5..10	2	1..2
4..11	3	1..3

Table 12-12. FlexCAN standard compliant bit timing segment settings (continued)

Time Segment 1	Time Segment 2	RJW
5..12	4	1..4
6..13	5	1..4
7..14	6	1..4
8..15	7	1..4
9..16	8	1..4

Timing segment 2 is kept as large as possible to keep the sample time within the bit time.

Table 12-13. Lookup table for FlexCAN bit timings

Desired number of Time quanta (DTq)	Time Segment 2	Time segment 1	
	PSEG2+1	PSEG1+1	PROPSEG+1
8 to 13[1]	2	2	DTq – 5
8 to 13[2]	3	2	DTq – 6
14 to 15	3	3	DTq – 6
16 to 17	4	4	DTq – 7
18 to 19	5	5	DTq – 8
20 to 21	6	6	DTq – 9
22 to 23	7	7	DTq – 10
24 to 25	8	8	DTq – 11

1 PRESDIV + 1 > 1

2 PRESDIV + 1 = 1 (to accommodate information processing time IPT of 3 tq)
NOTE: All TSEG1 and TSEG2 times have been chosen to preserve a sample time between 70% and 85% of the bit time.

Table 12-14. Presdiv +1 = 1

Desired number of time quanta	Register contents for CANA_CR
8	0x004A_2001
9	0x004A_2002
10	0x004A_2003
11	0x004A_2004
12	0x004A_2005
13	0x004A_2006

Table 12-15. Presdiv+1 >1 (YY = PRESDIV)

Desired number of time quanta	Register contents for CANA_CR
8	0xYY49_2002
9	0xYY49_2003
10	0xYY49_2004
11	0xYY49_2005
12	0xYY49_2006
13	0xYY49_2007
14	0xYY52_2007
15	0xYY52_2008

Table 12-15. Presdiv+1 >1 (YY = PRESDIV) (continued)

Desired number of time quanta	Register contents for CANA_CR
16	0xYY5B_2008
17	0xYY5B_2009
18	0xYY64_2009
19	0xYY64_200A
20	0xYY6D_200A
21	0xYY6D_200B
22	0xYY76_200B
23	0xYY76_200C
24	0xYY7F_200C
25	0xYY7F_200D

Examples showing the FlexCAN autobaud rate:

Example 12-3. 8 MHz crystal

Consider the case where using an 8 MHz crystal, user attempts to send 1 MB (max permissible baud rate) FlexCAN message.

— Time base, clocking at crystal frequency, would measure:
— 1 MB = 8 clocks/bit → 29 × 8 = 232 clocks
— To calculate PRESDIV = 232/725 → PRESDIV = 0
— To calculate time quanta requirement:
— Time base result = 29 × (Presdiv+1) × (SYNCSEG + TSEG1 + TSEG2)
— 232 = 29 × 1 × (1 + TSEG1 + TSEG2)
— 1 + TSEG1 + TSEG2 = 8.
— From the lookup table, CANA_CR = 0x004A_2001.
— This give a baud rate of X. This gives 0% error.

Example 12-4. 20 MHz crystal

Consider the case where using a 20 MHz crystal, user attempts to send 62.5 Kbaud FlexCAN message.

— Time base, clocking at crystal frequency, would measure:
— 62.5 Kbaud = 320 clocks/bit → 29 × 320 = 9280 clocks
— To calculate PRESDIV = 9280/725 = 12r580 → PRESDIV = 12
— To calculate time quanta requirement:
— Time base result = 29 × (PRESDIV + 1) × (SYNCSEG + TSEG1 + TSEG2)
— 9280 = 29 × 13 × (1 + TSEG1 + TSEG2)
— 1 + TSEG1 + TSEG2 = 24.6~ 25 (need to round up — S/W must track remainder)
— From the lookup table, CANA_CR = 0x0C7F_200D.

— This gives a baud rate of 61.538 Kbaud
— This equates to an error of: ~1.6%

This excludes cycle count error introduced due to software polling, likely to be ~6 system clocks.

12.6.8 Interrupt

No interrupts are generated or enabled by the BAM.

Chapter 13
Clock Architecture

13.1 System clock generation

Table 13-1 lists the clock signals generated on the MPC5675K.

Table 13-1. MPC5675K clock signals

Clock	Description	Minimum frequency	Maximum frequency
IRCOSC	Internal RC oscillator. 16 MHz nominal. See Section 40.2, IRCOSC 16 MHz internal RC oscillator.	16 MHz nominal	
XOSC	External crystal oscillator 4–40 MHz. See Section 40.1, XOSC external oscillator.	4 MHz	40 MHz
SYS_CLK	System clock.	8 MHz	90 MHz
CORE0_CLK	Clock for Core_0. Since Core_0 and Core_1 have the same clock source, CORE0_CLK and CORE1_CLK always have the same frequency.	8 MHz	180 MHz
CORE1_CLK	Clock for CORE_1. Since Core_0 and Core_1 have the same clock source, CORE0_CLK and CORE1_CLK always have the same frequency.	8 MHz	180 MHz
DRAMC_CLK	Clock for DRAM controller. The external DRAM clock receives half this frequency from DRAMC_CLK. **Note:** The minimum programmable speed is 16 MHz, but this clock should be set to a minimum of 60 MHz. In addition, the customer must align this speed with the selected memory being used.	60 MHz (see clock description)	180 MHz
FLASH_CLK	Clock for Flash memory.	4 MHz	45 MHz
CLK_OUT	External clock out.	2 MHz	60 MHz
FMPLL0_CLK	PHI output of PLL0.	16 MHz	180 MHz
FMPLL0_PCS_CLK	Progressive clock switching output (PHI_PCS) of the FMPLL_0. When progressive clock switching finishes, this clock has the same frequency as the PHI output.	16 MHz	180 MHz
FMPLL1_1D0_CLK	PHI output of PLL1. FMPLL_1D0_CLK = F_{VCO}/ODF	16 MHz	120 MHz
FMPLL1_1D1_CLK	PHI1 output of PLL1. FMPLL_1D1_CLK = F_{VCO}/6	42.66 MHz	80 MHz
SoR_Part_0_CLK	Clock signal for the Sphere of Replication Part 0 (SoR_0). Uses SYS_CLK as reference. Since SoR_0 and SoR_1 have the same clock source, SoR_Part_0_CLK and SoR_Part_1_CLK always have the same frequency.	8 MHz	90 MHz

Table 13-1. MPC5675K clock signals (continued)

Clock	Description	Minimum frequency	Maximum frequency
SoR_Part_1_CLK	Clock signal for the Sphere of Replication Part 1(SoR_1). Uses SYS_CLK as reference. Since SoR_0 and SoR_1 have the same clock source, SoR_Part_0_CLK and SoR_Part_1_CLK always have the same frequency.	8 MHz	90 MHz
PERI0_CLK	Peripheral clock 0. Module clock for these peripherals: • CRC0, 1 • DSPI0, 1, 2 • FlexCAN0	8 MHz	90MHz
PERI1_CLK	Peripheral clock 1. Module clock for these peripherals: • DSPI2 • FlexCAN1, 2, 3 • I^2C0, 1, 2 • LINFlex0, 1, 2 • PIT	4 MHz	45 MHz
MOTC_CLK	Motor control clock (FlexPWM). Module clock for these Motor Control related peripherals: • ADC0, 1, 2, 3 • CTU0, 1 • eTimer0, 1, 2 • FlexPWM0, 1, 2	4 MHz	120 MHz
FR_CLK	FlexRay clock. Protocol clock engine for the FlexRay module. Uses 80 MHz source to generate two 90 degree out of phase protocol clocks.	40 MHz (when using crystal oscillator)	80 MHz (when using PLL)
CANPE_CLK	FlexCAN clock. Protocol engine clock of all FlexCAN instantiations. Uses 80 MHz source to generate 40 MHz CAN clock sources. Protocol clock for FlexCAN cannot exceed its ipg_clk, which is PERI1_CLK for CAN1/2/3. Hence, the Aux divider should always be set to 1 before using FlexCAN clocks.	4 MHz	40 MHz

Figure 13-1 shows the MPC5675K clock generation.

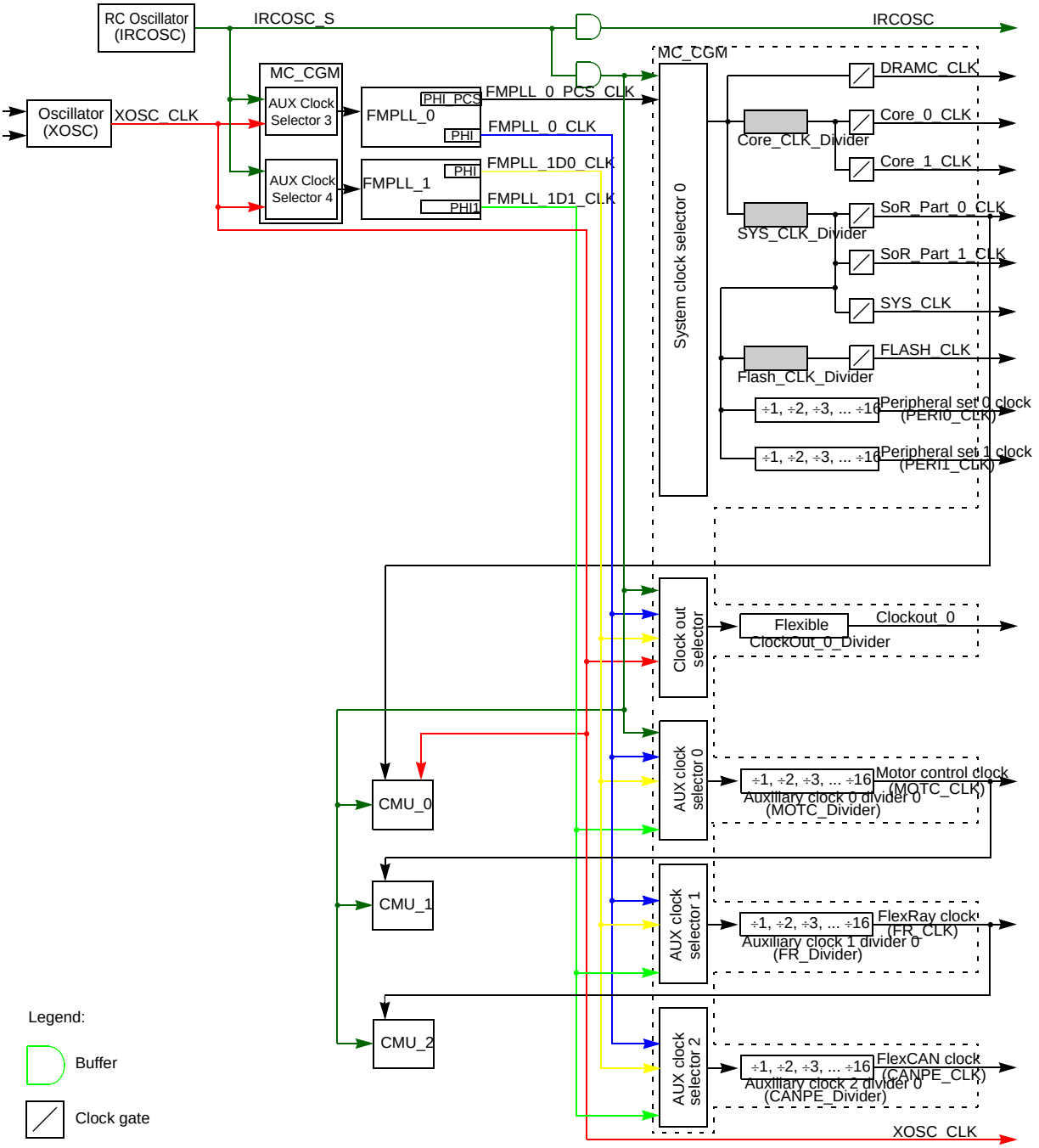


Figure 13-1. MPC5675K system clock generation

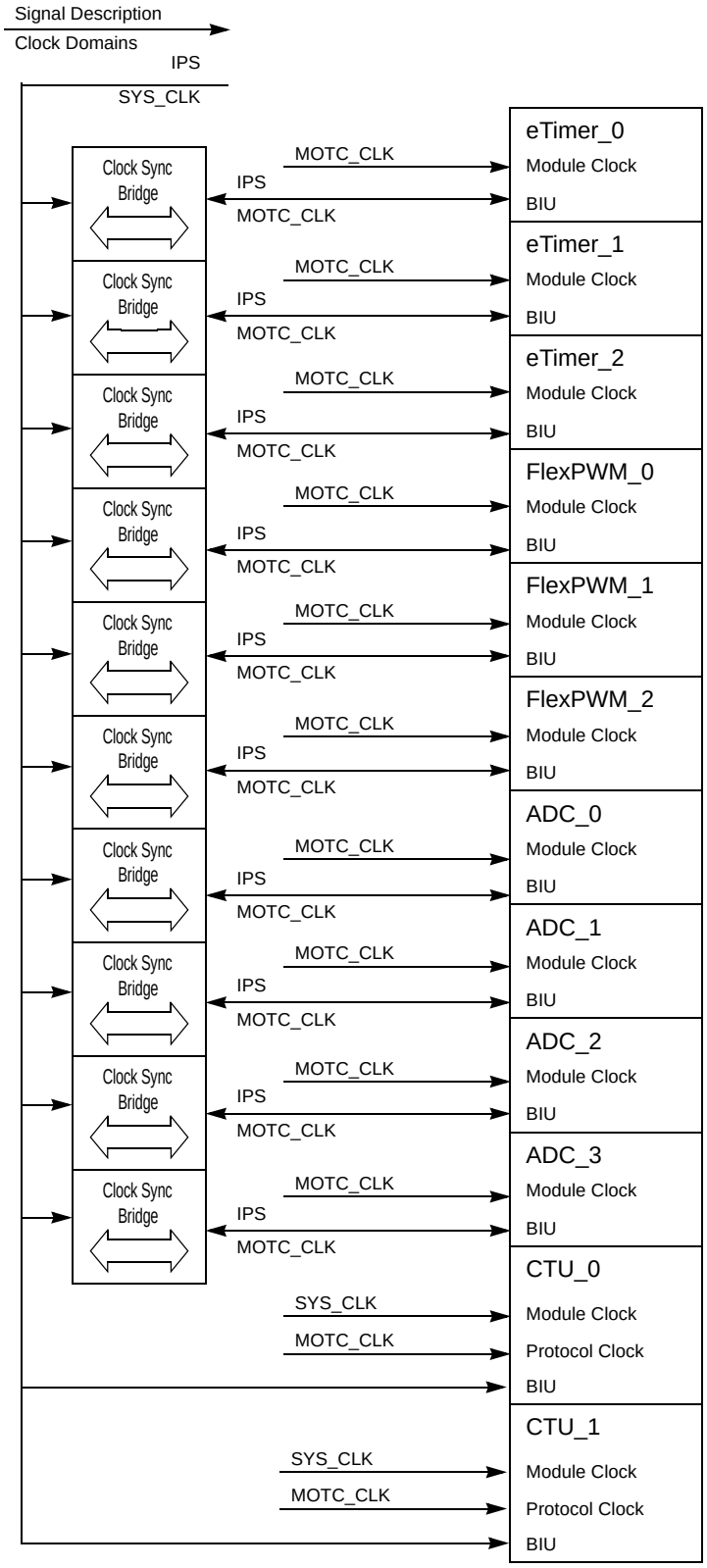


Figure 13-2. MPC5675K system clock distribution part A

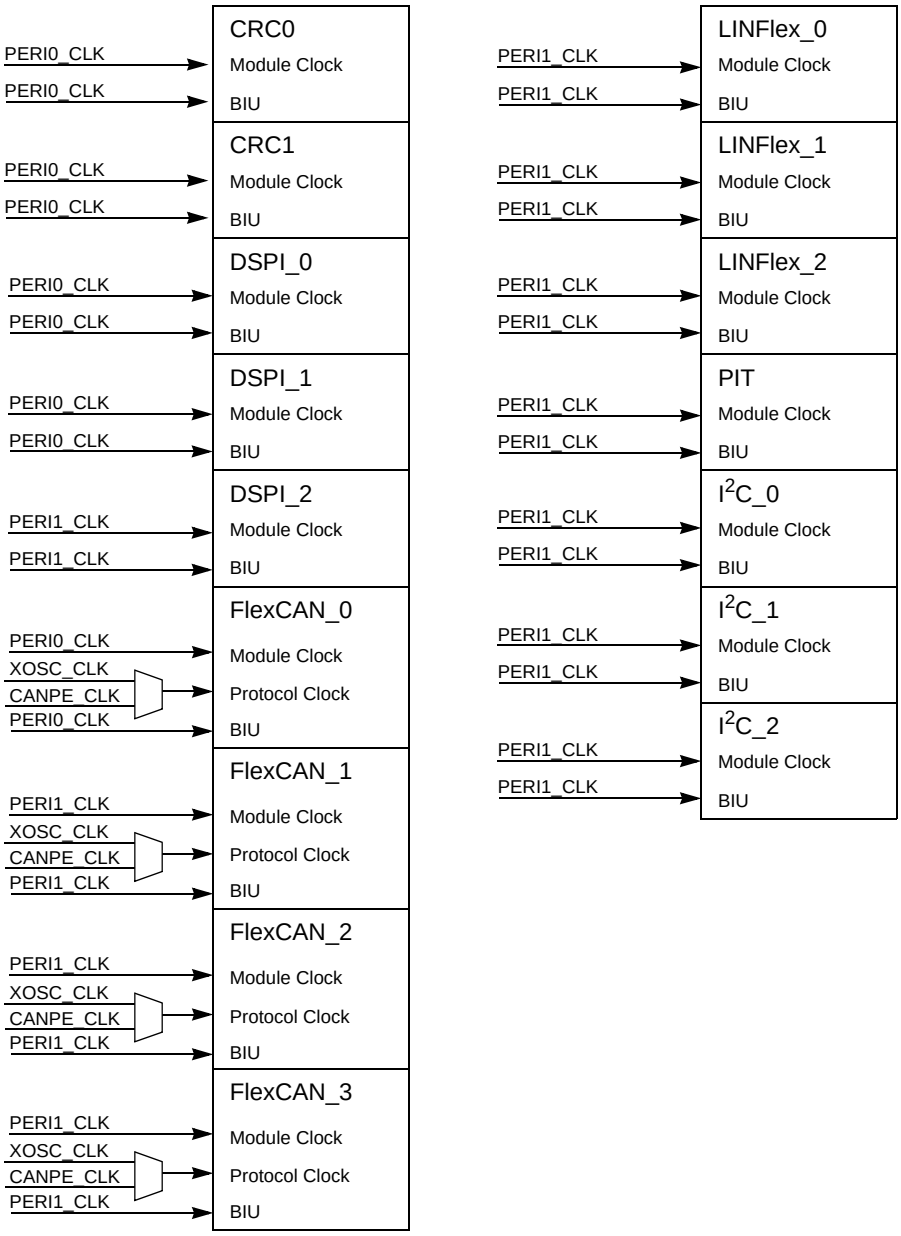


Figure 13-3. MPC5675K system clock distribution part B

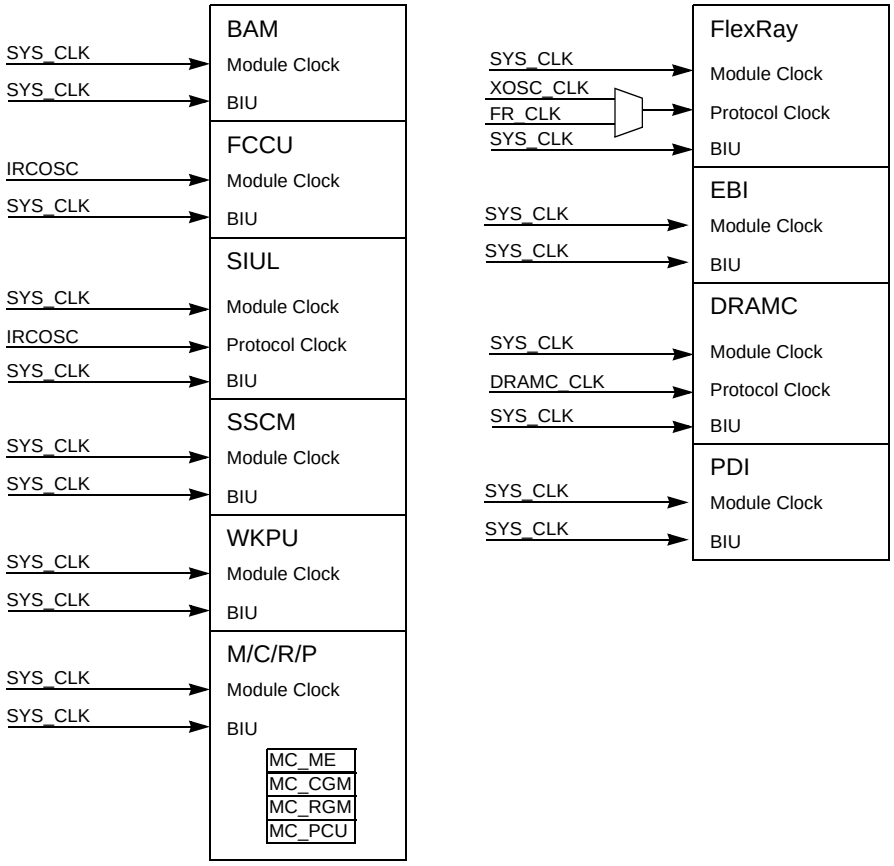


Figure 13-4. MPC5675K system clock distribution part C

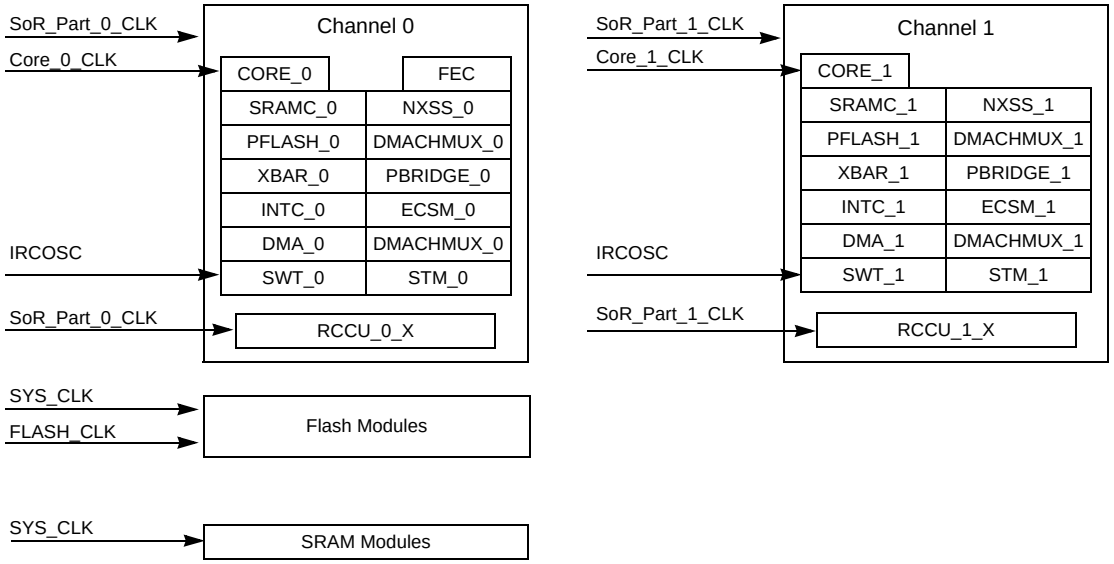


Figure 13-5. MPC5675K system clock distribution part D

13.2 Clock selection

The MPC5675K devices provide the following clock selectors that are programmable in the Clock Generation Module (MC_CGM) or in Shadow flash, depending on the selector.

- System Clock Selector 0
- External Clock Selector
- Auxiliary Clock Selector 0
- Auxiliary Clock Selector 1
- Auxiliary Clock Selector 2
- Auxiliary Clock Selector 3
- Auxiliary Clock Selector 4

The design architecture provides slots for multiple clock inputs, but most input slots are not used. Thus, these unused clock selector inputs are not selectable. In the following descriptions, the input number reflects the decimal value programmed into the SELCTL bitfield of the respective clock selector control register.

13.2.1 System Clock Selector 0

The main part of the device is operated from the internal RC oscillator (IRCOSC) clock or from the progressive output clock of the FMPLL_0 called PHI_PCS.

The System Clock Selector 0 can be configured via software. The System Clock Selector 0 selects the clock source for the system clock (SYS_CLK and derivatives) among the before mentioned different clock signals.

The system clock selector also provides the DRAMC_CLK clock signal used by the protocol engine of the DRAMC to generate the DRAM interface clock to the external DRAM memory.

The following clock source assignments are used for the System Clock Selector on any MPC5675K device. Note that not all inputs are used. See also Figure 13-1.

- Input # 0: IRCOSC (fixed output); default selection after reset
- Input # 4: FMPLL_0 (progressive output called PHI_PCS)

Unused clock selector inputs are not selectable.

13.2.2 External Clock Selector

A clock output selector is provided that allows software to select among these clock sources:

- Input # 0: IRCOSC (fixed output); default selection after reset
- Input # 1: XOSC_CLK (fixed output)
- Input # 2: FMPLL_0 (non-progressive output called PHI)
- Input # 3: FMPLL_1 (non-progressive output called PHI)

13.2.3 Auxiliary Clock Selector 0

The Auxiliary Clock Selector 0 is fed by the IRCOSC clock from the internal RC-Oscillator, the non-progressive output of the FMPLL_0 called PHI, the non-progressive output of the FMPLL_1 called PHI, and the non-progressive output of the FMPLL_1 called $F_{VCO}/6$.

The Auxiliary Clock Selector 0 is configured via software and neither supports glitch free nor hardware initiated source switching. The Auxiliary Clock 0 divider must be disabled while software is configuring clock sources. This means that when they are re-enabled, they start from a known value and behave deterministically. The Auxiliary Clock Selector 0 selects the clock source for the clock divider called MOTC_Divider.

The following clock source assignments are used for the Auxiliary Clock Selector 0 on any MPC5675K device. Note that not all inputs are used. See also Figure 13-1.

- Input # 0: IRCOSC (fixed output); default selection after reset
- Input # 4: FMPLL_0 (non-progressive output called PHI)
- Input # 5: FMPLL_1 (non-progressive output called PHI)
- Input # 8: FMPLL_1 (non-progressive output called $F_{VCO}/6$)

Unused clock selector inputs are not selectable.

13.2.4 Auxiliary Clock Selector 1

The Auxiliary Clock Selector 1 is fed by the non-progressive output of the FMPLL_0 called PHI, the non-progressive output of the FMPLL_1 called PHI and the non-progressive output of the FMPLL_1 called $F_{VCO}/6$.

The Auxiliary Clock Selector 1 is configured via software and neither supports glitch free nor hardware initiated source switching. The Auxiliary Clock 1 divider must be disabled while software is configuring clock sources. This means that when they are re-enabled, they start from a known value and behave deterministically. The Auxiliary Clock Selector 1 selects the clock source for the clock divider called FR_Divider.

The following clock source assignments are used for the Auxiliary Clock Selector 0 on any MPC5675K device. Note that not all inputs are used. See also Figure 13-1.

- Input # 4: FMPLL_0 (non-progressive output called PHI); default selection after reset
- Input # 5: FMPLL_1 (non-progressive output called PHI)
- Input # 8: FMPLL_1 (non-progressive output called $F_{VCO}/6$)

Unused clock selector inputs are not selectable.

13.2.5 Auxiliary Clock Selector 2

The Auxiliary Clock Selector 2 is fed by the non-progressive output of the FMPLL_0 called PHI, the non-progressive output of the FMPLL_1 called PHI and the non-progressive output of the FMPLL_1 called $F_{VCO}/6$.

The Auxiliary Clock Selector 2 is configured via software and neither supports glitch free nor hardware initiated source switching. The Auxiliary Clock 2 divider must be disabled while software is configuring clock sources. This means that when they are re-enabled, they start from a known value and behave deterministically. The Auxiliary Clock Selector 2 selects the clock source for the clock divider called CANPE_Divider.

The following clock source assignments are used for the Auxiliary Clock Selector 2 on any MPC5675K device. Note that not all inputs are used. See also Figure 13-1.

- Input # 4: FMPLL_0 (non-progressive output called PHI); default selection after reset
- Input # 5: FMPLL_1 (non-progressive output called PHI)
- Input # 8: FMPLL_1 (non-progressive output called $F_{VCO}/6$)

Unused clock selector inputs are not selectable.

13.2.6 Auxiliary Clock Selector 3

The Auxiliary Clock Selector 3 is fed by the IRCOSC from the internal RC-Oscillator and the XOSC_CLK directly coming from the XOSC.

The Auxiliary Clock Selector 3 is configured via software and neither supports glitch free nor hardware initiated source switching. This selector should only be changed when PLL0 is powered down. The Auxiliary Clock Selector 3 selects the clock source for the FMPLL_0.

The default clock source for the Auxiliary Clock Selector 3 is the IRCOSC signal.

13.2.7 Auxiliary Clock Selector 4

The Auxiliary Clock Selector 4 is fed by the IRCOSC from the internal RC Oscillator and the XOSC_CLK directly coming from the XOSC.

The Auxiliary Clock Selector 4 is configured via software and neither supports glitch free nor hardware initiated source switching. This selector should only be changed when PLL1 is powered down. The Auxiliary Clock Selector 4 selects the clock source for the FMPLL_1.

The default clock source for the Auxiliary Clock Selector 4 is the IRCOSC signal.

13.3 System clock dividers

The MPC5675K device provides two types of clock dividers for thee system clocks. These are implemented in the NVUSRO[CR] field (see Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO)), which affect all system clocks; and in the Clock Generation Module, which allows division on the PERI0 and PERI1 system clocks.

Table 13-2 shows the system clock divider settings that are supported by the MPC5675K via the NVUSRO[CR] field. See Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO), for more information.

Table 13-2. System clock frequency ratios supported via NVM user option bits

	Reset Phase Configuration		1:2 Mode (Default after Reset)		1:3 Mode		1:1 Mode		
NVUSRO[CR1:0] (NVM option bits)	**00**		**11**		**10**		**01**		
	Divider	**[MHz]**	**Divider**	**[MHz]**	**Divider**	**[MHz]**	**Divider**	**[MHz]**	**Remark**
System Clock Selector Input Clock	—	16	—	180	—	180	—	180	—
DRAMC_CLK	—	16	—	180	—	180	—	180	DRAMC available only in 1:2 and 1:1 mode
CORE0_CLK and CORE1_CLK (Core_CLK_Divider)	1	16	1	180	1	180	2	90	Divider selectable in NVM option bits only
SYS_CLK (SYS_CLK_Divider)	1	16	2	90	3	60	2	90	Divider selectable in NVM option bits only
FLASH_CLK (Flash_CLK_Divider)	1	16	2	45	2	30	2	45	Divider selectable in NVM option bits only
Peri0 Clock	1	16	1	90	1	60	1	90	Additional divider selectable in CGM_SC_DC0 register
Peri1 Clock	1	16	2	45	2	30	2	45	Additional divider selectable in CGM_SC_DC1 register

13.3.1 SYS_CLK, Core_CLK, and Flash_CLK divider

The SYS_CLK_Divider system clock divider allows generating a reduced clock frequency for both parts of the SoR (excluding the core complex subsystems) and the majority of the device not running on any of the other clock dividers. This divider determines whether the platform of the device operates in a 2:1, 1:1 or 3:1 clock ration between the two cores and the platform/rest of the device.

The Core_CLK_Divider, the SYS_CLK_Divider, and the Flash_CLK_Divider can only be configured via the corresponding NVM user option bits. These two bits allow divide ratios between these three clock dividers, as shown in Table 13-2.

The SYS_CLK_Divider is enabled after reset and is set according to Table 13-2 to enable the default mode of operation of 2:1 between the core and the platform frequency.

13.3.2 PERI0 divider

The PERI0_Divider system clock divider allows generating a reduced clock frequency for selected peripherals that can run on a lower frequency without performance degradation (for example, being able to meet their required throughput numbers while running on a clock slower than the CPU clock). Refer to Figure 13-2 and Figure 13-3 for the selected peripherals that are operated with this clock.

The PERI0_Divider is enabled after reset and is set to ÷ 1.

13.3.3 PERI1 divider

The s PERI1_Divider system clock divider allows generating a reduced clock frequency for selected peripherals that can run on a lower frequency without performance degradation (for example, being able to meet their required throughput numbers while running on a clock slower than the CPU clock). Refer to Figure 13-2 and Figure 13-3 for the selected peripherals that are operated with this clock.

The PERI1_Divider is enabled after reset and is set to ÷ 1.

13.3.4 External clock divider

The clock signal generated by this output can be used as a clock source for external peripherals like the sensor connected to the PDI interface. For the PDI sensor the clock does not need to be aligned to any other interface signals as the PDI interface has its own clock input signal. This clock output is not meant to be used as the external EBI interface clock. The EBI interface clock (separate pin) needs to be generated on the device as described in the EBI BG/IG.

One output clock divider provides a 50% duty cycle (nominal) clock output signal and allows the selected output clock source to be divided with these divide options:

- ÷1, ÷2, ÷4, ÷8

The External Clock Divider is enabled after reset and is set to no division.

13.3.5 Auxiliary clock dividers

Three auxiliary clock dividers are implemented to generate these clock signals:

- MOTC_CLK
- FR_CLK
- CANPE_CLK

All dividers support these divide options:

- ÷1, ÷2, ÷3, ÷4, ÷5, ÷6, ÷7, ÷8, ÷9, ÷10, ÷11, ÷12, ÷13, ÷14, ÷15, ÷16

The auxiliary clock dividers are enabled after reset and are set to ÷1.

MOTC_CLK is the module clock for the motor control related peripherals. Its divider is controlled by the CGM_AC0_SC[SELCTL] bitfield.

FR_CLK is the protocol clock engine for the FlexRay module. Its divider is controlled by the CGM_AC1_SC[SELCTL] bitfield.

CANPE_CLK is the protocol engine clock for the FlexCAN modules. Its divider is controlled by the CGM_AC2_SC[SELCTL] bitfield.

NOTE

This requires that all FlexCAN modules use the same clock.

These dividers allow to run the FlexRay module with the target frequency of 80 MHz and the motor control related peripherals with a clock of 120 MHz originating from FMPLL_1. For motor control applications with reduced accuracy requirements but stricter power consumption requirements in RUN mode, the MOTC_CLK can be reduced to 80 MHz or even down to 60 MHz.

This allows to provide the DRAMC with a clock that has twice the frequency of the system clock. This clock is used by the DRAMC at the interface to the external DRAM memories.

13.4 Clock tree architecture functional safety

This section defines the requirements for the MPC5675K clock tree architecture.

13.4.1 Clock monitoring

The monitoring input of CMU_0 is connected to the clock signal routed to the SoR Part 0 (behind the clock gate). Clock frequency failure or failure in propagating the clock signal along the clock tree part that is common for both parts of the SoR will be covered by CMU_0. A failure in the separated / replicated clock tree routed to each part of the SoR will be detected by one or more RCCU instantiations, depending on the failure propagation inside the SoR.

13.4.2 Clock domains and clock tree

The clock tree is independent between any two lakes of the SoR. Both instantiations of the SWT are clocked by the IRCOSC clock. No selection option is available. There must not be any dependencies between the clock input to the SWTs and the CGM, RGM, or ME modules.

The FCCU runs on the IRCOSC clock. There must not be any dependencies between the clock input to the FCCU and the CGM, RGM, or ME modules. The FCCU must always run on the IRCOSC. No selection option is available.

The RCCUs are clocked as shown in Figure 13-4.

13.4.3 Safe Mode

When entering the Safe mode, the MPC5675K switches the System Clock Selector 0 to the IRCOSC originating from the IRCOSC. The Auxiliary Clock Selectors are not affected of any automatic switching taking place when the system changes mode. The Auxiliary Clock Selectors are always switched under software control.

Upon entering the Safe mode, peripherals can be clock gated and the output buffers of pads controlled by the SIUL can be switched off. Whether a peripheral is clock gated in Safe mode can be configured by software inside the CGM module. Whether the output driver of a pad controlled by the SIUL is switched off upon entering Safe mode can be configured by software inside the SIUL on a pad-by-pad level.

13.5 Alternate module clock domains

13.5.1 FlexCAN clock domains

The protocol clock domain of the FlexCAN module can either be operated on the CANPE_CLK from the Auxiliary Clock Selector 1 or on the XOSC_CLK directly from the oscillator (XOSC). The clock source options provided by the Auxiliary Clock Selector 1 are defined in Figure 13-1 and Section 13.2.4, Auxiliary Clock Selector 1.

The FlexCAN modules need to be able to operate up to a maximum bus speed of 1.0 MBit/s.

The FlexCAN module has two distinct software controlled clock domains. The first clock domain (Module Clock) is always supplied from the PERI0_CLK. This clock domain includes the Message Buffer logic. The source for the second clock domain (Protocol Clock) can either be the CANPE_CLK or the XOSC_CLK. The logic in the protocol clock domain controls the FlexCAN interface pins.

The CLK_SRC bit in the FlexCAN CTRL register selects between the CANPE_CLK clock and the XOSC_CLK as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the FlexCAN bus. See Section 27.3.4.2, Control Register (CTRL).

System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR register. See Section 27.3.4.1, Module Configuration Register (MCR).

To switch the clock source, the FlexCAN MDIS should be set and thus a simple mux is used to select the clock instead of a glitch free clock switcher.

The clock frequency used for the FlexCAN Module Clock domain must always be equal or greater than the clock frequency used for the FlexCAN protocol clock domain. The EQUAL condition applies to the CANPE_CLK selected by CLK_SRC = 1. The GREATER_THAN condition applies to the XOSC_CLK selected by CLK_SRC = 0.

NOTE: Robustness regarding clocks of Communication Protocol Engines

The clock domain (Protocol Clock) of the Communication Protocol Engines (FlexRay Module and FlexCAN Module) can either be fed from one of the on-chip frequency modulated PLLs (FMPLL0 or FMPLL1) or directly from the external crystal oscillator. To improve the robustness of the system regarding clock disturbs picked up by components of the external and internal crystal oscillator circuitry, a low pass filter is integrated within the circuitry of the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1). Using the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1) as the clock source for Communication Protocol Engines (FlexRay Module and FlexCAN Module) will reduce the probability of external clock disturb related failures. Using directly the external crystal oscillator as clock source for a Communication Protocol Engine (FlexRay Module or FlexCAN Module) may require additional system measures to improve the robustness regarding clock disturbs picked up by the external and internal crystal oscillator circuitry, as the clock direct from the external crystal oscillator is not filtered by an integrated filter circuitry.

13.5.2 FlexRay clock domains

The protocol clock domain of the FlexRay module can either be operated on the FR_CLK from the Auxiliary Clock Selector 2 or on the XOSC_CLK directly from the oscillator (XOSC).

Independent of the protocol clock source the FlexRay module needs to be able to operate up to a maximum bus speed of 10 MBit/s.

The FlexRay block has two distinct clock domains. The first clock domain (Module Clock or CHI clock) is always derived from the SYS_CLK clock. The bus interface unit to the AHB and the IPS bus is located in this clock domain. The source for the second clock domain can either be the FR_CLK clock or the XOSC_CLK clock. The logic in the second clock domain controls the FlexRay interface pins (Protocol Clock or PE clock).

The CLKSEL bit in the FlexRay CTRL register selects between the FR_CLK clock and the XOSC_CLK clock as the clock source for the second domain.

The FR_CLK clock source is selected by the Auxiliary Clock Selector 1 and divided by its divider called FR_Divider. See Section 13.2.4, Auxiliary Clock Selector 1, and Section 13.3.5, Auxiliary clock dividers, for more details.

For a typical FlexRay system running at 10 MBit/s, the protocol clock domain must either be supplied by an 80 MHz FR_CLK or a 40 MHz (50% duty cycle) XOSC_CLK.

NOTE: Robustness regarding clocks of Communication Protocol Engines

The clock domain (Protocol Clock) of the Communication Protocol Engines (FlexRay Module and FlexCAN Module) can either be fed from one of the on-chip frequency modulated PLLs (FMPLL0 or FMPLL1) or directly from the external crystal oscillator. To improve the robustness of the system regarding clock disturbs picked up by components of the external and internal crystal oscillator circuitry, a low pass filter is integrated within the circuitry of the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1). Using the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1) as the clock source for Communication Protocol Engines (FlexRay Module and FlexCAN Module) will reduce the probability of external clock disturb related failures. Using directly the external crystal oscillator as clock source for a Communication Protocol Engine (FlexRay Module or FlexCAN Module) may require additional system measures to improve the robustness regarding clock disturbs picked up by the external and internal crystal oscillator circuitry, as the clock direct from the external crystal oscillator is not filtered by an integrated filter circuitry.

13.5.3 SWT clock domains

The SWT module has two distinct clock domains. The first clock domain (Module Clock) is always supplied from the SYS_CLK. This clock domain includes the register interface. The source for the second clock domain (Protocol Clock, clock used for the timer) is always the IRCOSC generated by the IRCOSC. No selection option for any other clock source for the protocol clock domain is provided.

The SWT is part of the SoR and therefore duplicated. This is a duplicated, dual clock domain module. The clock domain synchronization between the two domains takes place in both instantiations of the SWT independently and therefore no cycle accurate lock step configuration is granted.

13.5.4 Cross Triggering Unit (CTU) clock domains

Each CTU module has two distinct clock domains. The first clock domain (Module Clock) is supplied from the SYS_CLK. This clock domain includes the Command Buffer logic. The source for the second clock domain (Protocol Clock) is the MOTC_CLK. The logic in the protocol clock domain controls the CTU interface pins to the eTimer modules and the ADC modules.

13.5.5 IPS bus clock sync bridge

For advanced motor control applications, each FlexPWM module needs to be able to create PWM pulses as short as 100 ns. Customers are asking to provide a positioning accuracy of the PWM pulse of 12 bits at 20 kHz motor control loop cycles. This translates into a resolution of 12 ns or a minimum FlexPWM clock of 84 MHz. The clock frequency for the Motor Control related peripherals is selectable independent of the device main SYS_CLK especially as not all applications require to run the whole device on an 84 MHz or higher clock. The FM broadcast radio band ranging from 86 MHz to 108 MHz should never be used by any module on the device to limit the radiation in that frequency range. The next integer multiple that can satisfy the Motor Control requirements and the FlexRay requirements is 120 MHz.

The sampling of the two ADC converters as well as the input capturing of the two eTimer modules need to be synchronous to the PWM output signals of the FlexPWM module. This synchronization and the required flexibility of the ADC conversions is provided by the Cross Triggering Unit (CTU).

The available IP of the ADC, the eTimer and the FlexPWM modules does not allow the modules to be run on two different clock domains like the Module Clock and the protocol clock in case of the FlexCAN module. Due to the project schedule and the available resources the existing IP will not be modified to allow a dual clock domain operation. In order to achieve the required PWM accuracy the whole FlexPWM module needs to run on a higher Module Clock. To prevent clock boundary crossing of the synchronization signals between the motor control related peripherals the following modules shall run from the same high frequency clock signal MOTC_CLK and will therefore be placed behind the PBRIDGE.

- All eTimer instantiations
- All FlexPWM instantiations
- All ADC instantiations

These peripherals will completely run on the MOTC_CLK. In order to synchronize the BIU of these peripherals to the IPS bus that can run on and independent SYS_CLK frequency an “IPS Bus Clock Sync Bridge” is integrated on all MPC5675K devices, as shown in Figure 13-2.

The IPS Bus Clock Sync Bridge synchronizes all bus traffic to and from these motor control related peripherals running on the MOTC_CLK clock derived from the second PLL and the IPS bus running on the SYS_CLK clock. For motor control applications, an access delay for every access to these peripherals of up to 4 additional SYS_CLK cycles plus 3 MOTC_CLK cycles introduced by this bridge is acceptable.

The DMA request and DMA acknowledge signals also need to be synchronized by this bridge.

The IPS Bus Clock Sync Bridge supports clock synchronization from the SYS_CLK domain to the MOTC_CLK domain. It also supports the case in which the SYS_CLK frequency is higher than the MOTC_CLK, and the case in which the MOTC_CLK is higher or equal to the SYS_CLK.

13.5.6 Peripherals behind the IPS Bus Clock Sync Bridge

13.5.6.1 FlexPWM clock domain

The FlexPWM modules have only one clock domain. Each FlexPWM module runs on the MOTC_CLK. Therefore it is placed behind the IPS Bus Clock Sync Bridge.

13.5.6.2 eTimer clock domain

The eTimer modules have only one clock domain. The eTimer modules run on the MOTC_CLK. Therefore they are placed behind the IPS Bus Clock Sync Bridge.

13.5.6.3 ADC clock domains

The ADC modules have only one clock domain. The ADC modules run on the MOTC_CLK. Therefore they are placed behind the IPS Bus Clock Sync Bridge.

13.6 Clock requirements in STOP and HALT mode

In this subsection, the term “resume” describes the transition from STOP or HALT mode back to a RUN mode.

MPC5675K devices do not support the STANDBY mode. However, they support the STOP and the HALT mode. These two modes allow to transfer the device into a limited power saving mode with the configuration options defined in the CGM module. For all MPC5675K devices, the following implementation constraints apply to enable that in all modes of operation, a resume from STOP or HALT mode is always possible without the need to reset the device.

- STOP and HALT mode:
  - SIUL clock is not gateable
  - FCCU clock is not gateable
  - SIUL filter for external interrupt capable pins is always clocked with IRCOSC
  - Resume via interrupt that can be generated by any peripheral that clock is not gated
  - Resume via $\overline{\text{NMI}}$ pin is always possible if once enabled after reset (no software configuration that could block resume afterwards)
- STOP mode:
  - IRCOSC cannot be switched off
  - The System Clock Selector 0 is switched to the IRCOSC and therefore the SYS_CLK is fed by the IRCOSC signal
  - Resume via external interrupt pin is always possible (if not masked)
- HALT mode:

— IRCOSC should not be switched off by software
— The output of the System Clock Selector 0 can only be switched to a running clock input
— Resume via external interrupt pin is always possible (if not masked)

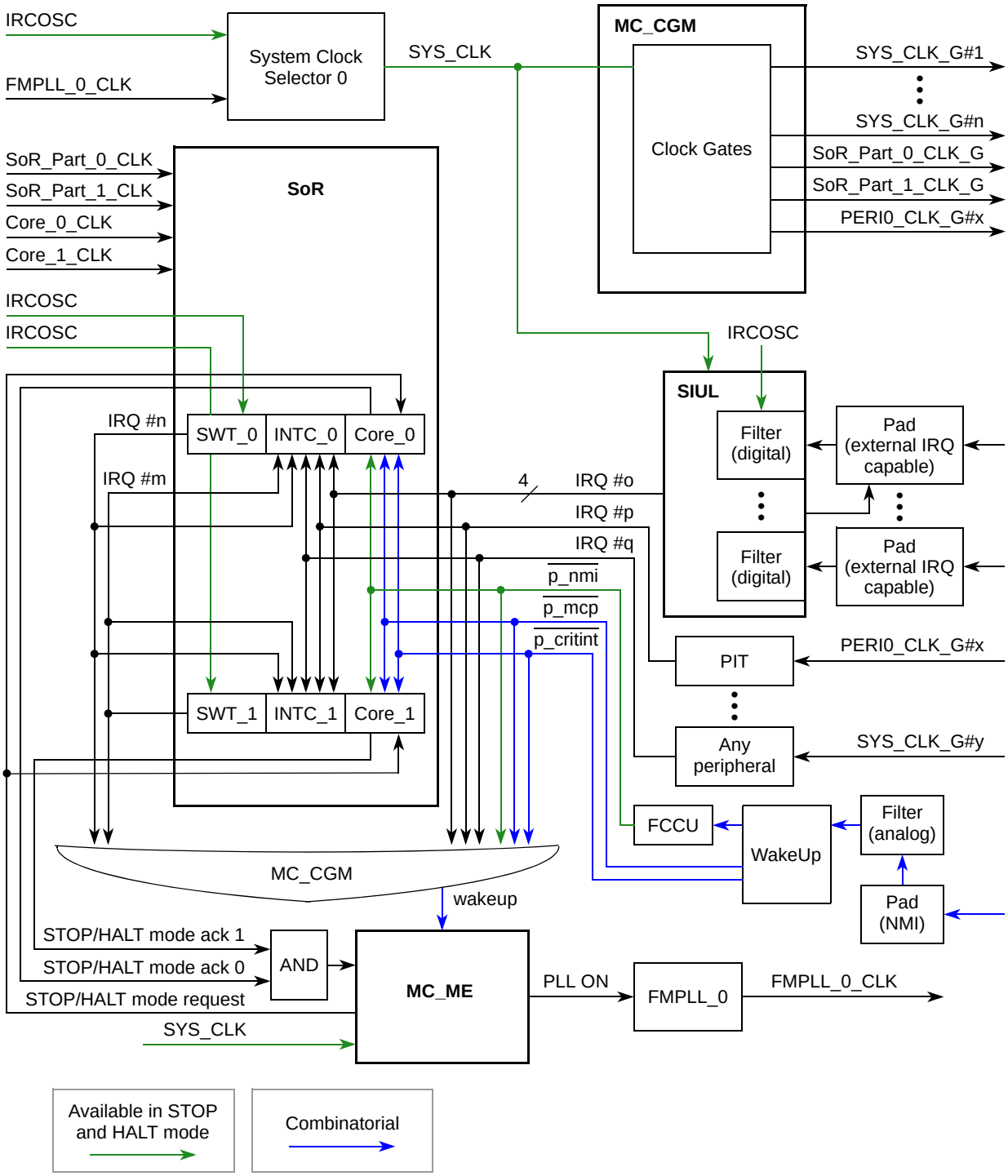


Figure 13-6. Clocks in HALT and STOP mode

13.7 Detailed module descriptions

Additional details on the clock-related modules on this device are provided in the following chapters:

- Chapter 14, Clock Generation Module (MC_CGM)
- Chapter 30, Frequency-Modulated Phase-Locked Loop (FMPLL)
- Chapter 40, Oscillators

Chapter 14
Clock Generation Module (MC_CGM)

14.1 Introduction

This chapter describes the Clock Generation Module (MC_CGM) which includes the functions, pin descriptions, and registers of the MC_CGM module.

14.1.1 Overview

The clock generation module (MC_CGM) generates reference clocks for all the on-chip modules. The MC_CGM selects one of the system clock sources to supply the system clock. The MC_ME controls the system clock selection (see Chapter 35, Mode Entry Module (MC_ME), for more details). A set of MC_CGM registers controls the clock dividers that are used for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources that have addressable memory spaces are accessed through the MC_CGM memory space. The MC_CGM also selects and generates an output clock.

Figure 14-1 shows the MC_CGM block diagram.

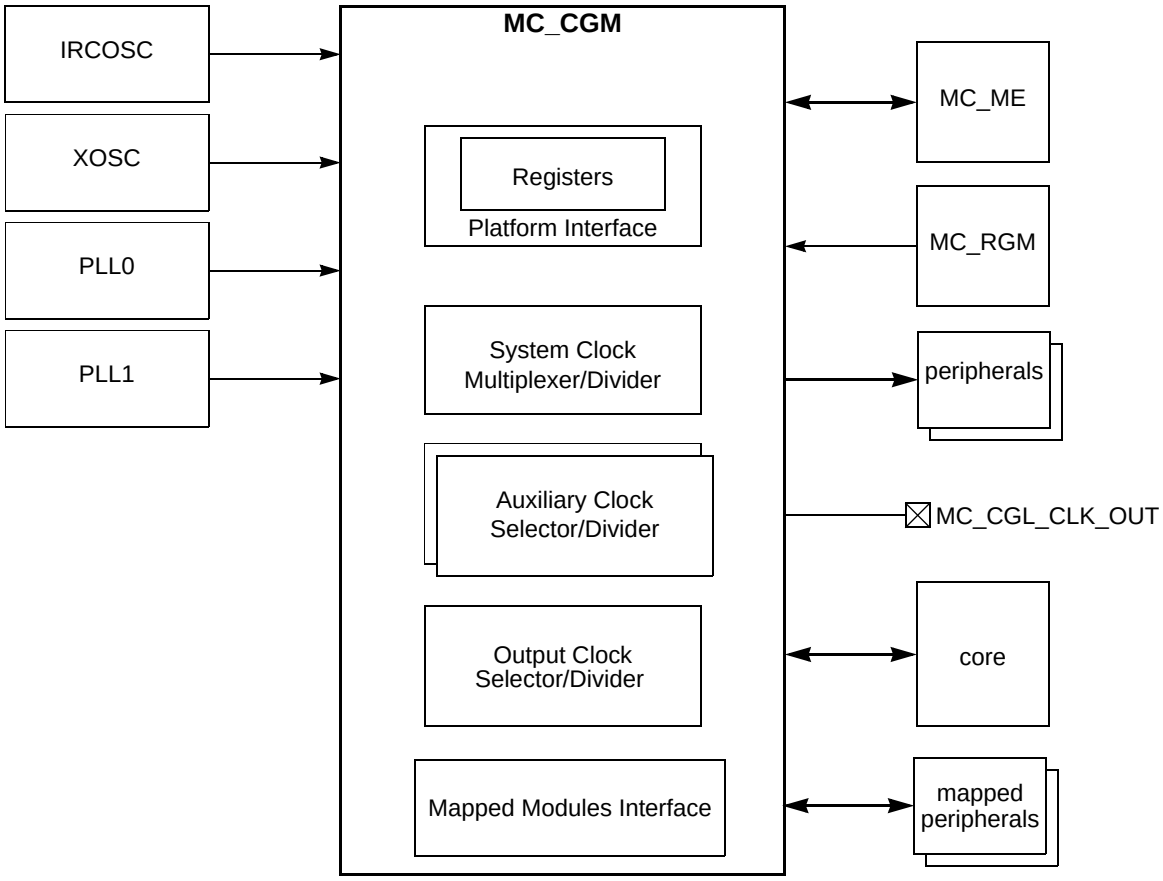


Figure 14-1. MC_CGM Block Diagram

14.1.2 Features

The MC_CGM includes the following features:

- Generates system and peripheral clocks
- Selects and enables/disables the system clock supply from system clock sources according to MC_ME control
- Contains a set of registers to control clock dividers for divided clock generation
- Supports multiple clock sources and maps their address spaces to its memory map
- Generates an output clock
- Enables glitch-less clock transitions when changing the system clock selection
- Supports 8-, 16-, and 32-bit wide read/write accesses

14.2 External signal description

The MC_CGM delivers an output clock to the MC_CGL_CLK_OUT pin for off-chip use and/or observation.

14.3 Memory map and register definition

Table 14-1. MC_CGM register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_0370	CGM_OC_EN	Output Clock Enable	word	read	read/write	read/write	on page 377
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write	on page 378
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read	on page 378
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read	read/write	read/write	on page 379
0xC3FE_037D	CGM_SC_DC1	System Clock Divider Configuration 1	byte	read	read/write	read/write	on page 380
0xC3FE_0380	CGM_AC0_SC	Aux Clock 0 Select Control	word	read	read/write	read/write	on page 380
0xC3FE_0384	CGM_AC0_DC0	Aux Clock 0 Divider Configuration 0	byte	read	read/write	read/write	on page 381
0xC3FE_0388	CGM_AC1_SC	Aux Clock 1 Select Control	word	read	read/write	read/write	on page 382
0xC3FE_038C	CGM_AC1_DC0	Aux Clock 1 Divider Configuration 0	byte	read	read/write	read/write	on page 383

Table 14-1. MC_CGM register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_0390	CGM_AC2_SC	Aux Clock 2 Select Control	word	read	read/write	read/write	on page 383
0xC3FE_0394	CGM_AC2_DC0	Aux Clock 2 Divider Configuration 0	byte	read	read/write	read/write	on page 384
0xC3FE_0398	CGM_AC3_SC	Aux Clock 3 Select Control	word	read	read/write	read/write	on page 384
0xC3FE_03A0	CGM_AC4_SC	Aux Clock 4 Select Control	word	read	read/write	read/write	on page 385

NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

Table 14-2. MC_CGM memory map

Address	Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0000 … 0xC3FE_001C	XOSC registers See Chapter 40, Oscillators”																
0xC3FE_0020 … 0xC3FE_003C	reserved																
0xC3FE_0040 … 0xC3FE_005C	reserved																
0xC3FE_0060 … 0xC3FE_007C	IRCOSC registers See Chapter 40, Oscillators”																

Table 14-2. MC_CGM memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0080 … 0xC3FE_009C	reserved																	
0xC3FE_00A0 … 0xC3FE_00BC	PLL0 registers See Chapter 30, Frequency-Modulated Phase-Locked Loop (FMPLL).																	
0xC3FE_00C0 … 0xC3FE_00DC	PLL1 registers See Chapter 30, Frequency-Modulated Phase-Locked Loop (FMPLL).																	
0xC3FE_00E0 … 0xC3FE_00FC	reserved																	
0xC3FE_0100 … 0xC3FE_011C	CMU0 registers See Chapter 15, Clock Monitor Unit (CMU).																	
0xC3FE_0120 … 0xC3FE_013C	CMU1 registers See Chapter 15, Clock Monitor Unit (CMU).																	
0xC3FE_0140 … 0xC3FE_015C	CMU2 registers See Chapter 15, Clock Monitor Unit (CMU).																	
0xC3FE_0160 … 0xC3FE_0178	reserved																	
0xC3FE_017C	EBI_CR	R	DIV						0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TS_ALE
		W																

Table 14-2. MC_CGM memory map (continued)

Address	Name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0180 … 0xC3FE_019C	reserved																
0xC3FE_01A0 … 0xC3FE_01BC	reserved																
0xC3FE_01C0 … 0xC3FE_01DC	reserved																
0xC3FE_01E0 … 0xC3FE_01FC	reserved																
0xC3FE_0200 … 0xC3FE_021C	reserved																
0xC3FE_0220 … 0xC3FE_023C	reserved																
0xC3FE_0240 … 0xC3FE_025C	reserved																
0xC3FE_0260 … 0xC3FD_C27C	reserved																
0xC3FE_0280 … 0xC3FE_029C	reserved																

Table 14-2. MC_CGM memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_02A0 … 0xC3FE_02BC	reserved																	
0xC3FE_02C0 … 0xC3FE_02DC	reserved																	
0xC3FE_02E0 … 0xC3FE_02FC	reserved																	
0xC3FE_0300 … 0xC3FE_031C	reserved																	
0xC3FE_0320 … 0xC3FE_033C	reserved																	
0xC3FE_0340 … 0xC3FE_035C	reserved																	
0xC3FE_0360 … 0xC3FE_036C	reserved																	
0xC3FE_0370	CGM_OC_EN	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
		W																
0xC3FE_0374	CGM_OCDS_SC	R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 14-2. MC_CGM memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0378	CGM_SC_SS	R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_037C	CGM_SC_DC0 / CGM_SC_DC1	R	DE0	0	0	0	DIV0				DE1	0	0	0	DIV1			
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0380	CGM_AC0_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0384	CGM_AC0_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0388	CGM_AC1_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_038C	CGM_AC1_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0390	CGM_AC2_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0394	CGM_AC2_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 14-2. MC_CGM memory map (continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0398	CGM_AC3_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_039C	reserved																	
0xC3FE_03A0	CGM_AC4_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_03A4 … 0xC3FE_3FFC	reserved																	

14.3.1 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the CGM_OC_EN register may be accessed as a word at address 0xC3FE_0370, as a half-word at address 0xC3FE_0372, or as a byte at address 0xC3FE_0373.

14.3.1.1 EBI Control Register (EBI_CR)

The register is mainly for set divider between sys_clk, an EBI module clock. It is necessary to use the divider when the system clock is higher than 45 MHz because EBI maximum frequency is 45 MHz.

NOTE

Divider should be selected such that EBI clock is <= 45 MHz.

Address 0xC3FE_017C Access: User read/write,, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DIV						0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TS_ALE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-2. EBI Control Register (EBI_CR)

Table 14-3. EBI_CR field descriptions

Field	Description
DIV	EBI Divider 0 No division 1 Divide by 2 1 Divide by 3 1 Divide by 4 ... 63 Divide by 64
TS_ALE	0 TS_B 1 ALE

14.3.1.2 Output Clock Enable Register (CGM_OC_EN)

The Output Clock Enable Register (CGM_OC_EN) enables and disables the output clock.

Address 0xC3FE_0370 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-3. Output Clock Enable Register (CGM_OC_EN)

Table 14-4. CGM_OC_EN field descriptions

Field	Description
EN	Output Clock Enable control 0 Output Clock is disabled. 1 Output Clock is enabled.

14.3.1.3 Output Clock Division Select Register (CGM_OCDS_SC)

The Output Clock Division Select Register (CGM_OCDS_SC) selects the current output clock source and by which factor it is divided before being delivered at the output clock.

Address 0xC3FE_0374 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	0	0	SELDIV		SELCTL			
W								
Reset	0	0	0	0	0	0	0	0

Figure 14-4. Output Clock Division Select Register (CGM_OCDS_SC)

Table 14-5. CGM_OCDS_SC field descriptions

Field	Description
SELDIV	Output Clock Division Select 00 Output selected Output Clock without division. 01 Output selected Output Clock divided by 2. 10 Output selected Output Clock divided by 4. 11 Output selected Output Clock divided by 8.
SELCTL	Output Clock Source Selection Control — This value selects the current source for the output clock. 0000 16 MHz int. RC osc. 0001 4-40 MHz crystal osc. 0010 FMPLL0_CLK. 0011 FMPLL1_1D1_CLK. 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.3.1.4 System Clock Select Status Register (CGM_SC_SS)

The System Clock Select Status Register (CGM_SC_SS) provides the current system clock source selection.

Address 0xC3FE_0378 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-5. System Clock Select Status Register (CGM_SC_SS)

Table 14-6. CGM_SC_SS field descriptions

Field	Description
SELSTAT	System Clock Source Selection Status — This value indicates the current source for the system clock. 0000 16 MHz int. RC osc. 0001 reserved 0010 reserved 0011 reserved 0100 FMPLL0_CLK 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled

14.3.1.5 System Clock Divider Configuration Register 0 (CGM_SC_DC0)

The System Clock Divider Configuration Register 0 (CGM_SC_DC0) controls the system clock divider 0.

Address 0xC3FE_037C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

Figure 14-6. System Clock Divider Configuration Register 0 (CGM_SC_DC0)

Table 14-7. CGM_SC_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable system clock divider 0. 1 Enable system clock divider 0.
DIV0	Divider 0 Division Value — The resultant peripheral set 0 clock has a period DIV0 + 1 times that of the system clock. If the DE0 is set to ‘0’ (Divider 0 is disabled), any write access to the DIV0 field is ignored and the peripheral I/O clock remains disabled.

14.3.1.6 System Clock Divider Configuration Register 1 (CGM_SC_DC1)

The System Clock Divider Configuration Register 1 (CGM_SC_DC1) controls the system clock divider 1.

Address 0xC3FE_037D Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE1	0	0	0	DIV1			
W								
Reset	1	0	0	0	0	0	0	0

Figure 14-7. System Clock Divider Configuration Register 1 (CGM_SC_DC1)

Table 14-8. CGM_SC_DC1 field descriptions

Field	Description
DE1	Divider 1 Enable 0 Disable system clock divider 1. 1 Enable system clock divider 1.
DIV1	Divider 1Division Value — The resultant peripheral set 1 clock has a period DIV1+ 1 times that of the system clock. If the DE1 is set to ‘0’ (Divider 1 is disabled), any write access to the DIV1 field is ignored and the peripheral I/O clock remains disabled.

14.3.1.7 Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

The Auxiliary Clock 0 Select Control Register (CGM_AC0_SC) selects the current clock source for the following clocks:

- Undivided: (unused)
- Divided by auxiliary clock 0 divider 0: motor control clock

See Figure 14-17 for details.

Address 0xC3FE_0380 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-8. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

Table 14-9. CGM_AC0_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 0 Source Selection Control — This value selects the current source for auxiliary clock 0. 0000 16 MHz int. RC osc. 0001 reserved 0010 reserved 0011 reserved 0100 FMPLL0_CLK 0101 FMPLL1_1D0_CLK 0110 reserved 0111 reserved 1000 FMPLL1_1D1_CLK 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.3.1.8 Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)

The Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0) controls the auxiliary clock 0 divider.

Address0xC3FE_0384 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

Figure 14-9. Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)

Table 14-10. CGM_AC0_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 0 divider 0. 1 Enable auxiliary clock 0 divider 0.
DIV0	Divider 0 Division Value — The resultant motor control clock has a period DIV0 + 1 times that of auxiliary clock 0. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the motor control clock remains disabled.

14.3.1.9 Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

The Auxiliary Clock 1 Select Control Register (CGM_AC1_SC) selects the current clock source for the following clocks:

- Undivided: (unused)
- Divided by auxiliary clock 1 divider 0: FlexRay clock

Address 0xC3FE_0388 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-10. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

Table 14-11. CGM_AC1_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 1 Source Selection Control — This value selects the current source for auxiliary clock 1. 0000 reserved 0001 reserved 0010 reserved 0011 reserved 0100 FMPLL0_CLK 0101 FMPLL1_1D0_CLK 0110 reserved 0111 reserved 1000 FMPLL1_1D1_CLK 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.3.1.10 Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)

The Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0) controls the auxiliary clock 1 divider.

Address0xC3FE_038C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

Figure 14-11. Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)

Table 14-12. CGM_AC1_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 1 divider 0. 1 Enable auxiliary clock 1 divider 0.
DIV0	Divider 0 Division Value — The resultant FlexRay clock has a period DIV0 + 1 times that of auxiliary clock 1. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the FlexRay clock remains disabled.

14.3.1.11 Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

The Auxiliary Clock 2 Select Control Register (CGM_AC2_SC) selects the current clock source for the following clocks:

- Undivided: (unused)
- Divided by auxiliary clock 2 divider 0: FlexCAN clock

See Figure 14-19 for details.

Address 0xC3FE_0390 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-12. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

Table 14-13. CGM_AC2_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 2 Source Selection Control — This value selects the current source for auxiliary clock 2. 0000 reserved 0001 reserved 0010 reserved 0011 reserved 0100 FMPLL0_CLK 0101 FMPLL1_1D0_CLK 0110 reserved 0111 reserved 1000 FMPLL1_1D1_CLK 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.3.1.12 Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

The Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0) controls the auxiliary clock 2 divider 0.

Address0xC3FE_0394 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	DE0	0	0	0	DIV0			
W								
Reset	1	0	0	0	0	0	0	0

Figure 14-13. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

Table 14-14. CGM_AC2_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 2 divider 0. 1 Enable auxiliary clock 2 divider 0.
DIV0	Divider 0 Division Value — The resultant FlexCAN clock has a period DIV0 + 1 times that of auxiliary clock 2. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the FlexCAN clock remains disabled.

14.3.1.13 Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

The Auxiliary Clock 3 Select Control Register (CGM_AC3_SC) selects the current clock source for the PLL0 reference clock.

See Figure 14-20 for details.

Address 0xC3FE_0398 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-14. Auxiliary Clock 3 Select Control Register (CGM_AC3_SC)

Table 14-15. CGM_AC3_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 3 Source Selection Control — This value selects the current source for auxiliary clock 3. 0000 16 MHz int. RC osc. 0001 4-40 MHz crystal osc. 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.3.1.14 Auxiliary Clock 4 Select Control Register (CGM_AC4_SC)

The Auxiliary Clock 4 Select Control Register (CGM_AC4_SC) selects the current clock source for the PLL1 reference clock.

See Figure 14-21 for details.

Address 0xC3FE_03A0 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-15. Auxiliary Clock 4 Select Control Register (CGM_AC4_SC)

Table 14-16. CGM_AC4_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 4 Source Selection Control — This value selects the current source for auxiliary clock 4. 0000 16 MHz int. RC osc. 0001 4-40 MHz crystal osc. 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

14.4 Functional Description

14.4.1 System Clock Generation

Figure 14-16 shows the block diagram of the system clock generation logic. The MC_ME provides the system clock select and switch mask (see Chapter 35, Mode Entry Module (MC_ME), for more details), and the MC_RGM provides the safe clock request (see Chapter 46, Reset Generation Module (MC_RGM), for more details). The safe clock request forces the selector to select the 16 MHz int. RC osc. as the system clock and to ignore the system clock select.

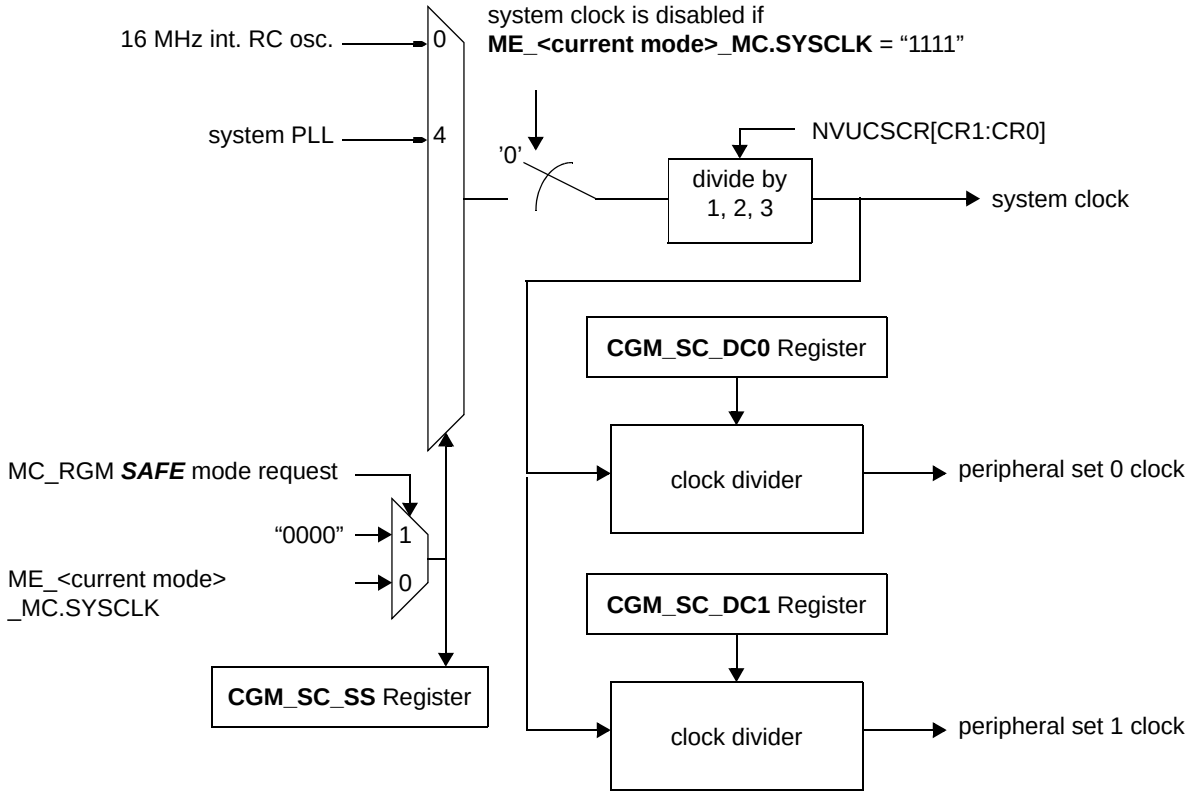


Figure 14-16. MC_CGM System Clock Generation Overview

14.4.1.1 System Clock Source Selection

During normal operation, the system clock selection is controlled

- On a SAFE mode or reset event, by the MC_RGM
- Otherwise, by the MC_ME

14.4.1.2 System Clock Disable

During the TEST mode, the system clock can be disabled by the MC_ME.

14.4.1.3 System Clock Dividers

The MC_CGM generates the following derived clock from the system clock:

- peripheral set 0 clock — controlled by the CGM_SC_DC0 register
- peripheral set 1 clock — controlled by the CGM_SC_DC1 register

14.4.1.4 Auxiliary Clock Generation

Figure 14-16 shows the block diagram of the auxiliary clock generation logic. See Section 14.3.1.7, Auxiliary Clock 0 Select Control Register (CGM_AC0_SC), Section 14.3.1.9, Auxiliary Clock 1 Select

Control Register (CGM_AC1_SC), Section 14.3.1.11, Auxiliary Clock 2 Select Control Register (CGM_AC2_SC), Section 14.3.1.13, Auxiliary Clock 3 Select Control Register (CGM_AC3_SC), and Section 14.3.1.14, Auxiliary Clock 4 Select Control Register (CGM_AC4_SC), for auxiliary clock selection control.

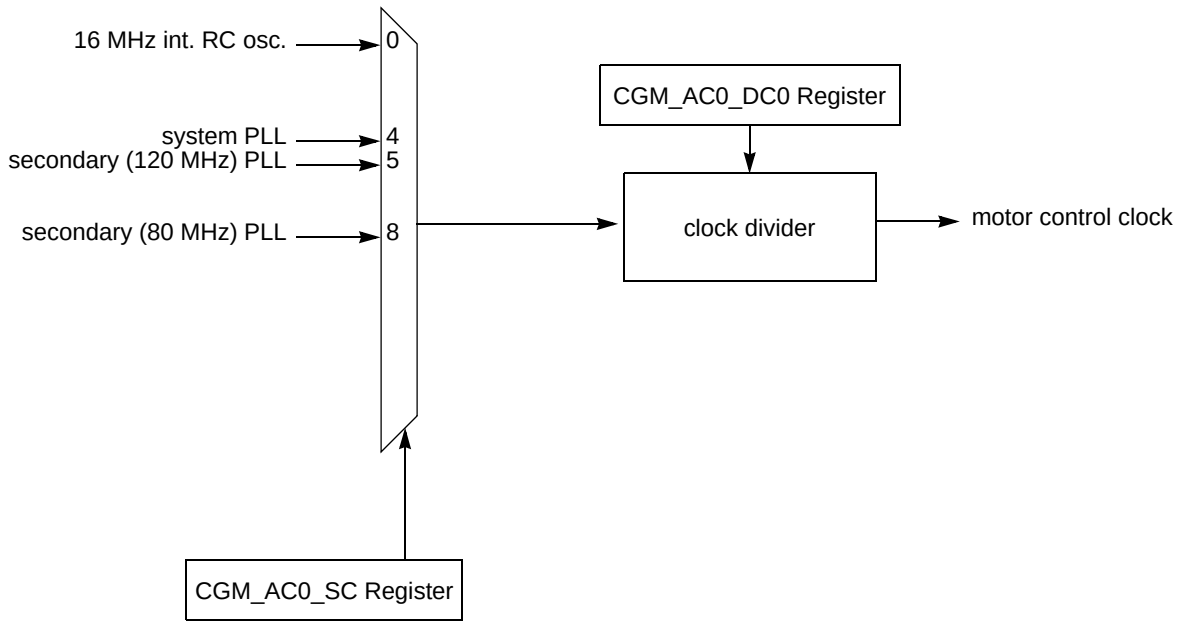


Figure 14-17. MC_CGM Auxiliary Clock 0 Generation Overview

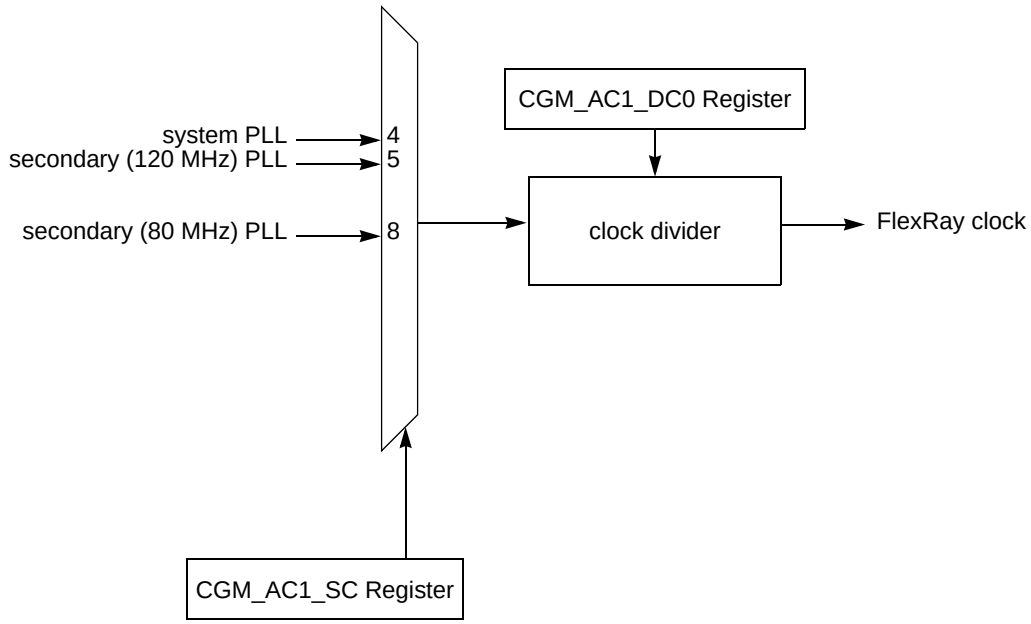


Figure 14-18. MC_CGM Auxiliary Clock 1 Generation Overview

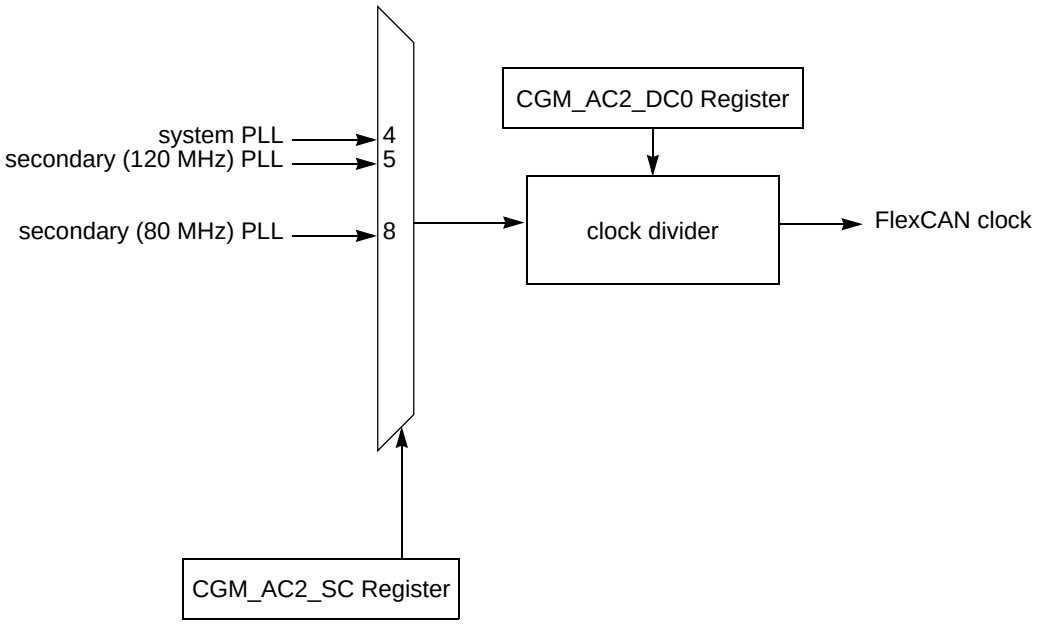


Figure 14-19. MC_CGM Auxiliary Clock 2 Generation Overview

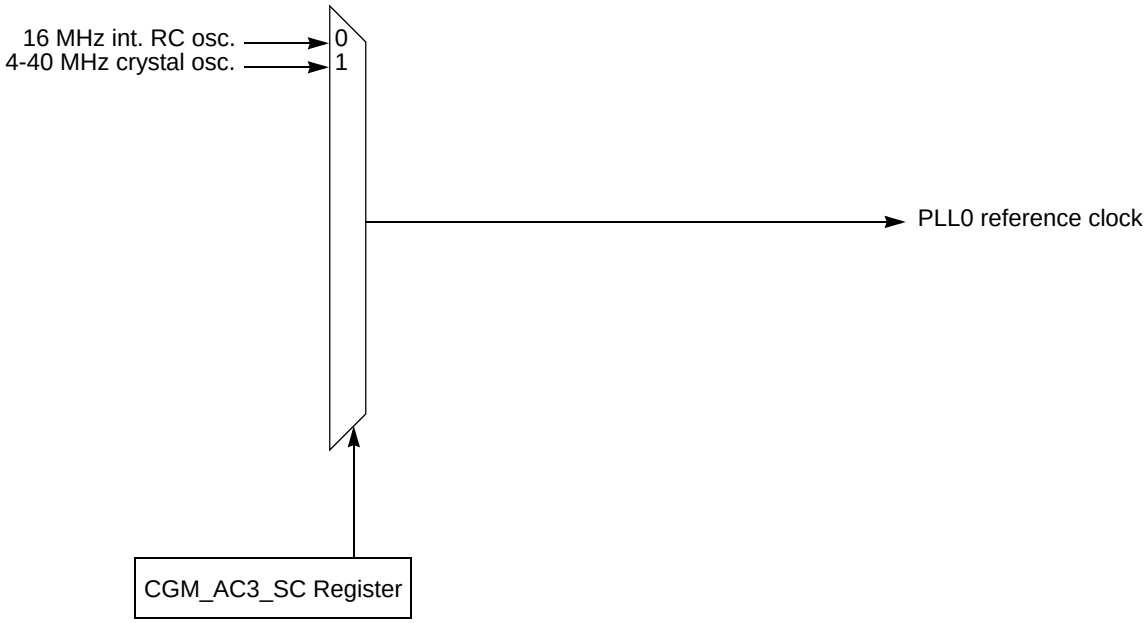


Figure 14-20. MC_CGM Auxiliary Clock 3 Generation Overview

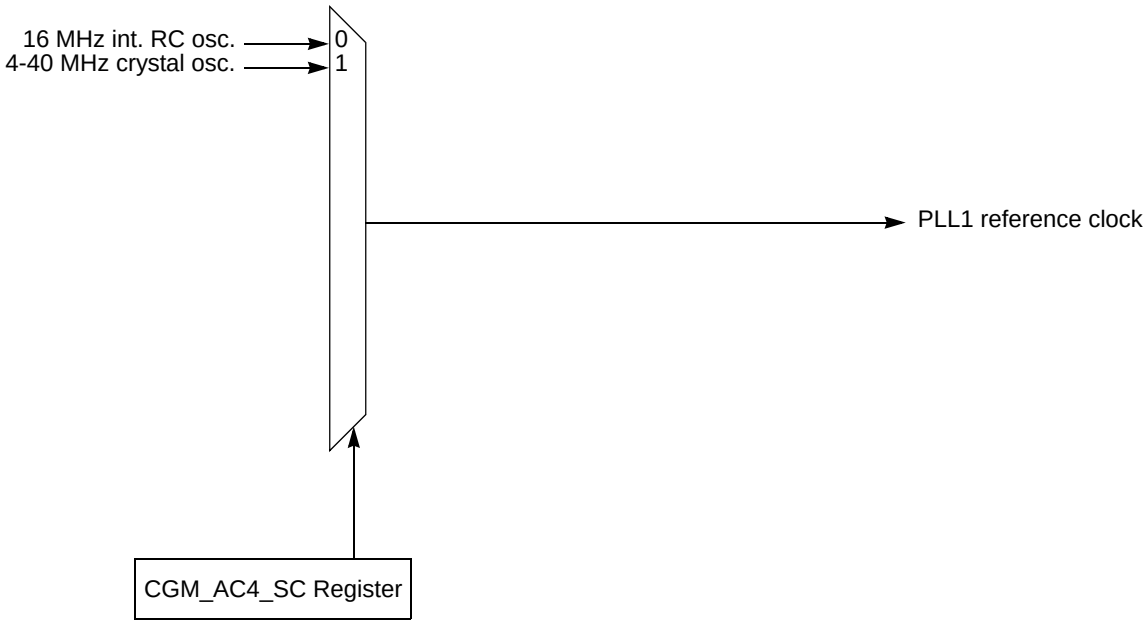


Figure 14-21. MC_CGM Auxiliary Clock 4 Generation Overview

14.4.1.5 Auxiliary Clock Dividers

The MC_CGM generates the following derived clocks:

- motor control clock—controlled by the CGM_AC0_DC0 register
- FlexRay clock—controlled by the CGM_AC1_DC0 register
- FlexCAN clock—controlled by the CGM_AC2_DC0 register

14.4.2 Dividers Functional Description

Dividers are used for the generation of divided system and peripheral clocks. The MC_CGM has the following control registers for built-in dividers:

- Section 14.3.1.5, System Clock Divider Configuration Register 0 (CGM_SC_DC0)”
- Section 14.3.1.6, System Clock Divider Configuration Register 1 (CGM_SC_DC1)”
- Section 14.3.1.8, Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)”
- Section 14.3.1.10, Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)”
- Section 14.3.1.12, Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)”

The reset value of all counters is ‘1’. If a divider has its DE bit in the respective configuration register set to ‘0’ (the divider is disabled), any value in its DIV*n* field is ignored.

14.4.3 Output Clock Multiplexing

The MC_CGM contains a multiplexing function for a number of clock sources, which can then be used as output clock sources. The selection is done via the CGM_OCDS_SC register.

14.4.4 Output Clock Division Selection

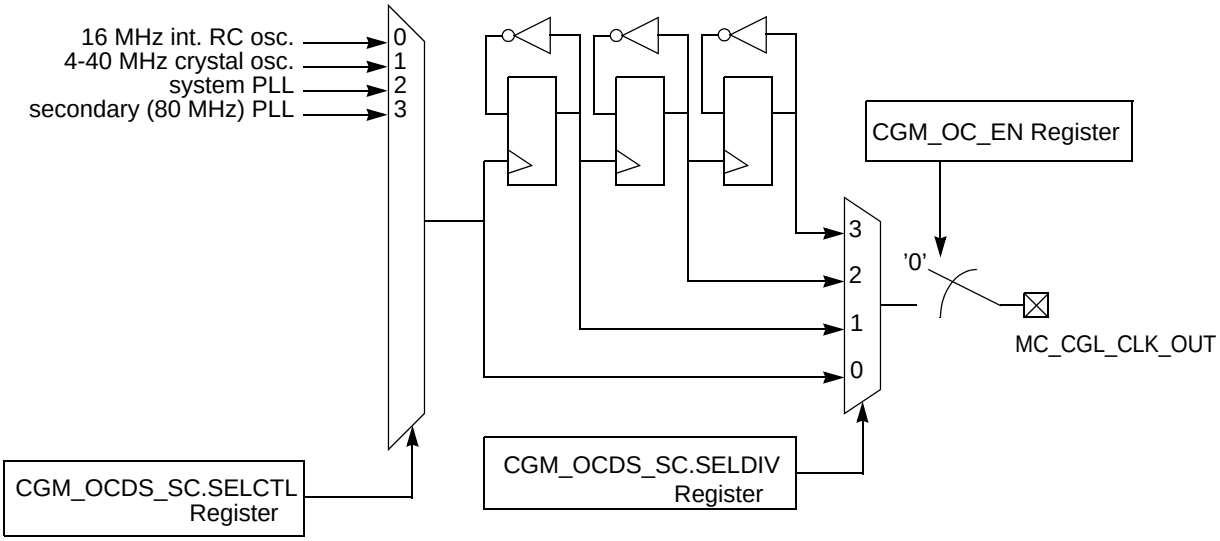


Figure 14-22. MC_CGM Output Clock Multiplexer and MC_CGL_CLK_OUT Generation

The MC_CGM provides the following output signals for the output clock generation:

- MC_CGL_CLK_OUT (see Figure 14-22). This signal is generated by using one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not guaranteed to be 50% duty cycle by the MC_CGM.

The MC_CGM also has an output clock enable register (see Section 14.3.1.2, Output Clock Enable Register (CGM_OC_EN)") that contains the output clock enable/disable control bit.

Chapter 15
Clock Monitor Unit (CMU)

15.1 Overview

The Clock Monitor Unit (CMU) serves three purposes:

- Crystal clock monitoring: monitor the external crystal oscillator clock, which must be greater than the internal RC clock divided by a division factor given by CMU_CSR[RCDIV]
- PLL clock monitoring: detect if the PLL leaves an upper or lower frequency boundary
- Frequency meter: measure the frequency of IRCOSC versus a known reference clock XOSC

The CMU forwards errors from the above events to the MC_CGM, MC_ME, and FCCU modules. These can then switch the MCU into a safe mode, generate a reset, or generate an interrupt.

The MPC5675K device has three CMUs. All three CMUs use the 16 MHz IRCOSC clock as the reference. Table 15-1 lists the three module names and the clocks that they monitor.

Table 15-1. CMU module summary

Module	Monitored clocks
CMU_0	• System clock • XOSC
CMU_1	Motor control clock
CMU_2	FlexRay clock

15.2 Main features

- RC oscillator frequency measurement
- External oscillator clock monitoring with respect to IRCOSC ÷ *n* clock
- PLL clock frequency monitoring with respect to IRCOSC ÷ 4 clock
- Event generation for various failures detected inside the monitoring unit

15.3 Functional description

The names of the clocks involved in this block have the following meaning:

- CK_XOSC: clock coming from the external crystal oscillator
- IRCOSC: clock coming from the low frequency internal RC oscillator
- CK_PLL: clock coming from the PLL
- FOSC: frequency of external crystal oscillator clock
- FRC: frequency of low frequency internal RC oscillator
- FPLL: frequency of FMPLL clock

15.3.1 Crystal clock monitor

If FOSC is smaller than FRC divided by 2^{RCDIV} bits of CMU_0_CSR and the CK_XOSC is ‘ON’ and stable as signaled by the MC_ME, then:

- CMU_ISR[OLRI] is set
- A failure event OLR is signaled to the MC_RGM and FCCU, which in turn can generate an interrupt

15.3.1.1 PLL clock monitor

The PLL clock CK_PLL frequency can be monitored by setting the CMU_CSR[CME] bit to 1. CK_PLL monitor starts as soon as CME bit is set. This monitor can be disabled at any time by writing CME bit to 0.

If CK_PLL frequency (FPLL) is greater than a reference value determined by the HFREF[11:0] bits of CMU_HFREFR_A and the CK_PLL is ‘ON’ and the PLL locked as signaled by the MC_ME then

- An event pending bit FHHI in CMU_CH_30FMPLLISR is set
- A failure event is signaled to the MC_RGM and Fault Collection Unit, which in turn can generate an interrupt

If FPLL is less than a reference clock frequency (FRC/4) and the CK_PLL is ‘ON’ and the PLL locked as signaled by the MC_ME, then:

- An event pending bit FLCI in CMU_ISR is set

If FPLL is less than a reference value determined by the LFREF[11:0] bits of CMU_LFREFR_A and the CK_PLL is ‘ON’ and the PLL is locked as signaled by the MC_ME, then:

- An event pending bit FLLI in CMU_ISR is set
- A failure event is signaled to the MC_RGM and Fault Collection Unit, which in turn can generate an interrupt

NOTE

The OSC and PLL monitors may produce false events when the OSC/PLL frequency is too close to the $RC/2^{RCDIV}$ frequency due to the limitation of comparing two clocks accurately.

15.3.1.2 Frequency meter

The frequency meter calibrates the internal RC oscillator (CK_IRC) using a known frequency.

NOTE

This value can then be stored in the flash memory so that application software can reuse it later on.

The reference clock is always the XOSC. A simple frequency meter returns a draft value of IRCOSC. The measurement starts when CMU_CSR[SFM] is set. The CMU_CSR register is writable only when register protection to the register is disabled. Hence, the user should take register protection into account if the frequency meter needs to be used.

The measurement duration is given by the CMU_MDR register in terms of IRCOSC clock cycles with a width of 20 bits. The SFM bit is cleared by the hardware after the frequency measurement is done and the count is loaded in the CMU_FDR. The frequency FRC can be derived from the value loaded in the CMU_FDR register as follows:

$$FRC = (FOSC \times MD) / n \qquad \textit{Eqn. 15-1}$$

where *n* is the value in CMU_FDR register and MD is the value in CMU_MDR.

When FXOSC > FIRC and MDR=0xFFFFF, the FDR overflows and the value stored cannot be relied upon.

At the IP level, the frequency relationship between the measured and reference frequency in not known and may vary across products.

Based on the frequency relationship in a particular product, the application must decide the appropriate value of MDR to avoid rollover of FDR.

MDR=1 is not allowed as the FDR counter counts from 0 to MDR–1.

15.4 Memory map and register description

The CMU registers are mapped through the MC_CGM (see the memory map in Chapter 14, Clock Generation Module (MC_CGM)"). The base address for each CMU is shown in Table 15-2.

Table 15-2. CMU base addresses

Module	Base address
CMU_0	0xC3FE_0100
CMU_1	0xC3FE_0120
CMU_2	0xC3FE_0140

The memory map of each CMU is shown in Table 15-3.

Table 15-3. CMU memory map

Offset from CMU_BASE	Register	Access[1]	Reset value[2]	Location
0x00	Control status register (CMU_CSR)	R/W	0x0000_0006	on page 396
0x04	Frequency display register (CMU_FDR)	R	0x0000_0000	on page 397
0x08	High-frequency reference register A (CMU_HFREFR_A)	R/W	0x0000_0FFF	on page 398
0x0C	Low-frequency reference register A (CMU_LFREFR_A)	R/W	0x0000_0000	on page 398
0x10	Interrupt status register (CMU_ISR)	R/W	0x0000_0000	on page 399
0x14	Reserved			

Table 15-3. CMU memory map (continued)

Offset from CMU_BASE	Register	Access[1]	Reset value[2]	Location
0x18	Measurement duration register (CMU_MDR)	R/W	0x0000_0000	on page 400
0x1C–0x1F	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

15.4.1 Control status register (CMU_CSR)

Address: Base + 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SFM	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CKSEL1		0	0	0	0	0	RCDIV		CME_A
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 15-1. Control status register (CMU_CSR)

Table 15-4. CMU_CSR field descriptions

Field	Description
SFM	Start frequency measure Set this bit to start a clock frequency measure. The bit is cleared by hardware when the measure is ready in the CMU_FDR register. Software cannot clear this bit. 0 Frequency measurement is completed or not yet started. 1 Frequency measurement is not completed.
CKSEL1	Clock selection This field selects the clock to be measured by the frequency meter. 00 IRCOSC is selected. 01 Reserved. 10 Reserved. 11 IRCOSC is selected.
RCDIV	RC clock division factor These bits specify the RC clock division factor. The output clock is IRCOSCfast divided by the factor 2^{RCDIV}. This output clock is compared with CK_XOSC for crystal clock monitor feature.The clock division coding is as follows. 00 Clock divided by 1 (No division). 01 Clock divided by 2. 10 Clock divided by 4. 11 Clock divided by 8.

Table 15-4. CMU_CSR field descriptions (continued)

Field	Description
CME_A	Clock monitor enable 0 Monitor is disabled. 1 Monitor is enabled.

NOTE

The CMU_CSR register is functional only on CMU_0.

15.4.2 Frequency display register (CMU_FDR)

Address: Base + 0x04 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	FD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-2. Frequency display register (CMU_FDR)

Table 15-5. CMU_FDR field descriptions

Field	Description
FD	Measured frequency bits This register displays the measured frequency f_{RC} with respect to f_{OSC}. The measured value is given by the following formula: $f_{RC} = (f_{OSC} \times MD) / n$ where *n* is the value in the CMU_FDR register.

NOTE

The CMU_FDR register is functional only on CMU_0.

15.4.3 High-frequency reference register A (CMU_HFREFR_A)

Address: Base + 0x08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	HFREF_A											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 15-3. High-frequency reference register A (CMU_HFREFR_A)

Table 15-6. CMU_HFREFR_A field descriptions

Field	Description
HFREF_A	High-frequency reference value These bits determine the high reference value for the PLL clock. The reference value is given by: (HFREF_A/16) × (f_{RC}/4).

NOTE

The CMU_HFREFR_A register is functional only on CMU_0.

15.4.4 Low-frequency reference register A (CMU_LFREFR_A)

Address: Base + 0x0C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF_A											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-4. Low-frequency reference register A (CMU_LFREFR_A)

Table 15-7. CMU_LFREFR_A field descriptions

Field	Description
LFREF_A	Low-frequency reference value These bits determine the low reference value for the PLL clock. The reference value is given by: (LFREF_A/16) × (f_{RC}/4).

NOTE

The CMU_LFREFR_A register is functional only on CMU_0.

15.4.5 Interrupt status register (CMU_ISR)

Address: Base + 0x10 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_A	FHHI_A	FLLI_A	OLRI
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-5. Interrupt status register (CMU_ISR)

Table 15-8. CMU_ISR field descriptions

Field	Description
FLCI_A	PLL clock frequency less than reference clock interrupt This bit is set by hardware when both of the following are true: • The PLL frequency becomes lower than the reference clock frequency (FRC/4) value • CK_FMPLL is ‘ON’ and the PLL locked as signaled by the MC_ME. It can be cleared by software by writing 1. 0 No FLC event. 1 FLC event is pending.
FHHI_A	FMPLL_0 Clock frequency higher than high reference interrupt This bit is set by hardware when both of the following are true: • The PLL frequency becomes higher than HFREF_A value • CK_FMPLL is ‘ON’ and the PLL locked as signaled by the MC_ME. It can be cleared by software by writing 1. 0 No FHH event. 1 FHH event is pending.
FLLI_A	FMPLL_0 Clock frequency less than low reference event This bit is set by hardware when both of the following are true: • The PLL frequency becomes lower than LFREF_A value • CK_FMPLL is ‘ON’ and the PLL locked as signaled by the MC_ME. It can be cleared by software by writing 1. 0 No FLL event. 1 FLL event is pending.

Table 15-8. CMU_ISR field descriptions (continued)

Field	Description
OLRI	Oscillator frequency less than RC frequency event This bit is set by hardware when both of the following are true: • The frequency of CK_XOSC is less than IRCOSCfast/2^{RCDIV} frequency • CK_XOSC is 'ON' and stable as signaled by the MC_ME. It can be cleared by software by writing 1. 0 No OLR event. 1 OLR event is pending.

NOTE

The CMU_ISR register is functional on all CMU instantiations.

15.4.6 Measurement duration register (CMU_MDR)

Address: Base + 0x18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	MD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-6. Measurement duration register (CMU_MDR)

Table 15-9. CMU_MDR field descriptions

Field	Description
MD	Measurement duration bits This register displays the measured duration in term of IRCOSC clock cycles. This value is loaded in the frequency meter down-counter. When CMU_CSR[SFM] = 1, the down-counter starts counting.

NOTE

The CMU_MDR register is functional only on CMU_0.

15.5 CMU integration

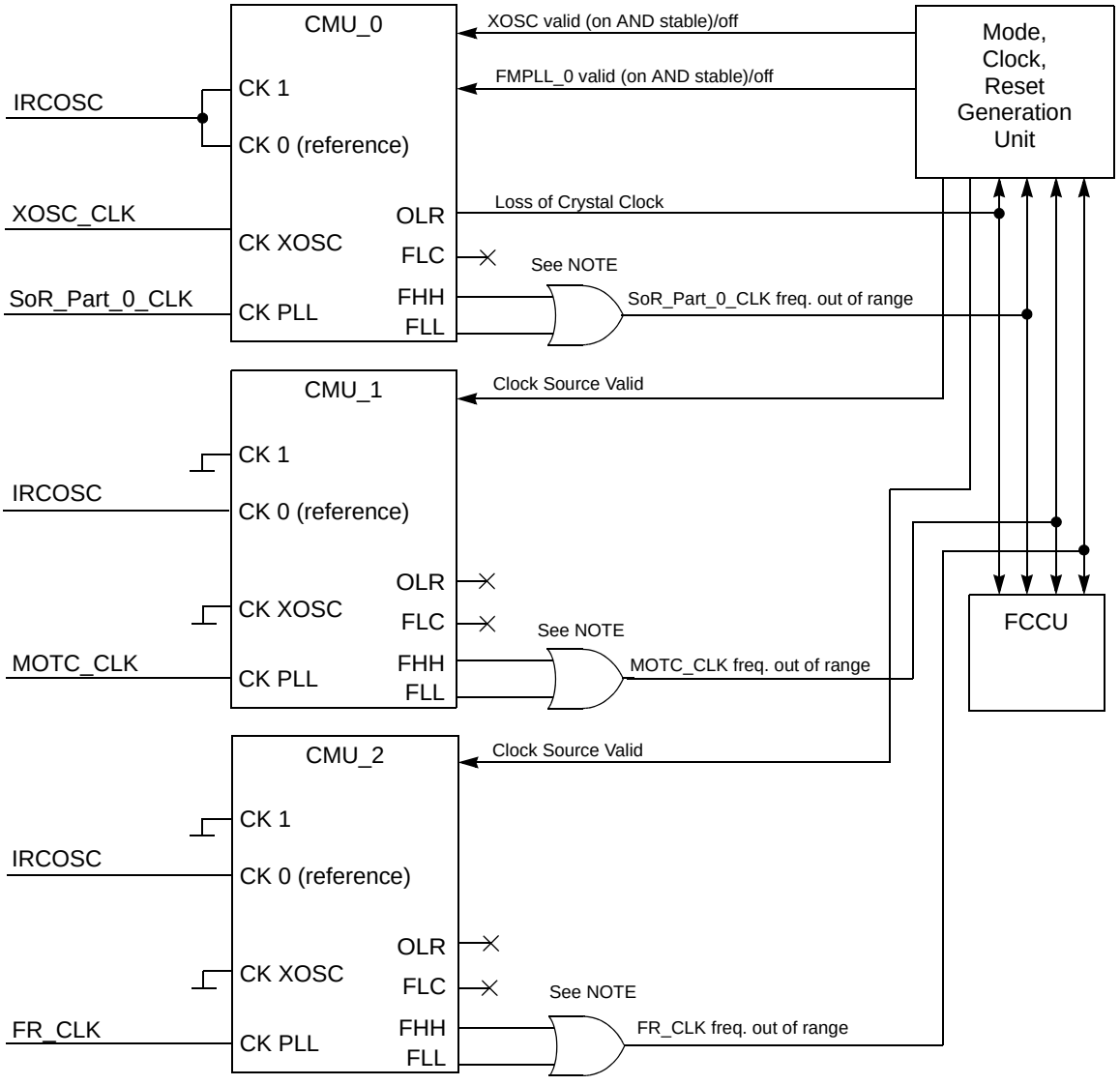
All three CMUs use the IRCOSC as their reference clock input.

CMU_0 monitors the SoR_Part_0_CLK clock signal, which is used as the clock signal for the Sphere of Replication Part 0. The clock source selected by the system clock selector can change due to software- or hardware-initiated mode changes. In case of a software-initiated clock source change or a change of the source clock frequency, software must adjust the CMU_0 frequency surveillance range as required for proper function.

CMU_1 monitors the MOTC_CLK clock signal, which is used as the clock signal for the motor control-related peripherals.

CMU_2 monitors the FR_CLK clock signal, which is used as the clock signal for the protocol engine of the FlexRay module.

The frequency above limit (FHH) event and the frequency below limit (FLL) event of each CMU are logically combined to generate a signal that tells whether the frequency of the supervised signal is within or outside the limits before feeding the result into the CGM, RGM, or ME modules and the FCCU shown in Figure 15-7.



NOTE:

The exact logical combination depends upon polarities of the CMU event signals and the expected polarity of the input signals to the CMU.

Figure 15-7. CMU integration

During the device reset phase, all CMUs are disabled, with the exception of the XOSC monitor in CMU0, which is always enabled. This is dependent upon the status of XOSC (if valid). The default state of the CMUs after reset is disabled as the application software needs to program the valid frequency range first before enabling the CMUs.

Chapter 16
Cyclic Redundancy Checker (CRC) Unit

16.1 Introduction

The Cyclic Redundancy Checker (CRC) computing unit is dedicated to the computation of CRC off-loading the CPU. The number of contexts is a static parameter. Each context has a separate CRC computation engine in order to allow the concurrent computation of the CRC of multiple data streams. The CRC computation is performed at speed without wait states insertion. Bit-swap and bit-inversion operations can be applied on the final CRC signature. Each context can be configured with one of two hard-wired polynomials, normally used for most of the standard communication protocols. The data stream supports multiple data width formats (byte/half-word/word).

16.2 Main features

- Three contexts (static parameter) for the concurrent CRC computation
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- Two hard-wired polynomials (CRC-32 ethernet and CRC-16-CCITT)
- Support for byte/half-word/word width of the input data stream

16.2.1 Standard features

- CRC-16-CCITT
- CRC-32 ethernet

16.3 Block diagram

The top level diagram of the CRC module is given in Figure 16-1. For MPC5675K, the number of contexts (CRC_CNTX_NUM) is three.

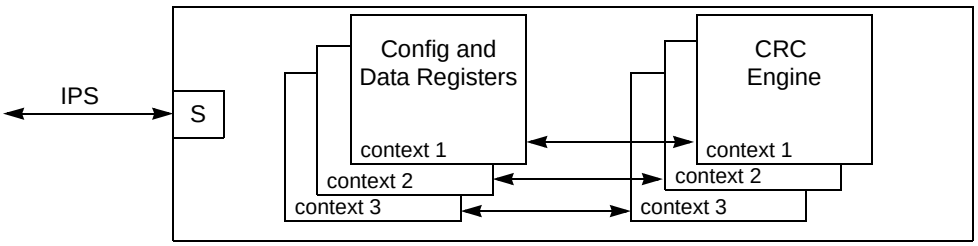


Figure 16-1. CRC top level diagram

16.4 Signal description

The CRC unit does not generate any external signals.

16.5 Memory map and registers

16.5.1 Register description

Table 16-1 shows the base addresses for the CRCU modules. Table 16-2 shows the memory map for the CRC registers.

Table 16-1. CRCU module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM) Decoupled Parallel Mode (DPM)	CRCU_0	0xFFE6_8000
	CRCU_1	0xC3E6_0000

Table 16-2. CRC unit memory map

Offset from CRCU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	CRC Configuration Register Context 1 (CRC_CFG)	R/W	0x0000_0000	on page 405
0x0004	CRC Input Register Context 1 (CRC_INP)	R/W	0x0000_0000	on page 406
0x0008	CRC Current Status Register Context 1 (CRC_CSTAT)	R/W	0xFFFF_FFFF	on page 406
0x000C	CRC Output Register Context 1 (CRC_OUTP)	R	0xFFFF_FFFF	on page 407
0x0010	CRC Configuration Register Context 2 (CRC_CFG)	R/W	0x0000_0004	on page 405
0x0014	CRC Input Register Context 2 (CRC_INP)	R/W	0x0000_0000	on page 406
0x0018	CRC Current Status Register Context 2 (CRC_CSTAT)	R/W	0xFFFF_FFFF	on page 406
0x001C	CRC Output Register Context 2 (CRC_OUTP)	R	0xFFFF_FFFF	on page 407
0x0020	CRC Configuration Register Context 3 (CRC_CFG)	R/W	0x0000_0000	on page 405
0x0024	CRC Input Register Context 3 (CRC_INP)	R/W	0x0000_0000	on page 406
0x0028	CRC Current Status Register Context 3 (CRC_CSTAT)	R/W	0xFFFF_FFFF	on page 406
0x002C	CRC Output Register Context 3 (CRC_OUTP)	R	0xFFFF_FFFF	on page 407
0x0030–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

16.5.1.1 CRC Configuration Register (CRC_CFG)

Address: Base + 0x0000 (CRC_CFG1)
Base + 0x0010 (CRC_CFG2)
Base + 0x0020 (CRC_CFG3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	POLYG	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	—[1]	0	0

Figure 16-2. CRC Configuration Register (CRC_CFG)

1 For CRC_CFG1 and CRC_CFG3, this bit equals 1 at reset. For CRC_CFG2, this bit equals 0 at reset.

Table 16-3. CRC_CFG field descriptions

Field	Description
POLYG	Polynomial selection 0 CRC-CCITT polynomial. 1 CRC-32 polynomial. This bit can be read and written by the software. This bit can be written only during the configuration phase.
SWAP	SWAP selection 0 No swap selection applied on the CRC_OUTP content 1 Swap selection (MSB → LSB, LSB→ MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial, the swap operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.
INV	INV selection 0 No inversion selection applied on the CRC_OUTP content. 1 Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.

16.5.1.2 CRC Input Register (CRC_INP)

Address: Base + 0x0004 (CRC_INP1)
Base + 0x0014 (CRC_INP2)
Base + 0x0024 (CRC_INP3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-3. CRC Input Register (CRC_INP)

Table 16-4. CRC_INP field descriptions

Field	Description
INP	Input data for the CRC computation The INP register can be written at byte, half-word (high and low), or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by the software.

16.5.1.3 CRC Current Status Register (CRC_CSTAT)

Address: Base + 0x0008 (CRC_CTSAT1)
Base + 0x0018 (CRC_CSTAT2)
Base + 0x0028 (CRC_CSTAT3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 16-4. CRC Current Status Register (CRC_CSTAT)

Table 16-5. CRC_CSTAT field descriptions

Field	Description
CSTAT	Status of the CRC signature The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 LSB bits are significative. The 16 MSB bits are tied at 0b during the computation. The CSTAT register can be written at byte, half-word, or word. This register can be read and written by the software. This register can be written only during the configuration phase.

16.5.1.4 CRC Output Register (CRC_OUTP)

Address: Base + 0x000C (CRC_OUTP1)
Base + 0x001C (CRC_OUTP2)
Base + 0x002C (CRC_OUTP3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 16-5. CRC Output Register (CRC_OUTP)

Table 16-6. CRC_CSTAT field descriptions

Field	Description
OUTP	Final CRC signature The OUTP register includes the final signature corresponding to the CRC_CSTAT register value eventually swapped and inverted. In case of CRC-CCITT polynomial only the16 LSB bits are significative. The 16 MSB bits are tied at 0b during the computation. This register can be read by the software.

16.6 Functional description

The CRC module supports the CRC computation for each context. Each context has its own complete set of registers including the CRC engine. The data flow of each context can be interleaved. The data stream can be structured as a sequence of byte, half-words, or words. The input data sequence is provided, eventually mixing the data formats (byte, half-word, word), writing to the input data register (CRC_INP).

The data stream is generally executed by N concurrent DMA data transfers (mem2mem) where N is less than or equal to the number of contexts (3).

Two standard generator polynomials are given in Equation 16-1 and Equation 16-2 for the CRC computation of each context.

$$X^{16} + X^{12} + X^5 + 1$$

CRC-CCITT (x.25 protocol) ***Eqn. 16-1***

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

CRC-32 (ethernet protocol) ***Eqn. 16-2***

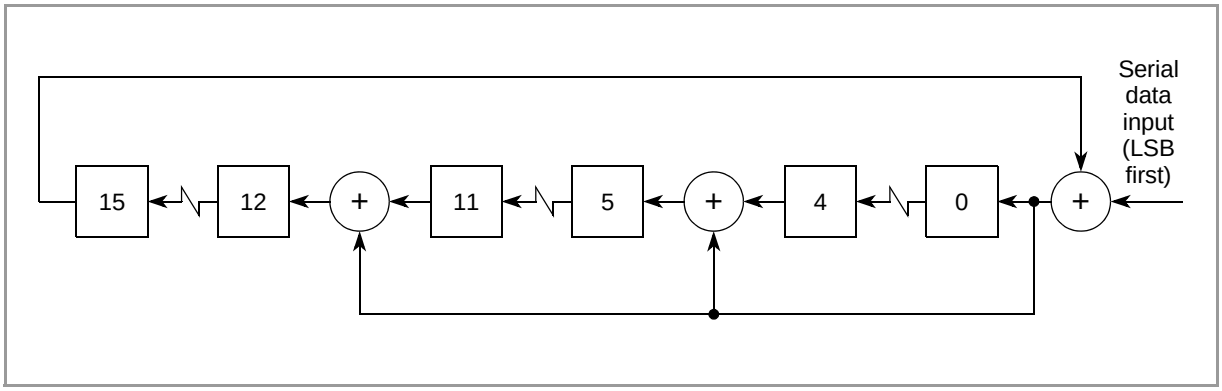


Figure 16-6. CRC-CCITT engine concept scheme

The initial seed value of the CRC can be programmed initializing the CRC_CSTAT register. The concept scheme (serial data loading) of the CRC engine is given in Figure 16-6 for the CRC-CCITT. The design implementation executes the CRC computation in a single clock cycle (parallel data loading). A pipeline scheme has been adopted to de-couple the IPS bus interface from the CRC engine in order to allow the computation of the CRC at speed (zero wait states).

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (for example, SPI) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register) before transmitting the CRC.

The usage of the CRC module is summarized in the flow-chart given in Figure 16-7.

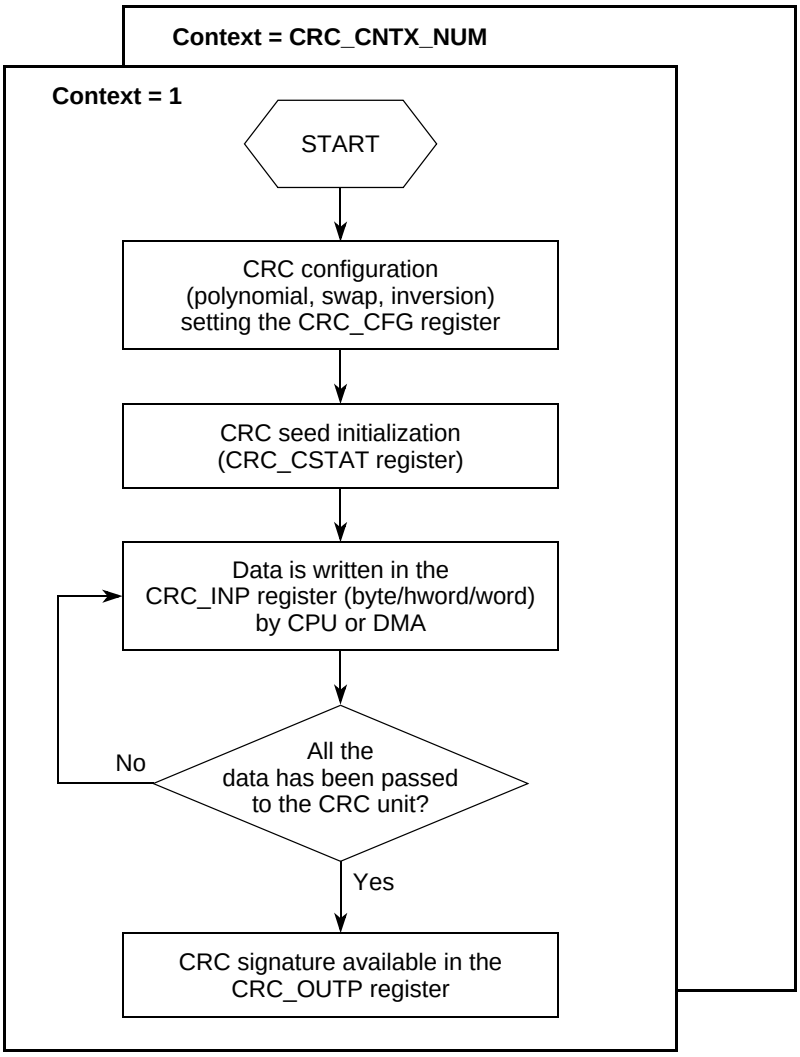


Figure 16-7. CRC computation flow

16.7 Use cases and limitations

Two main use cases are considered:

- Calculation of the CRC of the configuration registers within the respective process safety time or fault tolerant time interval
- Calculation of the CRC on the incoming/outgoing frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral

The signature of the configuration registers is computed in a correct way only if these registers do not contain any status bit. Assuming that the DMA engine has N channels (greater or equal to the number of contexts) configurable for these types of data transfer — mem2mem, periph2mem, and mem2periph — then the following sequence, as given in Figure 16-8, is applied to manage the transmission data flow:

- DMA/CRC module configuration (context *x*, channel *x*) by CPU

- Payload transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase 1) by DMA (mem2mem data transfer, channel *x*)
- CRC signature copy from the CRC module (CRC_OUTP register) to the MEM (phase 2) by CPU
- Data block (payload + CRC) transfer from the MEM to the PERIPH module (for example, SPI TX FIFO) (phase 3) by DMA (mem2periph data transfer, channel *x*)

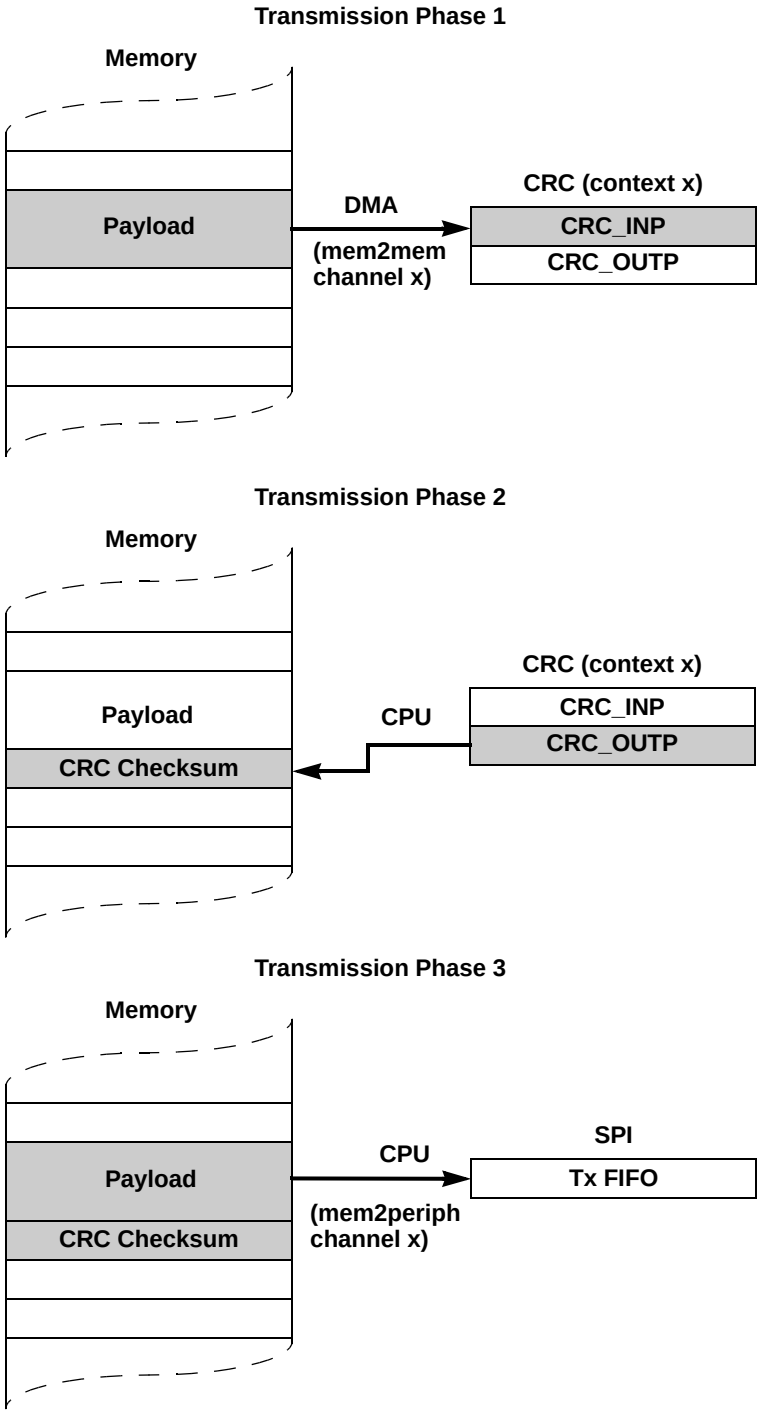


Figure 16-8. DMA-CRC Transmission Sequence

The following sequence, as given in Figure 16-9, is applied to manage the reception data flow:

- DMA/CRC module configuration (context *x*, channel *x*) by CPU
- Data block (payload + CRC) transfer from the PERIPH (for example, SPI RX FIFO) module to the MEM (phase 1) by DMA (periph2mem data transfer, channel *x*)
- Data block transfer (payload + CRC) transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase 2) by DMA (mem2mem data transfer, channel *x*)
- CRC signature check from the CRC module (CRC_OUTP register) by CPU (phase 3)

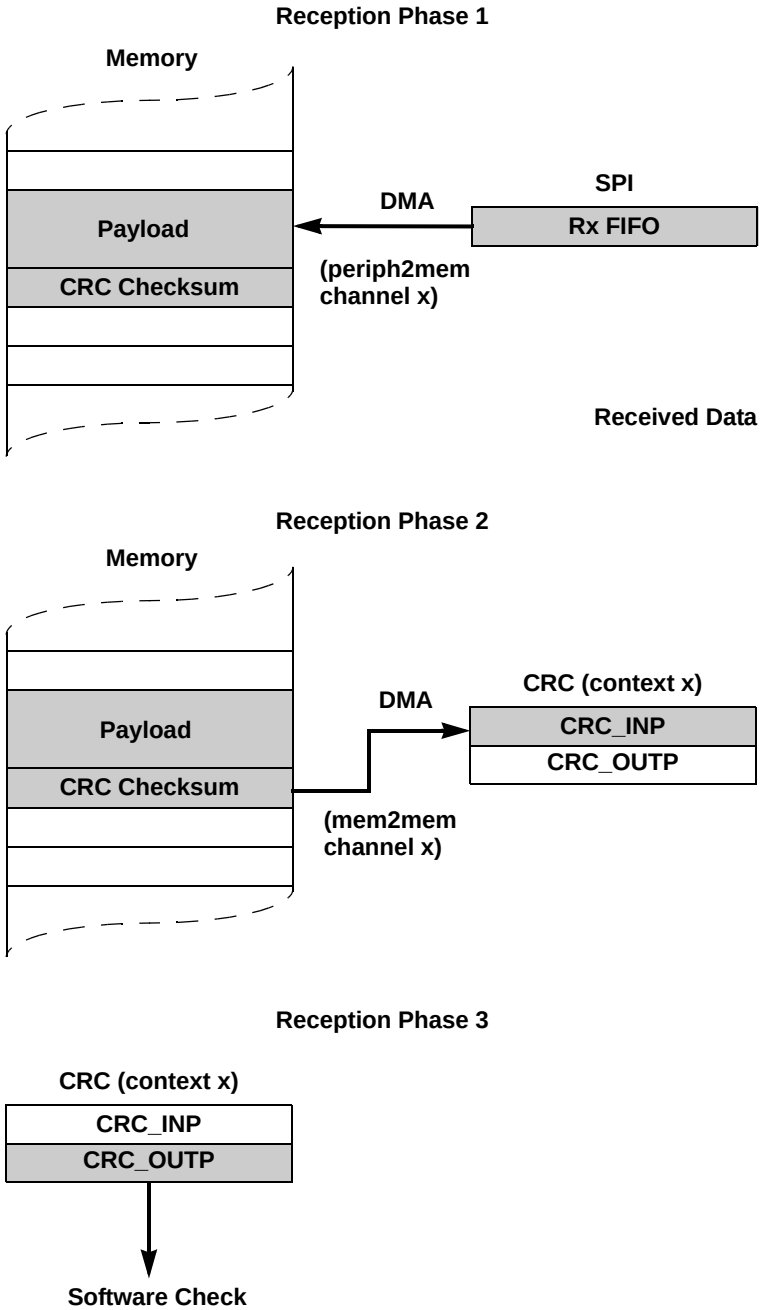


Figure 16-9. DMA-CRC Reception Sequence

This page is intentionally left blank.

Chapter 17
Coherency Unit (CU)

17.1 Introduction

The Coherency Unit (CU) is a hardware mechanism for maintaining memory coherency in a dual-core system with independent, tightly coupled local cache memories. By monitoring AHB transactions originating from the various masters, the CU monitors AHB bus traffic for updates to system memory spaces that are shared between the cores, and invokes coherency snoop requests.

17.2 Features

The following list summarizes the key features of the CU:

- Supports as many as four masters to be monitored
  - Two cores and two non-processor masters
  - Location is immediately downstream of each master, before the crossbar switch
- Supports as many as two snoop request streams, one for each core in a dual-core system
- Each snoop request stream is supported by a 16-deep snoop queue
- Overflow protection by issuing request to stall subsequent global writes
- Optimized for bursts, such that global writes to the same cache line collapse into a single snoop request
- Program-visible registers for capturing information on queue status and error results
- Optional interrupt generation on snoop requests that terminate with error

17.2.1 Block diagram

Figure 17-1 shows a typical device with a CU implemented.

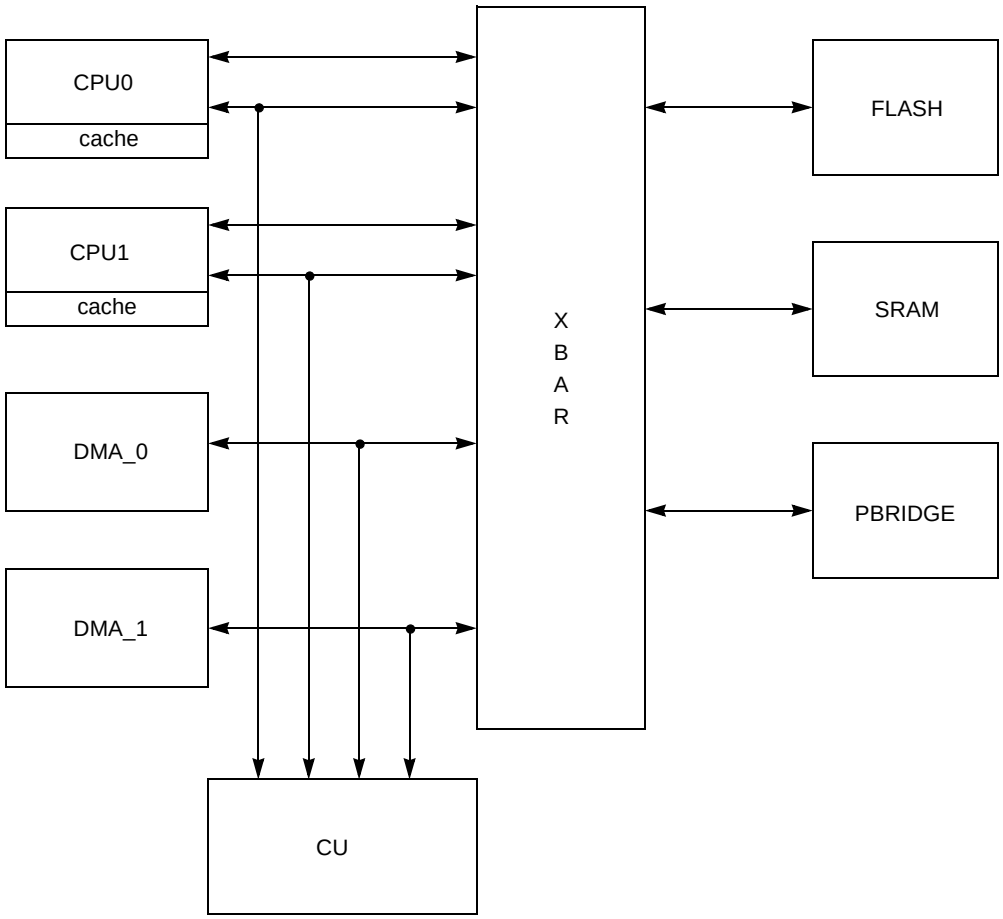


Figure 17-1. Platform with CU

Figure 17-2 shows a generic dual-core platform with relative positioning of the CU module. The CU module sits between the masters and crossbar switch.

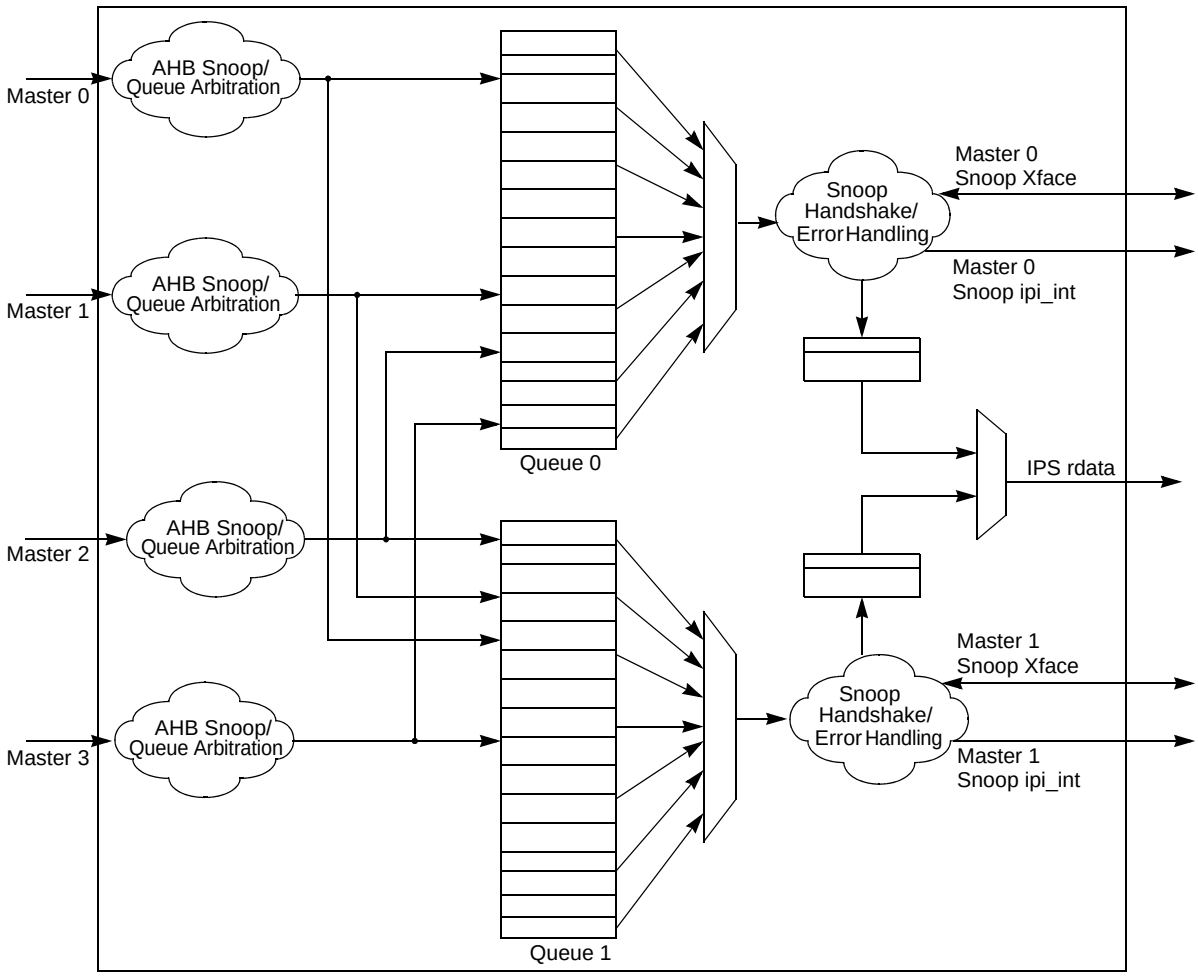


Figure 17-2. CU block diagram

Figure 17-3 shows the functional data path for the CU. As many as four masters can be monitored for global writes, where Master 0 and Master 1 are processor masters and Master 2 and Master 3 are non-processor masters. Upon detection of a global write, the CU determines which processor (Master 0 or Master 1) needs to be snooped. The snoop request is then placed in the appropriate queue and eventually dispatched to the processor.

17.2.2 ACP_CU interface signals

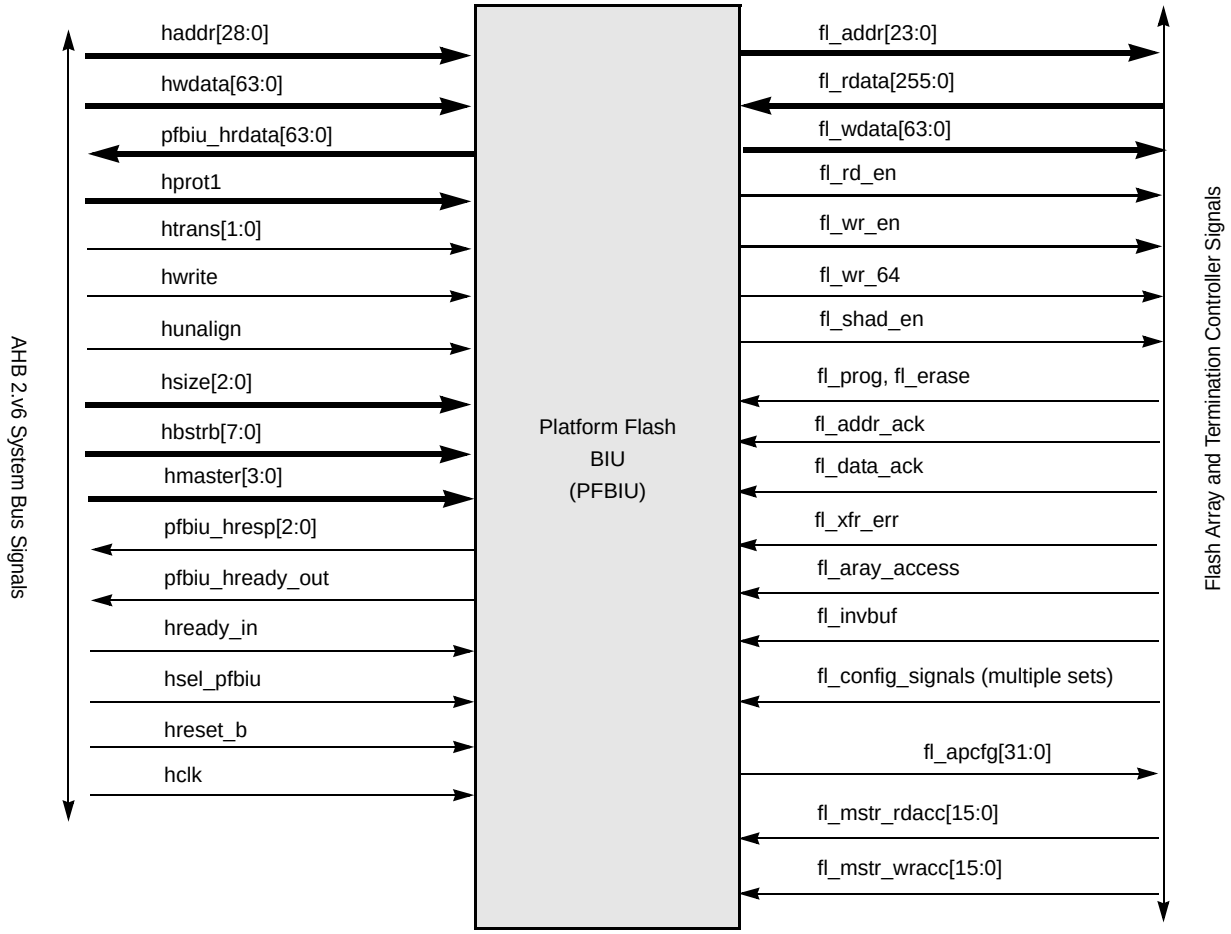


Figure 17-3. ACP_CU interface

Table 17-1. ACP_CU signals

Signal	Type	Description
System inputs		
clk	Input	AHB reference clock
reset	Input	AHB reset
Master interface		
m{0,1,2,3}_hmaster[3:0]	Input	Master ID
m{0,1,2,3}_haddr[28:0]	Input	Address bus
m{0,1,2,3}_htrans[1:0]	Input	Transfer type
m{0,1,2,3}_hwrite	Input	Transfer direction
m{0,1,2,3}_hprot[0]	Input	Inst/Data attribute
m{0,1,2,3}_hready	Input	Transfer done in

Table 17-1. ACP_CU signals (continued)

Signal	Type	Description
m{0,1,2,3}_gbl	Input	Write is marked as global
m{0,1,2,3}_clken	Input	Clock Enable for support of masters running in various clock domains
Snoop handshake signals		
m{0,1}_snp_req	Output	Snoop request
m{0,1}_snp_addr[31:0]	Output	Snoop request addr
m{0,1}_snp_id[3:0]	Output	Snoop request ID
m{0,1}_snp_cmd[1:0]	Output	Snoop request command
m{0,1}_snp_ack	Input	Snoop complete acknowledge
m{0,1}_snp_ack_id[3:0]	Input	Snoop complete acknowledge ID
Miscellaneous CU signals		
m{0,1}_ipi_int	Output	Optional interrupt on snoop request error
m{0,1}_snp_synch	Input	Command indicator to mark snoop requests as synch instructions via m{0,1}_snp_cmd[1:0] 0 Invalidate 1 Synch
stall_gbl_wr	Output	Stall global writes. Sent to all masters being monitored, indicating subsequent writes should be stalled until the queue is no longer full.
m{0,1}_queue_idle	Output	Status flag indicating Master *n* queue is empty

17.3 Memory map

The CU provides an IPS programming model mapped to an on-platform 16 KB space. The programming model is partitioned into groups: control/status registers, the data structure containing the contents of the snoop queues for Core_0 and Core_1.

The programming model can be referenced only using 32-bit (word) accesses. Writes to the configuration fields can be referenced only in supervisor mode. Attempted references using different access sizes, accesses to undefined or reserved addresses, or accesses with a non-supported access type result in IPS error termination.

Table 17-2. CU memory map

Offset from CU_BASE (0xFFF5_0000)	Register	Access[1]	Reset value[2]	Location
0x0000	PCU Configuration and Error Status Register (PCU_CESR)	R/W	0x0300_0000	on page 422
0x0004–0x000F	Reserved			

Table 17-2. CU memory map (continued)

Offset from CU_BASE (0xFFF5_0000)	Register	Access[1]	Reset value[2]	Location
0x0010	PCU Error Address Register, Core_0 (PCU_EAR0)	R	0xUUUU_UUUU[3]	on page 426
0x0014	PCU Error Detail Register, Core_0 (PCU_EDR0)	R	0xUUUU_UUUU[3]	on page 426
0x0018–0x001F	Reserved			
0x0020	PCU Error Address Register, Core_1 (PCU_EAR1)	R	0xUUUU_UUUU[3]	on page 426
0x0024	PCU Error Detail Register, Core_1 (PCU_EDR1)	R	0xUUUU_UUUU[3]	on page 422
0x0028–0x002F	Reserved			
0x0030	PCU Interrupt Register, Core_0 (PCU_IR0)	R/W	0xUUUU_UUUU[3]	on page 428
0x0034	PCU Interrupt Register, Core_1 (PCU_IR1)	R/W	0xUUUU_UUUU[3]	on page 428
0x0038–0x00FF	Reserved			
0x0100	PCU Core_0, Entry 0 Address Register (PCU_CP0E0_ADDR)	R	0x0000_0000	on page 428
0x0104	PCU Core_0, Entry 0 Attributes Register (PCU_CP0E0_ATTR)	R	0x0000_0000	on page 429
0x0108	PCU Core_0, Entry 1 Address Register (PCU_CP0E1_ADDR)	R	0x0000_0000	on page 428
0x010C	PCU Core_0, Entry 1 Attributes Register (PCU_CP0E1_ATTR)	R	0x0000_0000	on page 429
0x0110	PCU Core_0, Entry 2 Address Register (PCU_CP0E2_ADDR)	R	0x0000_0000	on page 428
0x0114	PCU Core_0, Entry 2 Attributes Register (PCU_CP0E2_ATTR)	R	0x0000_0000	on page 429
0x0118	PCU Core_0, Entry 3 Address Register (PCU_CP0E3_ADDR)	R	0x0000_0000	on page 428
0x011C	PCU Core_0, Entry 3 Attributes Register (PCU_CP0E3_ATTR)	R	0x0000_0000	on page 429
0x0120	PCU Core_0, Entry 4 Address Register (PCU_CP0E4_ADDR)	R	0x0000_0000	on page 428
0x0124	PCU Core_0, Entry 4 Attributes Register (PCU_CP0E4_ATTR)	R	0x0000_0000	on page 429
0x0128	PCU Core_0, Entry 5 Address Register (PCU_CP0E5_ADDR)	R	0x0000_0000	on page 428
0x012C	PCU Core_0, Entry 5 Attributes Register (PCU_CP0E5_ATTR)	R	0x0000_0000	on page 429
0x0130	PCU Core_0, Entry 6 Address Register (PCU_CP0E6_ADDR)	R	0x0000_0000	on page 428
0x0134	PCU Core_0, Entry 6 Attributes Register (PCU_CP0E6_ATTR)	R	0x0000_0000	on page 429

Table 17-2. CU memory map (continued)

Offset from CU_BASE (0xFFF5_0000)	Register	Access[1]	Reset value[2]	Location
0x0138	PCU Core_0, Entry 7 Address Register (PCU_CP0E7_ADDR)	R	0x0000_0000	on page 428
0x013C	PCU Core_0, Entry 7 Attributes Register (PCU_CP0E7_ATTR)	R	0x0000_0000	on page 429
0x0140	PCU Core_0, Entry 8 Address Register (PCU_CP0E8_ADDR)	R	0x0000_0000	on page 428
0x0144	PCU Core_0, Entry 8 Attributes Register (PCU_CP0E8_ATTR)	R	0x0000_0000	on page 429
0x0148	PCU Core_0, Entry 9 Address Register (PCU_CP0E9_ADDR)	R	0x0000_0000	on page 428
0x014C	PCU Core_0, Entry 9 Attributes Register (PCU_CP0E9_ATTR)	R	0x0000_0000	on page 429
0x0150	PCU Core_0, Entry 10 Address Register (PCU_CP0E10_ADDR)	R	0x0000_0000	on page 428
0x0154	PCU Core_0, Entry 10 Attributes Register (PCU_CP0E10_ATTR)	R	0x0000_0000	on page 429
0x0158	PCU Core_0, Entry 11 Address Register (PCU_CP0E11_ADDR)	R	0x0000_0000	on page 428
0x015C	PCU Core_0, Entry 11 Attributes Register (PCU_CP0E11_ATTR)	R	0x0000_0000	on page 429
0x0160	PCU Core_0, Entry 12 Address Register (PCU_CP0E12_ADDR)	R	0x0000_0000	on page 428
0x0164	PCU Core_0, Entry 12 Attributes Register (PCU_CP0E12_ATTR)	R	0x0000_0000	on page 429
0x0168	PCU Core_0, Entry 13 Address Register (PCU_CP0E13_ADDR)	R	0x0000_0000	on page 428
0x016C	PCU Core_0, Entry 13 Attributes Register (PCU_CP0E13_ATTR)	R	0x0000_0000	on page 429
0x0170	PCU Core_0, Entry 14 Address Register (PCU_CP0E14_ADDR)	R	0x0000_0000	on page 428
0x0174	PCU Core_0, Entry 14 Attributes Register (PCU_CP0E14_ATTR)	R	0x0000_0000	on page 429
0x0178	PCU Core_0, Entry 15 Address Register (PCU_CP0E15_ADDR)	R	0x0000_0000	on page 428
0x017C	PCU Core_0, Entry 15 Attributes Register (PCU_CP0E15_ATTR)	R	0x0000_0000	on page 429
0x0180–0x01FF	Reserved			
0x0200	PCU Core_1, Entry 0 Address Register (PCU_CP1E0_ADDR)	R	0x0000_0000	on page 428

Table 17-2. CU memory map (continued)

Offset from CU_BASE (0xFFF5_0000)	Register	Access[1]	Reset value[2]	Location
0x0204	PCU Core_1, Entry 0 Attributes Register (PCU_CP1E0_ATTR)	R	0x0000_0000	on page 429
0x0208	PCU Core_1, Entry 1 Address Register (PCU_CP1E1_ADDR)	R	0x0000_0000	on page 428
0x020C	PCU Core_1, Entry 1 Attributes Register (PCU_CP1E1_ATTR)	R	0x0000_0000	on page 429
0x0210	PCU Core_1, Entry 2 Address Register (PCU_CP1E2_ADDR)	R	0x0000_0000	on page 428
0x0214	PCU Core_1, Entry 2 Attributes Register (PCU_CP1E2_ATTR)	R	0x0000_0000	on page 429
0x0218	PCU Core_1, Entry 3 Address Register (PCU_CP1E3_ADDR)	R	0x0000_0000	on page 428
0x021C	PCU Core_1, Entry 3 Attributes Register (PCU_CP1E3_ATTR)	R	0x0000_0000	on page 429
0x0220	PCU Core_1, Entry 4 Address Register (PCU_CP1E4_ADDR)	R	0x0000_0000	on page 428
0x0224	PCU Core_1, Entry 4 Attributes Register (PCU_CP1E4_ATTR)	R	0x0000_0000	on page 429
0x0228	PCU Core_1, Entry 5 Address Register (PCU_CP1E5_ADDR)	R	0x0000_0000	on page 428
0x022C	PCU Core_1, Entry 5 Attributes Register (PCU_CP1E5_ATTR)	R	0x0000_0000	on page 429
0x0230	PCU Core_1, Entry 6 Address Register (PCU_CP1E6_ADDR)	R	0x0000_0000	on page 428
0x0234	PCU Core_1, Entry 6 Attributes Register (PCU_CP1E6_ATTR)	R	0x0000_0000	on page 429
0x0238	PCU Core_1, Entry 7 Address Register (PCU_CP1E7_ADDR)	R	0x0000_0000	on page 428
0x023C	PCU Core_1, Entry 7 Attributes Register (PCU_CP1E7_ATTR)	R	0x0000_0000	on page 429
0x0240	PCU Core_1, Entry 8 Address Register (PCU_CP1E8_ADDR)	R	0x0000_0000	on page 428
0x0244	PCU Core_1, Entry 8 Attributes Register (PCU_CP1E8_ATTR)	R	0x0000_0000	on page 429
0x0248	PCU Core_1, Entry 9 Address Register (PCU_CP1E9_ADDR)	R	0x0000_0000	on page 428
0x024C	PCU Core_1, Entry 9 Attributes Register (PCU_CP1E9_ATTR)	R	0x0000_0000	on page 429

Table 17-2. CU memory map (continued)

Offset from CU_BASE (0xFFF5_0000)	Register	Access[1]	Reset value[2]	Location
0x0250	PCU Core_1, Entry 10 Address Register (PCU_CP1E10_ADDR)	R	0x0000_0000	on page 428
0x0254	PCU Core_1, Entry 10 Attributes Register (PCU_CP1E10_ATTR)	R	0x0000_0000	on page 429
0x0258	PCU Core_1, Entry 11 Address Register (PCU_CP1E11_ADDR)	R	0x0000_0000	on page 428
0x025C	PCU Core_1, Entry 11 Attributes Register (PCU_CP1E11_ATTR)	R	0x0000_0000	on page 429
0x0260	PCU Core_1, Entry 12 Address Register (PCU_CP1E12_ADDR)	R	0x0000_0000	on page 428
0x0264	PCU Core_1, Entry 12 Attributes Register (PCU_CP1E12_ATTR)	R	0x0000_0000	on page 429
0x0268	PCU Core_1, Entry 13 Address Register (PCU_CP1E13_ADDR)	R	0x0000_0000	on page 428
0x026C	PCU Core_1, Entry 13 Attributes Register (PCU_CP1E13_ATTR)	R	0x0000_0000	on page 429
0x0270	PCU Core_1, Entry 14 Address Register (PCU_CP1E14_ADDR)	R	0x0000_0000	on page 428
0x0274	PCU Core_1, Entry 14 Attributes Register (PCU_CP1E14_ATTR)	R	0x0000_0000	on page 429
0x0278	PCU Core_1, Entry 15 Address Register (PCU_CP1E15_ADDR)	R	0x0000_0000	on page 428
0x027C	PCU Core_1, Entry 15 Attributes Register (PCU_CP1E15_ATTR)	R	0x0000_0000	on page 429
0x0280–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 One or more bits (indicated by “U”) are of indeterminate value at Reset. See register for more information.

17.3.1 Register descriptions

17.3.1.1 PCU Configuration and Error Status Register (PCU_CESR)

Address: Base + 0x0000 Access: Supervisor: read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CP1ERR		CP0ERR		SRSTFLAG	0	CP1IDLE	CP0IDLE	0	0	CP1IEN	CP0IEN	0	0	0	0
W																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M3WMEN	M2WMEN	M1WMEN	M0WMEN	0	0	CP1DS		CP0DS				0	0	SRSTEN	ENB
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-4. PCU Configuration and Error Status Register (PCU_CESR)

Table 17-3. PCU_CESR field descriptions

Field	Description
CP1ERR	Core Processor 1 Error. This field represents a flag maintained by the CU for signaling the presence of a captured snoop error or the occurrence of a queue overflow. The error details are contained in PCU_EAR1 and PCU_EDR1. PCU_CESR[31] is set when the snoop queue for Core_1 has overflowed; the address and attributes of the global write that caused the queue overflow are recorded in PCU_EAR1 and PCU_EDR1. PCU_CESR[30] is set when the hardware detects a snoop error and records the faulting address and attributes. This field is cleared when the corresponding bits are written as a logical one. If another Core_1 error occurs before CP1ERR bit is cleared, CP1ERR remains set while PCU_EAR1 and PCU_EDR1 capture address and attributes for the most recent error event. 00 PCU_EAR1/PCU_EDR1 does not contain a captured snoop error. 01 PCU_EAR1/PCU_EDR1 contains a captured snoop error. 10 PCU_EAR1/PCU_EDR1 contains a queue overflow occurrence. 11 An encoding of ‘11’ for this field indicates a snoop error event occurred followed by a queue overflow event. Since the PCU_EAR1/PCU_EDR1 registers capture the most recent error event, the PCU_EAR1/PCU_EDR1 contains the queue overflow occurrence.

Table 17-3. PCU_CESR field descriptions (continued)

Field	Description
CP0ERR	Core Processor 0 Error. This field represents a flag maintained by the CU for signaling the presence of a captured snoop error or the occurrence of a queue overflow. The error details are contained in PCU_EAR0 and PCU_EDR0. PCU_CESR[29] is set when the snoop queue for Core_0 has overflowed; the address and attributes of the global write that caused the queue overflow are recorded in PCU_EAR0 and PCU_EDR0. PCU_CESR[28] is set when the hardware detects a snoop error and records the faulting address and attributes. This field is cleared when the corresponding bits are written as a logical one. If another Core_0 error occurs before CP0ERR bit is cleared, CP0ERR remains set while PCU_EAR0 and PCU_EDR0 capture address and attributes for the most recent error event. 00 PCU_EAR0/PCU_EDR0 does not contain a captured snoop error. 01 PCU_EAR0/PCU_EDR0 contains a captured snoop error. 10 PCU_EAR0/PCU_EDR0 contains a queue overflow occurrence. 11 An encoding of '11' for this field indicates a snoop error event occurred followed by a queue overflow event. Since the PCU_EAR0/PCU_EDR0 registers capture the most recent error event, the PCU_EAR0/PCU_EDR0 contains the queue overflow occurrence.
SRST FLAG	Software reset flag. This field represents a flag maintained by the CU to indicate a software reset event has taken place. A software reset of the CU is invoked by programming the PCU_CESR[SRST_EN] field. 0 The last CU reset event was a hardware reset event. 1 The last CU reset event was a software-invoked reset event.
CP1IDLE	Core Processor 1 Idle. This read-only status flag indicates when Core_1 snoop queue is empty. 0 Core_1 snoop queue not empty; Core_1 snoop queue is busy processing valid snoop requests to Core_1. 1 Core_1 snoop queue is empty.
CP0IDLE	Core Processor 0 Idle. This read-only status flag indicates when Core_0 snoop queue is empty. 0 Core_0 snoop queue not empty; Core_0 snoop queue is busy processing valid snoop requests to Core_0. 1 Core_0 snoop queue is empty.
CP1IEN	Core Processor 1 Interrupt Enable. This bit enables interrupt generation upon detection of a Core_1 snoop request that results in an error. 0 CU generates an interrupt request upon detection of a Core_1 snoop that terminated with error. 1 CU does not generate an interrupt request upon detection of a Core_1 snoop request that terminated with error.
CP0IEN	Core Processor 0 Interrupt Enable. This bit enables interrupt generation upon detection of a Core_0 snoop request that results in an error. 0 CU generates an interrupt request upon detection of a Core_0 snoop that terminated with error. 1 CU does not generate an interrupt request upon detection of a Core_0 snoop request that terminated with error.
M3WMEN	Master 3 Write Monitor Enable. This bit enables monitoring of Master 3 for global writes 0 Disable Master 3 monitoring. Global writes from Master 3 do not initiate any snoop requests. 1 Enable Master 3 monitoring for global writes.
M2WMEN	Master 2 Write Monitor Enable. This bit enables monitoring of Master 2 for global writes 0 Disable Master 2 monitoring. Global writes from Master 2 do not initiate any snoop requests. 1 Enable Master 2 monitoring for global writes.
M1WMEN	Master 1 Write Monitor Enable. This bit enables monitoring of Master 1 for global writes 0 Disable Master 1 monitoring. Global writes from Master 1 do not initiate any snoop requests. 1 Enable Master 1 monitoring for global writes.

Table 17-3. PCU_CESR field descriptions (continued)

Field	Description
M0WMEN	Master 0 Snoop Enable. This bit enables monitoring of Master 0 for global writes 0 Disable Master 0 monitoring. Global writes from Master 0 do not initiate any snoop requests. 1 Enable Master 0 monitoring for global writes.
CP1DS	Core_1 Disable Snoop. This field allows for snooping activity targeting Core_1 to be suspended. Snooping can be disabled universally for Core_1 or upon detection of a Core_1 snoop termination with error or queue overflow occurrence. 000 Core_1 snooping disabled upon detection of an overflow of the Core_1 snoop queue as flagged in PCU_CESR[CP1ERR]. Once PCU_CESR[CP1ERR] field is cleared, Core_1 snooping activity can resume. 001 Core_1 snooping disabled upon detection of an overflow of the Core_1 snoop queue or upon a Core_1 snoop termination with error, as flagged in PCU_CESR[CP1ERR]. Any valid entries in the Core_1 snoop queue are invalidated and are not processed. However, the contents of the Core_1 queue are still available for inspection via the program-visible PCU_CP1Ex_ADDR/PCU_CP1Ex_ATTR registers. Once PCU_CESR[CP1ERR] field is cleared, Core_1 snooping activity can resume. 010 Core_1 snooping disabled upon detection of an overflow of the Core_1 snoop queue or upon detection of an overflow of the Core_0 snoop queue, as flagged in PCU_CESR[CP1ERR]. Any valid entries in the Core_1 snoop queue are invalidated and are not processed. However, the contents of the Core_1 queue are still available for inspection via the program-visible PCU_CP1Ex_ADDR/PCU_CP1Ex_ATTR registers. Once PCU_CESR[CP1ERR] field is cleared, Core_1 snooping activity can resume. 011 Core_1 snooping disabled upon detection of an overflow of either the Core_1 snoop queue or the Core_0 snoop queue or upon a Core_1 snoop termination with error, as flagged in PCU_CESR[CP1ERR] or PCU_CESR[CP0ERR]. Any valid entries in the Core_1 snoop queue are invalidated and are not processed. However, the contents of the Core_1 queue are still available for inspection via the program-visible PCU_CP1Ex_ADDR/PCU_CP1Ex_ATTR registers. Once PCU_CESR[CP1ERR] and PCU_CESR[CP0ERR] fields are cleared, Core_1 snooping activity can resume. 1*xx* Core_1 snooping entirely disabled. Any valid entries in the Core_1 snoop queue are invalidated and are not processed. However, the contents of the Core_1 queue are still available for inspection via the program-visible PCU_CP1Ex_ADDR/PCU_CP1Ex_ATTR registers.

Table 17-3. PCU_CESR field descriptions (continued)

Field	Description
CP0EDS	Core_0 Disable Snoop. This field allows for snooping activity targeting Core_0 to be suspended. Snooping can be disabled universally for Core_0 or upon detection of a Core_0 snoop termination with error or queue overflow occurrence. 000 Core_0 snooping disabled upon detection of an overflow of the Core_0 snoop queue as flagged in PCU_CESR[CP0ERR]. Once PCU_CESR[CP0ERR] field is cleared, Core_0 snooping activity can resume. 001 Core_0 snooping disabled upon detection of an overflow of the Core_0 snoop queue or upon a Core_0 snoop termination with error, as flagged in PCU_CESR[CP0ERR]. Any valid entries in the Core_0 snoop queue are invalidated and are not processed. However, the contents of the Core_0 queue are still available for inspection via the program-visible PCU_CP0Ex_ADDR/PCU_CP0Ex_ATTR registers. Once PCU_CESR[CP0ERR] field is cleared, Core_0 snooping activity can resume. 010 Core_0 snooping disabled upon detection of an overflow of the Core_1 snoop queue or upon detection of an overflow of the Core_0 snoop queue, as flagged in PCU_CESR[CP0ERR]. Any valid entries in the Core_0 snoop queue are invalidated and are not processed. However, the contents of the Core_0 queue are still available for inspection via the program-visible PCU_CP0Ex_ADDR/PCU_CP0Ex_ATTR registers. Once PCU_CESR[CP0ERR] field is cleared, Core_0 snooping activity can resume. 011 Core_0 snooping disabled upon detection of an overflow of either the Core_0 snoop queue or the Core_0 snoop queue or upon a Core_0 snoop termination with error, as flagged in PCU_CESR[CP0ERR] or PCU_CESR[CP1ERR]. Any valid entries in the Core_0 snoop queue are invalidated and are not processed. However, the contents of the Core_0 queue are still available for inspection via the program-visible PCU_CP0Ex_ADDR/PCU_CP0Ex_ATTR registers. Once PCU_CESR[CP0ERR] and PCU_CESR[CP1ERR] fields are cleared, Core_0 snooping activity can resume. 1*xx* Core_0 snooping entirely disabled. Any valid entries in the Core_0 snoop queue are invalidated and are not processed. However, the contents of the Core_0 queue are still available for inspection via the program-visible PCU_CP0Ex_ADDR/PCU_CP0Ex_ATTR registers.
SRST_EN	Software reset enable. Writing a ‘1’ to this bit invokes a software-driven reset of the CU. On the next cycle following the programming of PCU_CESR[SRST_EN], all programming model registers in the CU are reset, and all snoop queue entries are invalidated. PCU_CESR[SRST_FLAG] is set to 1 to indicate a software-driven reset of the CU has occurred. Writing a ‘0’ to this bit has no effect. Reading this bit returns 0. 0 No software-driven reset is requested. Writing a 0 to this field has no effect. Reading this bit always returns 0. 1 Invoke software-driven reset.
ENB	ENB. This bit provides a global enable/disable for the CU. 0 The CU is disabled. 1 The CU is enabled. While the CU is disabled, hardware coherency checking is not performed.

17.3.1.2 PCU Error Address Register, Core *n* (PCU_EAR*n*)

Address: Base + 0x0010(PCU_EAR0)
Base + 0x0020 (PCU_EAR1)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EADDR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EADDR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-5. PCU Error Address Register, Core *n* (PCU_EAR*n*)

Table 17-4. PCU_EAR*n* field descriptions

Field	Description
EADDR	Error Address. This read-only field is the reference address associated with the global write that resulted in a snoop error or overflow event.

17.3.1.3 PCU Error Detail Register, Core *n* (PCU_EDR*n*)

Address: Base + 0x0014 (PCU_EDR0)
Base + 0x0024 (PCU_EDR1)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR				EPTR				ESTMP							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	EPROT						ERESP					0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-6. PCU Error Detail Register, Core *n* (PCU_EDR*n*)

Table 17-5. PCU_EDR*n* field descriptions

Field	Description
EMSTR	Error Master. This read-only field holds the AHB master ID associated with the global write that resulted in a snoop error or overflow event.
EPTR	Error Pointer. This read-only field holds the value of the queue pointer associated with the global write that resulted in a snoop error or overflow event. Upon a snoop error event, this field describes the relative position of the corresponding entry in the 16-entry queue for the global write that resulted in a snoop error. Upon a queue overflow event, this field describes the relative position of the corresponding entry in the 16-entry queue for the global write that resulted in an overflow event.

Table 17-5. PCU_EDR*n* field descriptions (continued)

Field	Description
ESTMP	Error Time Stamp. This read-only fields holds the time stamp value associated with the snoop error or overflow event. The time stamp value is based on a free-running 8-bit timer that begins counting upon assertion of PCU_CESR[ENB]. Upon a snoop error event, this field captures the value of the time stamp at the time the snoop request was loaded into the queue. Upon a queue overflow event, this field captures the value of the time stamp at the time the overflow condition is detected.
EPROT	Error Protection Attribute. This read-only field holds the AHB protection attributes associated with the global write that triggered the snoop request. EPROT[5]: Exclusivity 0 Not Exclusive 1 Exclusive EPROT[4]: EPROT[3]: Cacheability 0 Non-cacheable 1 Cacheable EPROT[2]: Bufferable 0 Non-bufferable 1 Bufferable EPROT[1]: Mode 0 User Mode 1 Supervisor Mode EPROT[0]: Type 0 Instruction 1 Data
ERESP	Error snoop response. This read-only field holds the snoop response associated with the snoop that terminated with error. This field is cleared when PCU_EDR*n* is loaded on an overflow event. Upon a queue overflow event, this field reads as all zeroes. 000cc NULL. No operation performed or no matching cache entry. 001cc AutoInv. AutoInvalidation performed on clean unlocked lines with tag parity errors. 010cc ERROR. Error in processing a snoop request due to tag parity error. 01100 SYNC. Sync completed, snoop queue synchronized. 100cc Hit Clean. Matching unlocked clean cache entry found. 101cc Hit Dirty. Matching unlocked dirty cache entry found. 110cc Hit Locked. Matching locked clean cache entry found. 111cc Hit Dirty Locked. matching locked clean cache entry found. cc Number of collapsed requests: 00 0 collapsed requests. 01 2 requests combined. 10 3 requests combined. 11 4 requests combined.

17.3.1.4 PCU Interrupt Register, Core *n* (PCU_IR*n*)

Address: Base + 0x30 (PCU_IR0)
Base + 0x34 (PCU_IR1)

Access: Supervisor: read/write;
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	OIF	EIF
W															w1c	w1c
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	0

Figure 17-7. PCU Interrupt Register, Core *n* (PCU_IR*n*)

Table 17-6. PCU_IR*n* field descriptions

Field	Description
OIF	Overflow Interrupt Flag. This field indicates an outstanding interrupt request as a result of a queue overflow. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 to this bit has no effect. 0 No interrupt request. 1 Interrupt request due to snoop termination with error.
EIF	Error Interrupt Flag. This field indicates an outstanding interrupt request as a result of a snoop that terminated with error if PCU_CESR[CP*n*IEN]. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 to this bit has no effect. 0 No interrupt request. 1 Interrupt request due to snoop termination with error.

17.3.1.5 PCU Core *n*, Entry *x* Address (PCU_CP*n*E*x*_ADDR)

Address: See Table 17-2

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADDR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADDR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-8. PCU Core *n*, Entry *x* Address (PCU_CP*n*E*x*_ADDR)

Table 17-7. PCU_CP*n*E*x*_ADDR field descriptions

Field	Description
ADDR	Entry Address. This read-only field holds the address for the snoop request in the corresponding queue entry.

17.3.1.6 PCU Core *n*, Entry *x* Attributes (PCU_CP*n*E*x*_ATTR)

Address: See Table 17-2 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR				0	0	0	0	STAMP							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	HPROT						RESP					0	0	VLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-9. PCU Core *n*, Entry *x* Attributes (PCU_CP*n*E*x*_ATTR)

Table 17-8. PCU_CP*n*E*x*_ATTR field descriptions

Field	Description
MSTR	Entry Master. This read-only field holds the Master ID for the snoop request in the corresponding queue entry.
STMP	Entry Time Stamp. This read-only field holds the time stamp value at the time the snoop request was loaded into the queue.The time stamp value is based on a free-running 8-bit timer that begins counting upon assertion of PCU_CESR[ENB].
HPROT	Entry Protection Attribute. This read-only field is the holds the hprot values for the snoop request in the corresponding queue entry. HPROT[5]: Exclusivity 0 Not Exclusive 1 Exclusive HPROT[4]: HPROT[3]: Cacheability 0 Non-cacheable 1 Cacheable HPROT[2]: Bufferable 0 Non-bufferable 1 Bufferable HPROT[1]: Mode 0 User Mode 1 Supervisor Mode HPROT[0]: Type 0 Instruction 1 Data

Table 17-8. PCU_CP*n*E*x*_ATTR field descriptions (continued)

Field	Description
RESP	Snoop response. This read-only field holds the resultant snoop response from the corresponding processor. 000cc NULL. No operation performed or no matching cache entry. 001cc AutoInv. AutoInvalidation performed on clean unlocked lines with tag parity errors. 010cc ERROR. Error in processing a snoop request due to tag parity error. 01100 SYNC. Sync completed, snoop queue synchronized. 100cc Hit Clean. Matching unlocked clean cache entry found. 101cc Hit Dirty. Matching unlocked dirty cache entry found. 110cc Hit Locked. Matching locked clean cache entry found. 111cc Hit Dirty Locked. Matching locked clean cache entry found. cc Number of collapsed requests: 00 0 collapsed requests. 01 2 requests combined. 10 3 requests combined. 11 4 requests combined.
VLD	Entry Valid. This read-only field indicates the corresponding snoop queue entry is valid. 0 Queue entry is not valid. 1 Queue entry is valid.

17.4 Functional description

The CU is responsible for tracking updates to shared memory spaces and invoking snoop requests to the processor(s) in an effort to maintain coherency within a dual-core system. The CU monitors all AHB transactions originating from as many as four masters that have been identified to use shared memory space. Upon detection of a global write from any of these masters, the CU routes a coherency snoop request to the appropriate cores at the address in question. Writes are marked as ‘global’ via an AHB address-phase sideband signal. For transactions originating from the cores, the ‘global’ sideband information is provided by forwarding the M-bit from the corresponding MMU entry.

17.4.1 Arbitration and routing

Table 17-9 describes the rules used by the corresponding TCD for dispatching snoop requests to the appropriate core(s). Global writes under normal operation originating from Core_0 should be alerted to Core_1 as a coherency snoop request. Similarly, global writes under normal operation originating from Core_1 should be alerted to Core_0 as a coherency snoop request. Global Nexus writes originating from either core require a snoop request to both processors. Global writes originating from non-processor masters require snoop requests dispatched to both cores.

The determination of which master initiated a global write is performed by comparing the hmaster supplied in the address phase against the configuration of the inputs cpX_master_id[3:0] and cpX_nex_master_id[3:0], where X is 0 or 1.

- If the hmaster value for the current transaction matches cp0_master_id[3:0] or cp0_nex_master_id[3:0], then Core_0 is identified as the master that initiated the global write, and the snoop request is routed to Core_1 and optionally to Core_0 on a Nexus global write.

- If the hmaster value for the current transaction matches cp1_master_id[3:0] or cp1_nex_master_id[3:0], then Core_1 is identified as the master that initiated the global write, and the snoop request is routed to Core_0 and optionally to Core_1 on a Nexus global write.
- If the hmaster value for the current transaction does not match cpX_hmaster_id[3:0] nor cpX_nex_hmaster_id[3:0], then a non-processor master is identified as the master that initiated the global write, and the snoop request is routed to both Core_0 and Core_1.

Table 17-9. Routing rules for snoop requests

Global write originating from	Send snoop request to Core_0	Send snoop request to Core_1
Core_0 (normal access)		*
Core_0 (Nexus access)	*	*
Core_1 (normal access)	*	
Core_1 (Nexus access)	*	*
Master 2	*	*
Master 3	*	*

17.4.2 Protocol and handshaking

When issuing a snoop request, the CU interfaces with the core's snoop port. A request is issued by asserting cpX_snp_req and cpX_snp_cmd[1:0], which specifies the coherency action to be taken, where X is 0 or 1. As the CU only supports snoop invalidate requests, the only supported encoding for cpX_snp_cmd is 2'b01.

Table 17-10. Snoop command encoding

cpX_snp_cmd[1:0]	Command type
00	NULL — lookup performed, no status bit operation performed * Not supported
01	INV — invalidate matching cache entry
10	SYNC — synchronize snoop queue
11	Reserved

The CU provides cpX_snp_addr[31:5] as the physical address to be used for look-up. cpX_snp_id[3:0] provides a 4-digit ID to keep track of outstanding snoop requests.

The CU samples cpX_snp_rdy from the core to determine when a snoop request has been taken or when the CU needs to continue asserting the snoop request. When the snoop queue internal to the core is full, cpX_snp_rdy is de-asserted indicating the core is unavailable to take any new snoop requests. Any outstanding snoop requests must be held while cpX_snp_rdy is de-asserted. For a detailed description of cpX_snp_rdy behavior, refer to z760n3 Specification (Section 14.3.4 Cache Coherency Operation).

Upon termination of a snoop request, the core provides cpX_snp_ack along with cpX_snp_ack_id[3:0] confirming which request has terminated. Upon termination, the core also provides cpX_snp_ack_resp[4:0] to describe the outcome of the coherency action.

Table 17-11. Snoop response encoding

cpX_snp_resp[4:0]	Response type
000cc[1]	NULL. No operation performed or no matching cache entry
001cc	AutoInv. AutoInvalidation performed on clean unlocked lines with tag parity errors.
010cc	ERROR. Error in processing a snoop request due to TAG parity error.
01100	SYNC. Sync completed, snoop queue synchronized.
100cc	Hit Clean. Matching unlocked cache entry found.
101cc	Hit Dirty. Matching unlocked dirty cache entry found.
110cc	Hit Locked. Matching clean locked cache entry found.
111cc	Hit Dirty Locked. matching dirty locked cache entry found.

1 cc—Number of collapsed requests. 00 = 0 requests combined, 01 = 2 requests combined, 10 = 3 requests combined, 11 = 4 requests combined.

17.4.3 Snoop queue

NOTE

This section describes the CU snoop queue, which is unrelated to the snoop queue internal to the e200z7d core.

The snoop queue provides a mechanism for queueing and tracking snoop requests. There are two queues—one for tracking requests to each core. The 16-entry queue is loaded in a round-robin fashion. Snoop requests are dispatched FIFO-style. Each entry holds the following information:

- haddr
- hprot
- hmaster
- request ID
- valid bit

Upon detection of global write activity requiring a cache snoop, an entry is loaded in the queue and marked valid. The entry remains valid until confirmation is received from the core that the snoop request has been handled. When the queue is approaching full occupancy, the CU is responsible for issuing a request to all masters to stall subsequent global writes by asserting stall_gbl_write to avoid an overflow situation. The CU continues to assert stall_gbl_write until there is room in the queue to accept new snoop requests.

17.4.4 Lock Step mode (LSM) considerations

For integration on safety systems that operate in Lock Step mode (LSM), there are special considerations for snooping. Since the cores are presumably performing symmetric memory accesses in LSM, the CU ignores global writes from either core, thus preventing the loading of snoop requests in response to core-driven global writes. However, global Nexus writes initiated by either core should continue to be monitored and potentially generate snoop requests even in LSM. Also, global writes initiated by non-processor masters should continue to be monitored and potentially generate snoop requests even in LSM.

17.4.5 Stop mode considerations

There are specific considerations for how the CU should respond when the core(s) enter stop mode.

Consider an example where p_stop is asserted for Core_0, indicating a request for Core_0 to enter stop mode. The CU can expect Core_0 to finish processing any outstanding snoop requests before asserting p_stopped and entering stop mode, at which point Core_0 ignores any new snoop requests. In preparation for entering stop mode, Core_0 deasserts cp0_p_snp_rdy as an indication it will not accept any new snoop requests. Keep in mind the protocol associated with the deassertion of cp{0,1}_snp_rdy dictates that the Core_0 can potentially take as many as two more snoop requests following the deassertion of cp0_snp_rdy. Since Core_0 is unavailable to respond to any snoop requests once it enters stop mode, coherency can no longer be guaranteed. Accordingly, the CU invalidates any outstanding snoop requests in the corresponding queue. Furthermore, the CU does not load any new requests into the queue in response to global write activity for the duration of time that p_stop or p_stopped is asserted.

Each core can independently enter stop mode. Accordingly the guidelines described above are honored on a per-core basis. Also, while one core is in stop mode, snooping activity may continue uninterrupted in the other core, assuming it is not in stop mode as well.

17.4.6 Memory synchronization control

17.4.6.1 Memory synchronization request handling

In addition to generating snoop requests to maintain coherency, the CU is responsible for managing the handshaking between the two cores during the execution of an msync instruction. Consider an example where Core_0 is processing an msync instruction. Core_0 waits for all preceding memory accesses to complete, including snoop requests. Core_0 then sends a memory synchronization request to Core_1 by launching p_sync_req_out to be received at Core_1's input port, p_sync_req_in. For this example, this signal is referred to as cp0_p_snp_req_out. See Table 17-10. The CU is responsible for preventing Core_1 from seeing cp0_p_snp_req_out until all outstanding Core_1 snoop requests have been handled. More specifically, the CU must process all Core_1 snoop requests that are outstanding at the time cp0_p_sync_req_out is asserted. Subsequent Core_1 snoop requests beyond the first cycle that cp0_p_sync_req_out is asserted do not have any gating impact on cp0_p_sync_req_out. This ensures all outstanding memory accesses are completed before initiating a synchronization request. Once all outstanding snoop requests are processed, the CU forwards the synchronization request to Core_1 in the form of cp0_p_sync_req_out_qual. The CU continues to assert cp0_p_sync_req_out_qual until an acknowledge is received in the form of cp1_p_sync_ack_out from Core_1. Early negation of cp0_p_sync_req_out by Core_0 may result in Core_1 never seeing assertion of cp0_p_sync_req_out_qual if the CU is still processing outstanding snoop requests when Core_0 negates cp0_p_sync_req_out. The early negation of cp0_p_sync_req_out nullifies any indication that the synchronization completed successfully.

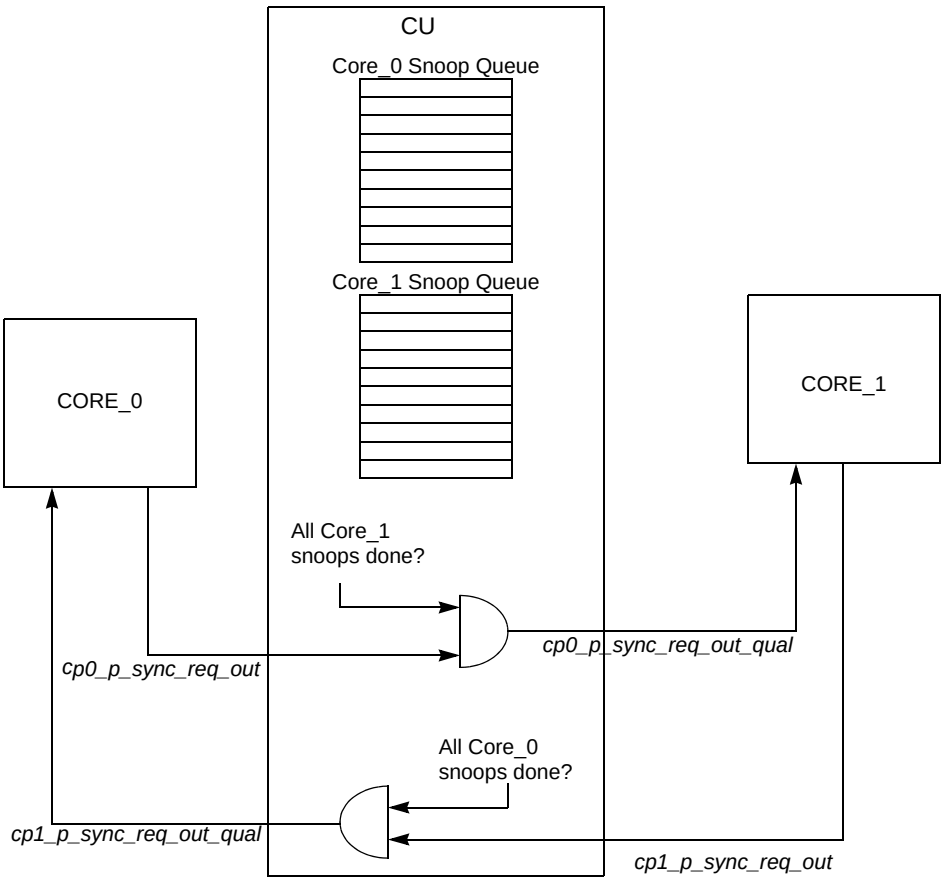


Figure 17-10. Synchronization request control

17.4.6.2 Memory synchronization in LSM

When operating in LSM, the CU gates all memory synchronization requests from proceeding to the destination core. Instead, the CU short-circuits the synchronization handshake and provides cpX_p_sync_ack_out to the initiating core.

17.5 Timing diagrams

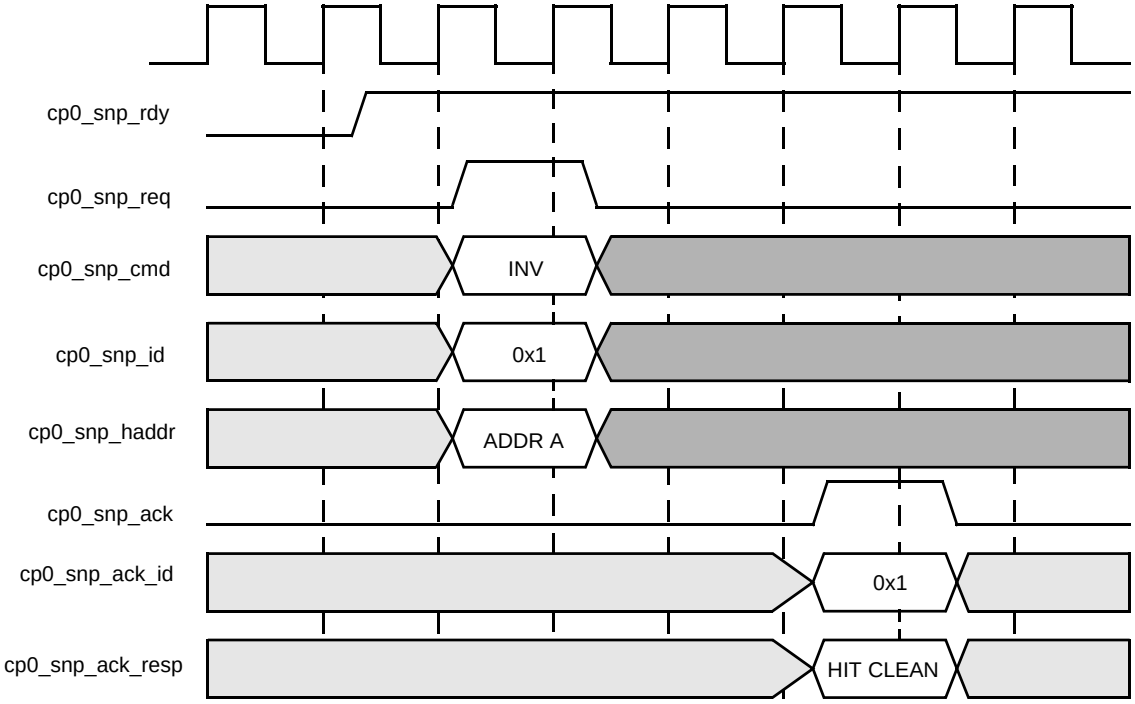


Figure 17-11. Basic snoop request protocol

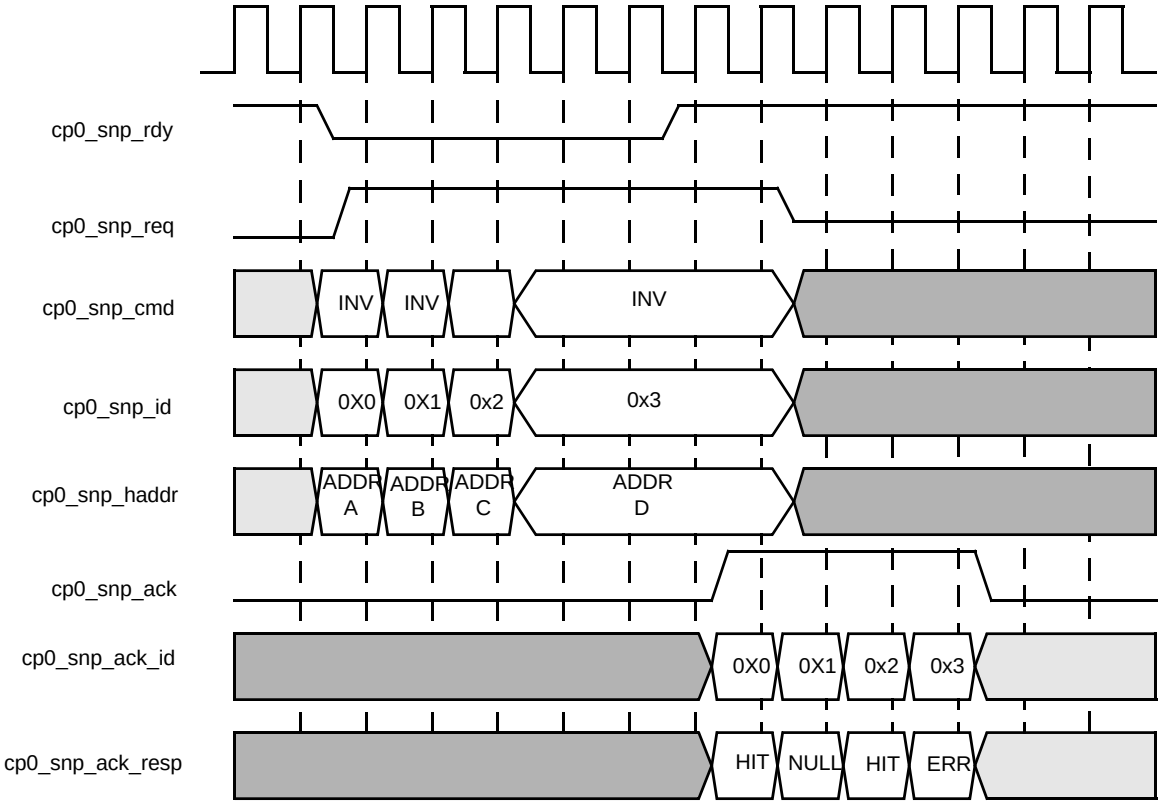


Figure 17-12. Back-to-back snoop requests to Master 0

Chapter 18
Cross-Triggering Unit (CTU)

18.1 Introduction

In pulse-width modulation (PWM)-driven systems, it is important to schedule the acquisition of the state variables with respect to PWM cycle. State variables are obtained through the following peripherals: ADC, position counter (for example, quadrature decoder, resolver, and sine-cosine sensor), and PWM duty cycle decoder.

The cross triggering unit (CTU) is intended to completely avoid CPU involvement in the time acquisitions of state variables during the control cycle that can be the PWM cycle, the half PWM cycle, or a number of PWM cycles. In such case the pre-setting of the acquisition time needs to be completed during the previous control cycle, where the actual acquisitions are to be made, and a double-buffered structure for the CTU registers is used, in order to activate the new settings at the beginning of the next control cycle. In addition, four FIFOs inside the CTU are available to store the ADC results.

18.2 Block diagram

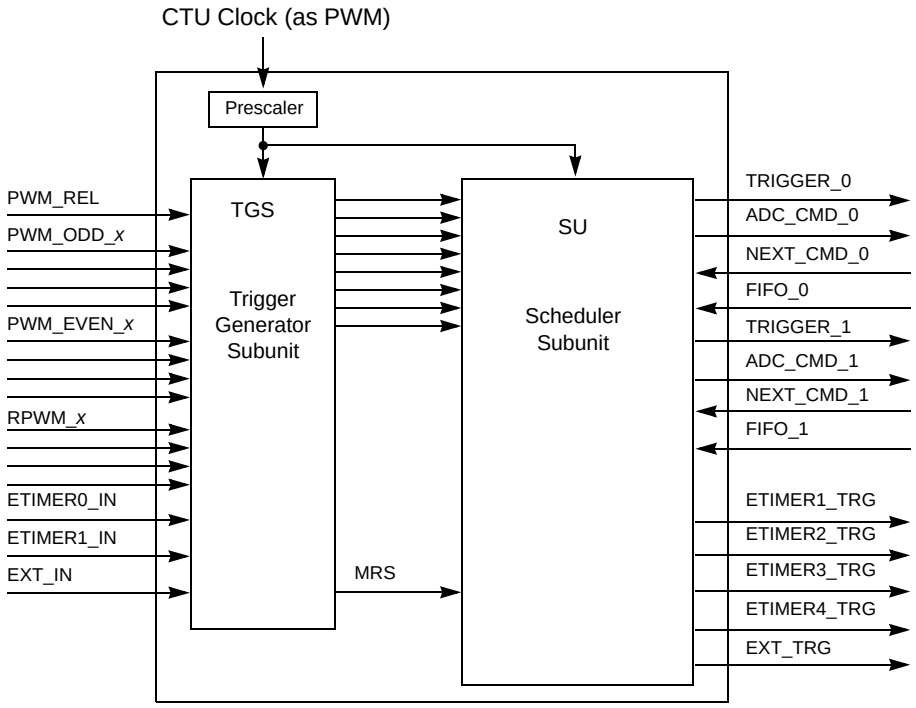


Figure 18-1. CTU block diagram

18.3 CTU overview

The CTU receives various incoming input signals from different sources (PWM, timers, external pins). An input can be a rising edge, a falling edge, or both edges of each incoming signal. These signals are then

processed to generate 8 discrete trigger events. The trigger-generated output can be a pulse, a command (or a stream of consecutive commands for over-sampling support), or both, sent to one or more peripherals (for example, ADC, timers, and so on).

The 16 input signals are digital signals and the CTU can detect a rising edge, a falling edge, or both edges for each of them. Outputs are driven internally to the on-chip peripherals, and can generate external signals depending on programmatic configuration.

The CTU comprises the following:

- PWM input signal interface
- User interface (including configuration registers)
- ADC input/output interface
- eTimer input/output interface
- External signal input/output interface

The block diagram of the CTU is shown in Figure 18-1. The CTU consists of two sub units:

- Trigger generator sub unit (TGS)
- Scheduler sub unit (SU)

The trigger generator sub unit (TGS) handles incoming signals and generates as many as eight trigger events (signals), selecting the active edges to generate the Master Reload Signal (MRS) for each signal. The scheduler sub unit (SU) generates the trigger event output according to the trigger event (signal) that has occurred.

18.4 Memory map and registers description

Different size registers with different accesses are available for this module:

- 32-bit registers: byte access for write operations and 32-bit access for read operation
- 16-bit registers: byte access for write operations and 32-bit access for read operations

If a 32-bit write operation is performed on a 16-bit register, the write operation is performed on the 32 aligned bits. Read operations can be performed only at 32 bits.

Table 18-1 shows the base addresses for the CTU modules. Addresses are the same for Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). Table 18-2 lists the CTU registers.

Table 18-1. CTU module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM) Decoupled Parallel Mode (DPM)	CTU_0	0xFFE0_C000
	CTU_1	0xC3E5_C000

Table 18-2. CTU memory map

Offset from CTU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Trigger Generator Sub Unit Input Selection Register (TGSISR)	R/W	0x0000_0000	on page 442
0x0004	Trigger Generator Sub Unit Control Register (TGSCR)	R/W	0x0000	on page 445
0x0006	Trigger 0 Compare Register (T0CR)	R/W	0x0000	on page 447
0x0008	Trigger 1 Compare Register (T1CR)	R/W	0x0000	on page 447
0x000A	Trigger 2 Compare Register (T2CR)	R/W	0x0000	on page 447
0x000C	Trigger 3 Compare Register (T3CR)	R/W	0x0000	on page 447
0x000E	Trigger 4 Compare Register (T4CR)	R/W	0x0000	on page 447
0x0010	Trigger 5 Compare Register (T5CR)	R/W	0x0000	on page 447
0x0012	Trigger 6 Compare Register (T6CR)	R/W	0x0000	on page 447
0x0014	Trigger 7 Compare Register (T7CR)	R/W	0x0000	on page 447
0x0016	TGS Counter Compare Register (TGSCCR)	R/W	0x0000	on page 447
0x0018	TGS Counter Reload Register (TGSCRR)	R/W	0x0000	on page 448
0x001A	Reserved			
0x001C	Commands List Control Register 1 (CLCR1)	R/W	0x0000_0000	on page 448
0x0020	Commands List Control Register 2 (CLCR2)	R/W	0x0000_0000	on page 449
0x0024	Trigger Handler Control Register 1 (THCR1)	R/W	0x0000_0000	on page 449
0x0028	Trigger Handler Control Register 2 (THCR2)	R/W	0x0000_0000	on page 452
0x002C	Commands List Register 1 (CLR1)	R/W	0x0000	on page 455
0x002E	Commands List Register 2 (CLR2)	R/W	0x0000	on page 455
0x0030	Commands List Register 3 CLR3)	R/W	0x0000	on page 455
0x0032	Commands List Register 4 (CLR4)	R/W	0x0000	on page 455
0x0034	Commands List Register 5 (CLR5)	R/W	0x0000	on page 455
0x0036	Commands List Register 6 (CLR6)	R/W	0x0000	on page 455
0x0038	Commands List Register 7 (CLR7)	R/W	0x0000	on page 455
0x003A	Commands List Register 8 (CLR8)	R/W	0x0000	on page 455
0x003C	Commands List Register 9 (CLR9)	R/W	0x0000	on page 455
0x003E	Commands List Register 10 (CLR10)	R/W	0x0000	on page 455
0x0040	Commands List Register 11 (CLR11)	R/W	0x0000	on page 455
0x0042	Commands List Register 12 (CLR12)	R/W	0x0000	on page 455
0x0044	Commands List Register 13 (CLR13)	R/W	0x0000	on page 455
0x0046	Commands List Register 14 (CLR14)	R/W	0x0000	on page 455

Table 18-2. CTU memory map (continued)

Offset from CTU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0048	Commands List Register 15 (CLR15)	R/W	0x0000	on page 455
0x004A	Commands List Register 16 (CLR16)	R/W	0x0000	on page 455
0x004C	Commands List Register 17 (CLR17)	R/W	0x0000	on page 455
0x004E	Commands List Register 18 (CLR18)	R/W	0x0000	on page 455
0x0050	Commands List Register 19 (CLR19)	R/W	0x0000	on page 455
0x0052	Commands List Register 20 (CLR20)	R/W	0x0000	on page 455
0x0054	Commands List Register 21 (CLR21)	R/W	0x0000	on page 455
0x0056	Commands List Register 22 (CLR22)	R/W	0x0000	on page 455
0x0058	Commands List Register 23 (CLR23)	R/W	0x0000	on page 455
0x005A	Commands List Register 24 (CLR24)	R/W	0x0000	on page 455
0x005C–0x006B	Reserved			
0x006C	FIFO DMA Control Register (FDCR)	R/W	0x0000_0000	on page 457
0x0070	FIFO Control Register (FCR)	R/W	0x0000_0000	on page 457
0x0074	FIFO Threshold Register (FTH)	R/W	0x0000_0000	on page 459
0x0078–0x007B	Reserved			
0x007C	FIFO Status Register (FST)	R/W	0x0000_0000	on page 460
0x0080	FIFO Right Aligned Data Register 0 (FR0)	R	0x0000_0000	on page 461
0x0084	FIFO Right Aligned Data Register 1 (FR1)	R	0x0000_0000	on page 461
0x0088	FIFO Right Aligned Data Register 2 (FR2)	R	0x0000_0000	on page 461
0x008C	FIFO Right Aligned Data Register 3 (FR3)	R	0x0000_0000	on page 461
0x0090–0x009F	Reserved			
0x00A0	FIFO Signed Left Aligned Data Register 0 (Fl0)	R	0x0000_0000	on page 462
0x00A4	FIFO Signed Left Aligned Data Register 1 (FL1)	R	0x0000_0000	on page 462
0x00A8	FIFO Signed Left Aligned Data Register 2 (FL2)	R	0x0000_0000	on page 462
0x00AC	FIFO Signed Left Aligned Data Register 3 (FL3)	R	0x0000_0000	on page 462
0x00B0–0x00BF	Reserved			
0x00C0	Cross Triggering Unit Error Flag Register (CTUEFR)	R/W	0x0000	on page 463
0x00C2	Cross Triggering Unit Interrupt Flag Register (CTUIFR)	R/W	0x0000	on page 465
0x00C4	Cross Triggering Unit Interrupt Register (CTUIR)	R/W	0x0000	on page 466
0x00C6	Control On-Time Register (COTR)	R/W	0x0000	on page 467
0x00C8	Cross Triggering Unit Control Register (CTUCR)	R/W	0x0000	on page 468
0x00CA	Cross Triggering Unit Digital Filter (CTUDF)	R/W	0x0000	on page 470

Table 18-2. CTU memory map (continued)

Offset from CTU_BASE	Register	Access[1]	Reset Value[2]	Location
0x00CC	Cross Triggering Unit Expected Value A Register (CTU_EXPECTED_A)	R/W	0xFFFF	on page 470
0x00CE	Cross Triggering Unit Expected Value B Register (CTU_EXPECTED_B)	R/W	0xFFFF	on page 470
0x00D2	Cross Triggering Unit Count Range Register (CTU_CNT_RANGE)	R/W	0x0000	on page 471
0x00D6–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

Table 18-3. CTU register buffering and synchronization

Address offset	Register	Double-buffered	Synchronization
TGS registers			
0x0000	TGSISR — Trigger Generator Sub Unit Input Selection Register	Yes	TGSISR_RE
0x0004	TGSCR — Trigger Generator Sub Unit Control Register	Yes	MRS
0x0006	T0CR — Trigger 0 Compare Register	Yes	MRS
0x0008	T1CR — Trigger 1 Compare Register	Yes	MRS
0x000A	T2CR — Trigger 2 Compare Register	Yes	MRS
0x000C	T3CR — Trigger 3 Compare Register	Yes	MRS
0x000E	T4CR — Trigger 4 Compare Register	Yes	MRS
0x0010	T5CR — Trigger 5 Compare Register	Yes	MRS
0x0012	T6CR — Trigger 6 Compare Register	Yes	MRS
0x0014	T7CR — Trigger 7 Compare Register	Yes	MRS
0x0016	TGSCCR — TGS Counter Compare Register	Yes	MRS
0x0018	TGSCRR — TGS Counter Reload Register	Yes	MRS
SU registers			
0x001C	CLCR1 — Commands List Control Register 1	Yes	MRS
0x0020	CLCR2 — Commands List Control Register 2	Yes	MRS
0x0024	THCR1 — Trigger Handler Control Register 1	Yes	MRS
0x0028	THCR2 — Trigger Handler Control Register 2	Yes	MRS

Table 18-3. CTU register buffering and synchronization (continued)

Address offset	Register	Double-buffered	Synchronization
0x002C ... 0x005A	CLR*x* — Commands List Register *x* (*x* = 1,...,24)	Yes	MRS
CTU registers			
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	No	—
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	No	—
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	No	—
0x00C6	COTR — Control ON-Time Register	Yes	MRS
0x00C8	CTUCR — Cross Triggering Unit Control Register	No	—
0x00CA	CTUDF — Cross Triggering Unit Digital Filter	No	DFE
0x00CC	CTU_EXP_A — Cross Triggering Unit Expected Value A	No	—
FIFO Registers			
0x006C	FDCR — FIFO DMA Control Register	No	—
0x0070	FCR — FIFO Control Register	No	—
0x0074	FTH — FIFO Threshold	No	—
0x007C	FST — FIFO Status Register	No	—
0x0080	FR0 — FIFO Right aligned data 0	No	—
0x0084	FR1 — FIFO Right aligned data 1	No	—
0x0088	FR2 — FIFO Right aligned data 2	No	—
0x008C	FR3 — FIFO Right aligned data 3	No	—
0x00A0	FL0 — FIFO Left aligned data 0	No	—
0x00A4	FL1 — FIFO Left aligned data 1	No	—
0x00A8	FL2 — FIFO Left aligned data 2	No	—
0x00AC	FL3 — FIFO Left aligned data 3	No	—

18.4.1 Register description

18.4.1.1 Trigger Generator Sub Unit Input Selection Register (TGSISR)

The Trigger Generator Sub Unit Input Selection Register (TGSISR) selects the input conditions that cause a Master Reload Signal (MRS) to be generated.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	I15_FE	I15_RE	I14_FE	I14_RE	I13_FE	I13_RE	I12_FE	I12_RE	I11_FE	I11_RE	I10_FE	I10_RE	I9_FE	I9_RE	I8_FE	I8_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	I7_FE	I7_RE	I6_FE	I6_RE	I5_FE	I5_RE	I4_FE	I4_RE	I3_FE	I3_RE	I2_FE	I2_RE	I1_FE	I1_RE	I0_FE	I0_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-2. Trigger Generator Sub Unit Input Selection Register (TGSISR)

Table 18-4. TGSISR field descriptions

Field	Description
I15_FE	Input 15 — GPIO falling edge enable 0 Disabled. 1 Enabled.
I15_RE	Input 15 — GPIO rising edge enable 0 Disabled. 1 Enabled.
I14_FE	Input 14 — ETIMER1_IN falling edge enable 0 Disabled. 1 Enabled.
I14_RE	Input 14 — ETIMER1_IN rising edge enable 0 Disabled. 1 Enabled.
I13_FE	Input 13 — ETIMER0_IN falling edge enable 0 Disabled. 1 Enabled.
I13_RE	Input 13 — ETIMER0_IN rising edge enable 0 Disabled. 1 Enabled.
I12_FE	Input 12 — Real PWM Channel 3 falling edge enable 0 Disabled. 1 Enabled.
I12_RE	Input 12 — Real PWM Channel 3 rising edge enable 0 Disabled. 1 Enabled.
I11_FE	Input 11 — Real PWM Channel 2 falling edge enable 0 Disabled. 1 Enabled.
I11_RE	Input 11 — Real PWM Channel 2 rising edge enable 0 Disabled. 1 Enabled.

Table 18-4. TGSISR field descriptions (continued)

Field	Description
I10_FE	Input 10 — Real PWM Channel 1 falling edge enable 0 Disabled. 1 Enabled.
I10_RE	Input 10 — Real PWM Channel 1 rising edge enable 0 Disabled. 1 Enabled.
I9_FE	Input 9 — Real PWM Channel 0 falling edge enable 0 Disabled. 1 Enabled.
I9_RE	Input 9 — Real PWM Channel 0 rising edge enable 0 Disabled. 1 Enabled.
I8_FE	Input 8 — PWM Channel 3 even falling edge enable 0 Disabled. 1 Enabled.
I8_RE	Input 8 — PWM Channel 3 even rising edge enable 0 Disabled. 1 Enabled.
I7_FE	Input 7 — PWM Channel 2 even falling edge enable 0 Disabled. 1 Enabled.
I7_RE	Input 7 — PWM Channel 2 even rising edge enable 0 Disabled. 1 Enabled.
I6_FE	Input 6 — PWM Channel 1 even falling edge enable 0 Disabled. 1 Enabled.
I6_RE	Input 6 — PWM Channel 1 even rising edge enable 0 Disabled. 1 Enabled.
I5_FE	Input 5 — PWM Channel 0 even falling edge enable 0 Disabled. 1 Enabled.
I5_RE	Input 5 — PWM Channel 0 even rising edge enable 0 Disabled. 1 Enabled.
I4_FE	Input 4 — PWM Channel 3 odd falling edge enable 0 Disabled. 1 Enabled.
I4_RE	Input 4 — PWM Channel 3 odd rising edge enable 0 Disabled. 1 Enabled.

Table 18-4. TGSISR field descriptions (continued)

Field	Description
I3_FE	Input 3 — PWM Channel 2 odd falling edge enable 0 Disabled. 1 Enabled.
I3_RE	Input 3 — PWM Channel 2 odd rising edge enable 0 Disabled. 1 Enabled.
I2_FE	Input 2 — PWM Channel 1 odd falling edge enable 0 Disabled. 1 Enabled.
I2_RE	Input 2 — PWM Channel 1 odd rising edge enable 0 Disabled. 1 Enabled.
I1_FE	Input 1 — PWM Channel 0 odd falling edge enable 0 Disabled. 1 Enabled.
I1_RE	Input 1 — PWM Channel 0 odd rising edge enable 0 Disabled. 1 Enabled.
I0_FE	Input 0 — PWM Reload falling edge enable 0 Disabled. 1 Enabled.
I0_RE	Input 0 — PWM Reload rising edge enable 0 Disabled. 1 Enabled.

18.4.1.2 Trigger Generator Sub Unit Control Register (TGSCR)

The Trigger Generator Sub Unit Control Register (TGSCR) sets a number of controls for CTU operation.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	ET_TM	PRES		MRS_SM					TGS_M
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-3. Trigger Generator Sub Unit Control Register (TGSCR)

Table 18-5. TGSCR field descriptions

Field	Description
ET_TM	This bit is used to enable toggle mode for external trigger. 0 Toggle mode is disabled. 1 Toggle mode is enabled.
PRES	TGS and SU prescaler selection bits. This bitfield is the prescaler for the counter clock. 00 Divide by 1. 01 Divide by 2. 10 Divide by 3. 11 Divide by 4.
MRS_SM	Master Reload Signal (MRS) Selection in Sequential Mode. When the CTU is configured in sequential mode, this bitfield selects the event that causes the Master Reload Signal. See Table 18-6.
TGS_M	Trigger Generator Sub Unit Mode 0 Triggered mode is selected. 1 Sequential mode is selected.

Table 18-6. MRS_SM encoding

Value		Description	Value		Description
Dec	Bin		Dec	Bin	
0	00000	PWM Reload rising edge enable	16	10000	PWM Channel 3 even rising edge enable
1	00001	PWM Reload falling edge enable	17	10001	PWM Channel 3 even falling edge enable
2	00010	PWM Channel 0 odd rising edge enable	18	10010	Real PWM Channel 0 rising edge enable
3	00011	PWM Channel 0 odd falling edge enable	19	10011	Real PWM Channel 0 falling edge enable
4	00100	PWM Channel 1 odd rising edge enable	20	10100	Real PWM Channel 1 rising edge enable
5	00101	PWM Channel 1 odd falling edge enable	21	10101	Real PWM Channel 1 falling edge enable
6	00110	PWM Channel 2 odd rising edge enable	22	10110	Real PWM Channel 2 rising edge enable
7	00111	PWM Channel 2 odd falling edge enable	23	10111	Real PWM Channel 2 falling edge enable
8	01000	PWM Channel 3 odd rising edge enable	24	11000	Real PWM Channel 3 rising edge enable
9	01001	PWM Channel 3 odd falling edge enable	25	11001	Real PWM Channel 3 falling edge enable
10	01010	PWM Channel 0 even rising edge enable	26	11010	ETIMER0_IN rising edge enable
11	01011	PWM Channel 0 even falling edge enable	27	11011	ETIMER0_IN falling edge enable
12	01100	PWM Channel 1 even rising edge enable	28	11100	ETIMER1_IN rising edge enable
13	01101	PWM Channel 1 even falling edge enable	29	11101	ETIMER1_IN falling edge enable

Table 18-6. MRS_SM encoding

Value		Description	Value		Description
Dec	Bin		Dec	Bin	
14	01110	PWM Channel 2 even rising edge enable	30	11110	GPIO rising edge enable
15	01111	PWM Channel 2 even falling edge enable	31	11111	GPIO falling edge enable

18.4.1.3 Trigger *x* Compare Register (T*x*CR)

The Trigger *x* Compare Registers (T*x*CR) contain the values that are compared with a counter value to generate triggers for the scheduler subunits.

Address: Base + 0x0006 (T0CR), Base + 0x0008 (T1CR), Base + 0x000A (T2CR), Base + 0x000C (T3CR), Base + 0x000E (T4CR), Base + 0x0010 (T5CR), Base + 0x0012 (T6CR), Base + 0x0014 (T7CR)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TxCRV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-4. Trigger *x* Compare Register (T*x*CR)

Table 18-7. TxCR field descriptions

Field	Description
TxCRV	Trigger *x* Compare Register Value

18.4.1.4 TGS Counter Compare Register (TGSCCR)

The TGS Counter Compare Register (TGSCCR) provides the value that is compared to the counter.

Address: Base + 0x0016

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCCV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-5. TGS Counter Compare Register (TGSCCR)

Table 18-8. TGSCCR field descriptions

Field	Description
TGSCCV	TGS Counter Compare Value

18.4.1.5 TGS Counter Reload Register (TGSCRR)

The TGSCRR provides the value that is loaded into the counter each time the counter is started or restarted. It is used to reload the counter when MRS (in triggered mode) or ES (in sequential mode) is one. The value of this register is load in the motor control domain after the master reload and is used only after an other master reload.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCRV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-6. TGS counter reload register (TGSCRR)

Table 18-9. TGSCRR field descriptions

Field	Description
TGSCRV	TGS Counter Reload Value

18.4.1.6 Commands List Control Register 1 (CLCR1)

The CLCR1 contains the command to be sent when a specified trigger is generated. It selects one of the 24 CLR commands as the first command when a trigger is generated and sends it to the ADC. CLCR1 and CLCR2 are functionally the same, but specify commands for a different trigger set.

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_INDEX					0	0	0	T2_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_INDEX					0	0	0	T0_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-7. Commands List Control Register 1 (CLCR1)

Table 18-10. CLCR1 field descriptions

Field	Description
T3_INDEX	Trigger 3 Commands List first command address
T2_INDEX	Trigger 2 Commands List first command address
T1_INDEX	Trigger 1 Commands List first command address
T0_INDEX	Trigger 0 Commands List first command address

18.4.1.7 Commands List Control Register 2 (CLCR2)

The CLCR2 contains the command to be sent when a specified trigger is generated. It selects one of the 24 CLR commands as the first command when a trigger is generated and sends it to the ADC. CLCR1 and CLCR2 are functionally the same, but specify commands for a different trigger set.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T7_INDEX					0	0	0	T6_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T5_INDEX					0	0	0	T4_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-8. Commands List Control Register 2 (CLCR2)

Table 18-11. CLCR2 field descriptions

Field	Description
T7_INDEX	Trigger 7 Commands List first command address
T6_INDEX	Trigger 6 Commands List first command address
T5_INDEX	Trigger 5 Commands List first command address
T4_INDEX	Trigger 4 Commands List first command address

18.4.1.8 Trigger Handler Control Register 1 (THCR1)

The Trigger Handler Control Register 1 (THCR1) enables sending a trigger pulse to another module, such as a timer or the ADC. THCR1 and THCR2 are functionally the same, but specify different triggers.

See Figure 18-29 for more information.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	T3_E	T3_ETE	T3_T4E	T3_T3E	T3_T2E	T3_T1E	T3_ADCE	0	T2_E	T2_ETE	T2_T4E	T2_T3E	T2_T2E	T2_T1E	T2_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	T1_E	T1_ETE	T1_T4E	T1_T3E	T1_T2E	T1_T1E	T1_ADCE	0	T0_E	T0_ETE	T0_T4E	T0_T3E	T0_T2E	T0_T1E	T0_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-9. Trigger Handler Control Register 1 (THCR1)

Table 18-12. THCR1 field descriptions

Field	Description
T3_E	Trigger 3 enable. Enables all available outputs for Trigger 3. 0 Disabled. 1 Enabled.
T3_ETE	Trigger 3 External Trigger output enable. Enables EXT_TRG (shown in Figure 18-29). 0 Disabled. 1 Enabled.
T3_T4E	Trigger 3 Timer 4 output enable. No internal or external connection. 0 Disabled. 1 Enabled.
T3_T3E	Trigger 3 Timer 3 output enable. On CTU_0, enables ETIMER3_TRG (shown in Figure 18-29) to drive the eTimer2_AUX2 input. On CTU_1, enables ETIMER3_TRG to drive the eTimer 1_AUX3 input. 0 Disabled. 1 Enabled.
T3_T2E	Trigger 3 Timer 2 output enable. On CTU_0, enables ETIMER2_TRG (shown in Figure 18-29) to drive the eTimer1_AUX0 input. On CTU_1, enables ETIMER2_TRG to drive the eTimer 2_AUX1 input. 0 Disabled. 1 Enabled.
T3_T1E	Trigger 3 Timer 1 output enable. On CTU_0, enables ETIMER1_TRG (shown in Figure 18-29) to drive the eTimer0_AUX0 input. On CTU_1, enables ETIMER1_TRG to drive the eTimer 2_AUX0 input. 0 Disabled. 1 Enabled.
T3_ADCE	Trigger 3 ADC command output enable 0 Disabled. 1 Enabled.
T2_E	Trigger 2 enable. Enables all available outputs for Trigger 2. 0 Disabled. 1 Enabled.
T2_ETE	Trigger 2 External Trigger output enable. Enables EXT_TRG (shown in Figure 18-29). 0 Disabled. 1 Enabled.
T2_T4E	Trigger 2 Timer 4 output enable. No internal or external connection. 0 Disabled. 1 Enabled.
T2_T3E	Trigger 2 Timer 3 output enable. On CTU_0, enables ETIMER3_TRG (shown in Figure 18-29) to drive the eTimer 2_AUX2 input. On CTU_1, enables ETIMER3_TRG to drive the eTimer 1_AUX3 input. 0 Disabled. 1 Enabled.
T2_T2E	Trigger 2 Timer 2 output enable. On CTU_0, enables ETIMER2_TRG (shown in Figure 18-29) to drive the eTimer1_AUX0 input. On CTU_1, enables ETIMER2_TRG to drive the eTimer 2_AUX1 input. 0 Disabled. 1 Enabled.

Table 18-12. THCR1 field descriptions (continued)

Field	Description
T2_T1E	Trigger 2 Timer 1 output enable. On CTU_0, enables ETIMER1_TRG (shown in Figure 18-29) to drive the eTimer0_AUX0 input. On CTU_1, enables ETIMER1_TRG to drive the eTimer 2_AUX0 input. 0 Disabled. 1 Enabled.
T2_ADCE	Trigger 2 ADC command output enable 0 Disabled. 1 Enabled.
T1_E	Trigger 1 enable. Enables all available outputs for Trigger 1. 0 Disabled. 1 Enabled.
T1_ETE	Trigger 1 External Trigger output enable. Enables EXT_TRG (shown in Figure 18-29). 0 Disabled. 1 Enabled.
T1_T4E	Trigger 1 Timer 4 output enable. No internal or external connection. 0 Disabled. 1 Enabled.
T1_T3E	Trigger 1 Timer 3 output enable. On CTU_0, enables ETIMER3_TRG (shown in Figure 18-29) to drive the eTimer 2_AUX2 input. On CTU_1, enables ETIMER3_TRG to drive the eTimer 1_AUX3 input. 0 Disabled. 1 Enabled.
T1_T2E	Trigger 1 Timer 2 output enable. On CTU_0, enables ETIMER2_TRG (shown in Figure 18-29) to drive the eTimer1_AUX0 input. On CTU_1, enables ETIMER2_TRG to drive the eTimer 2_AUX1 input. 0 Disabled. 1 Enabled.
T1_T1E	Trigger 1 Timer 1 output enable. On CTU_0, enables ETIMER1_TRG (shown in Figure 18-29) to drive the eTimer0_AUX0 input. On CTU_1, enables ETIMER1_TRG to drive the eTimer 2_AUX0 input. 0 Disabled. 1 Enabled.
T1_ADCE	Trigger 1 ADC command output enable 0 Disabled. 1 Enabled.
T0_E	Trigger 0 enable. Enables all available outputs for Trigger 0. 0 Disabled. 1 Enabled.
T0_ETE	Trigger 0 External Trigger output enable. Enables EXT_TRG (shown in Figure 18-29). 0 Disabled. 1 Enabled.
T0_T4E	Trigger 0 Timer 4 output enable. No internal or external connection. 0 Disabled. 1 Enabled.

Table 18-12. THCR1 field descriptions (continued)

Field	Description
T0_T3E	Trigger 0 Timer 3 output enable. On CTU_0, enables ETIMER3_TRG (shown in Figure 18-29) to drive the eTimer 2_AUX2 input. On CTU_1, enables ETIMER3_TRG to drive the eTimer 1_AUX3 input. 0 Disabled. 1 Enabled.
T0_T2E	Trigger 0 Timer 2 output enable. On CTU_0, enables ETIMER2_TRG (shown in Figure 18-29) to drive the eTimer1_AUX0 input. On CTU_1, enables ETIMER2_TRG to drive the eTimer 2_AUX1 input. 0 Disabled. 1 Enabled.
T0_T1E	Trigger 0 Timer 1 output enable. On CTU_0, enables ETIMER1_TRG (shown in Figure 18-29) to drive the eTimer0_AUX0 input. On CTU_1, enables ETIMER1_TRG to drive the eTimer 2_AUX0 input. 0 Disabled. 1 Enabled.
T0_ADCE	Trigger 0 ADC command output enable 0 Disabled. 1 Enabled.

18.4.1.9 Trigger Handler Control Register 2 (THCR2)

The Trigger Handler Control Register 2 (THCR2) enables sending a trigger pulse to another module, such as a timer or the ADC. THCR1 and THCR2 are functionally the same, but specify different triggers.

See Figure 18-29 for more information.

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	T7_E	T7_ETE	T7_T4E	T7_T3E	T7_T2E	T7_T1E	T7_ADCE	0	T6_E	T6_ETE	T6_T4E	T6_T3E	T6_T2E	T6_T1E	T6_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	T5_E	T5_ETE	T5_T4E	T5_T3E	T5_T2E	T5_T1E	T5_ADCE	0	T4_E	T4_ETE	T4_T4E	T4_T3E	T4_T2E	T4_T1E	T4_ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-10. Trigger Handler Control Register 2 (THCR2)

Table 18-13. THCR2 field descriptions

Field	Description
T7_E	Trigger 7 enable 0 Disabled. 1 Enabled.
T7_ETE	Trigger 7 External Trigger output enable 0 Disabled. 1 Enabled.
T7_T4E	Trigger 7 Timer 4 output enable 0 Disabled. 1 Enabled.
T7_T3E	Trigger 7 Timer 3 output enable 0 Disabled. 1 Enabled.
T7_T2E	Trigger 7 Timer 2 output enable 0 Disabled. 1 Enabled.
T7_T1E	Trigger 7 Timer 1 output enable 0 Disabled. 1 Enabled.
T7_ADCE	Trigger 7 ADC command output enable 0 Disabled. 1 Enabled.
T6_E	Trigger 6 enable 0 Disabled. 1 Enabled.
T6_ETE	Trigger 6 External Trigger output enable 0 Disabled. 1 Enabled.
T6_T4E	Trigger 6 Timer 4 output enable 0 Disabled. 1 Enabled.
T6_T3E	Trigger 6 Timer 3 output enable 0 Disabled. 1 Enabled.
T6_T2E	Trigger 6 Timer 2 output enable 0 Disabled. 1 Enabled.
T6_T1E	Trigger 6 Timer 1 output enable 0 Disabled. 1 Enabled.
T6_ADCE	Trigger 6 ADC command output enable 0 Disabled. 1 Enabled.

Table 18-13. THCR2 field descriptions (continued)

Field	Description
T5_E	Trigger 5 enable 0 Disabled. 1 Enabled.
T5_ETE	Trigger 5 External Trigger output enable 0 Disabled. 1 Enabled.
T5_T4E	Trigger 5 Timer 4 output enable 0 Disabled. 1 Enabled.
T5_T3E	Trigger 5 Timer 3 output enable 0 Disabled. 1 Enabled.
T5_T2E	Trigger 5 Timer 2 output enable 0 Disabled. 1 Enabled.
T5_T1E	Trigger 5 Timer 1 output enable 0 Disabled. 1 Enabled.
T5_ADCE	Trigger 5 ADC command output enable 0 Disabled. 1 Enabled.
T4_E	Trigger 4 enable 0 Disabled. 1 Enabled.
T4_ETE	Trigger 4 External Trigger output enable 0 Disabled. 1 Enabled.
T4_T4E	Trigger 4 Timer 4 output enable 0 Disabled. 1 Enabled.
T4_T3E	Trigger 4 Timer 3 output enable 0 Disabled. 1 Enabled.
T4_T2E	Trigger 4 Timer 2 output enable 0 Disabled. 1 Enabled.
T4_T1E	Trigger 4 Timer 1 output enable 0 Disabled. 1 Enabled.
T4_ADCE	Trigger 4 ADC command output enable 0 Disabled. 1 Enabled.

18.4.1.10 Commands List Register *x* (CLR*x*)

The ADC commands list registers (CLR*x*) provide for two different types of commands:

- Channel conversion commands
- Self-test commands

When used for channel conversion commands, CLR*x* has two formats. Figure 18-11 shows the CLR*x* in single conversion mode. Figure 18-12 shows the CLR*x* in dual conversion mode. The CMS bit selects the mode. When CMS = 0, CLR*x* is in single conversion mode (Figure 18-11). When CMS = 1, CLR*x* is in dual conversion mode (Figure 18-12). Table 18-16 lists the addresses for CLR*x*.

Address: SeeTable 18-16. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC	CMS	FIFO			ST	0	0	0	SU	0	CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-11. Commands List Register *x* (CLR*x*)—Single conversion mode

Address: SeeTable 18-16. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC	CMS	FIFO				CH_B				0	CH_A			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-12. Commands List Register *x* (CLR*x*)—Dual conversion mode

Table 18-14. CLR*x* field descriptions (channel conversion commands)

Field	Description
CIR	Command Interrupt Request 0 Disabled. 1 Enabled.
LC	Last command. The CTU state machine triggers the ADC conversions configured into the command list until it finds a command with the LC bit set to one (this command is not executed). 0 Not last. 1 Last. **Note:** This bit may be called FC or LC depending on the device, but the functionality is identical.
CMS	Conversion mode selection 0 Single conversion mode. See Figure 18-11. 1 Dual conversion mode. See Figure 18-12.
FIFO	FIFO for ADC unit A/B
ST	Self-test mode **Note:** When CMS = 1, this bit is combined with the CH_B field.
SU	Selection Unit (Single conversion mode only. See Figure 18-11.) 0 ADC unit A selected. 1 ADC unit B selected.
CH	ADC unit channel number (Single conversion mode only. See Figure 18-11.)

Table 18-14. CLR*x* field descriptions (channel conversion commands) (continued)

Field	Description
CH_B	ADC unit B channel number (Dual conversion mode only. See Figure 18-12.)
CH_A	ADC unit A channel number (Dual conversion mode only. See Figure 18-12.)

Address: SeeTable 18-16. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	LC	ST1	CMS	SU	0	ST0	ALG		BSIZE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-13. Commands List Register *x* (CLR*x*)—Self-test commands

Table 18-15. CLR*x* field descriptions (self-test commands)

Field	Description
CIR	Command Interrupt Request bit 0 Disabled. 1 Enabled.
LC	Last command bit. The CTU state machine triggers the ADC conversions configured into the command list until it finds a command with the LC bit set to one (this command is not executed). 0 Not last. 1 Last. **Note:** This bit may be called FC or LC depending on the device, but the functionality is identical.
ST1	Self-test mode 1. **Note:** This bit must be 0 to send a self-test command.
CMS	Conversion mode selection 0 Single conversion mode. See Figure 18-11. 1 Dual conversion mode. See Figure 18-12.
SU	Selection Unit bit 0 ADC unit A selected. 1 ADC unit B selected.
ST0	Self-test mode 0 **Note:** This bit must be 1 to send a self-test command.
ALG	Algorithm scheduled 00 Algorithm S. 01 Algorithm RC. 10 Algorithm C. 11 Full algorithm (S + RC + C).
BSIZE	Burst size of the algorithm iteration. 0 Single step execution. *n* N + 1 step execution with one trigger.

Table 18-16. CLR*x* addresses[1]

Register	Address	Register	Address	Register	Address	Register	Address
CLR1	0x002C	CLR7	0x0038	CLR13	0x0044	CLR19	0x0050
CLR2	0x002E	CLR8	0x003A	CLR14	0x0046	CLR20	0x0052
CLR3	0x0030	CLR9	0x003C	CLR15	0x0048	CLR21	0x0054
CLR4	0x0032	CLR10	0x003E	CLR16	0x004A	CLR22	0x0056
CLR5	0x0034	CLR11	0x0040	CLR17	0x004C	CLR23	0x0058
CLR6	0x0036	CLR12	0x0042	CLR18	0x004E	CLR24	0x005A

1 Offset from CTU module base.

18.4.1.11 FIFO DMA Control Register (FDCR)

The FIFO DMA Control Register (FDCR) controls the DMA and the EMPTY interrupts for the FIFO.

Address: Base + 0x006C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EMPTY_clear3	EMPTY_clear2	EMPTY_clear1	EMPTY_clear0	0	0	0	0	DE3	DE2	DE1	DE0
W					w1c	w1c	w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-14. FIFO DMA Control Register (FDCR)

Table 18-17. FDCR field descriptions

Name	Description
EMPTY_clear*n*	These bits reset the interrupt related to the empty flag bits (FST[EMP*x*]) even if the FIFO is still empty. These bits are set by hardware with a positive edge on the empty flag, and are cleared by writing 1 in this field or writing data to the FIFO.
DE*n*	These bits enable DMA for the FIFO*n*.

18.4.1.12 FIFO Control Register (FCR)

The FIFO Control Register (FCR) enables the FIFO interrupts.

Address: Base + 0x0070 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR_EN3	OF_EN3	EMPTY_EN3	FULL_EN3	OR_EN2	OF_EN2	EMPTY_EN2	FULL_EN2	OR_EN1	OF_EN1	EMPTY_EN1	FULL_EN1	OR_EN0	OF_EN0	EMPTY_EN0	FULL_EN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-15. FIFO Control Register (FCR)

Table 18-18. FCR field descriptions

Field	Description
OR_EN3	FIFO 3 overrun interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OF_EN3	FIFO 3 threshold overflow interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
EMPTY_EN3	FIFO 3 empty interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
FULL_EN3	FIFO 3 full interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OR_EN2	FIFO 2 overrun interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OF_EN2	FIFO 2 threshold overflow interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
EMPTY_EN2	FIFO 2 empty interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
FULL_EN2	FIFO 2 Full interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OR_EN1	FIFO 1 overrun interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.

Table 18-18. FCR field descriptions (continued)

Field	Description
OF_EN1	FIFO 1 threshold overflow interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
EMPTY_EN1	FIFO 1 empty interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
FULL_EN1	FIFO 1 full interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OR_EN0	FIFO 0 overrun interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
OF_EN0	FIFO 0 threshold overflow interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
EMPTY_EN0	FIFO 0 empty interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.
FULL_EN0	FIFO 0 full interrupt enable 0 Interrupt is disabled. 1 Interrupt is enabled.

18.4.1.13 FIFO Threshold Register (FTH)

The FIFO Threshold Register (FTH) defines the thresholds at which the FIFOs generate an interrupt when the number of elements in a FIFO exceeds the values present in this register.

Address: Base + 0x0074 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TH3								TH2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TH1								TH0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-16. FIFO Threshold Register (FTH)

Table 18-19. FTX field descriptions

Field	Description
TH3	FIFO 3 threshold
TH2	FIFO 2 threshold

Table 18-19. FTX field descriptions (continued)

Field	Description
TH1	FIFO 1 threshold
TH0	FIFO 0 threshold

18.4.1.14 FIFO Status Register (FST)

The FIFO Status Register (FST) provides status bits for the FIFOs.

NOTE

The CTU FIFO order is non-deterministic. FIFO order may be reversed under certain conditions.

Address: Base + 0x007C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR3	OF3	EMP3	FULL3	OR2	OF2	EMP2	FULL2	OR1	OF1	EMP1	FULL1	OR0	OF0	EMP0	FULL0
W	w1c				w1c				w1c				w1c			
Reset	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0

Figure 18-17. FIFO Status Register (FST)

Table 18-20. FST field descriptions

Field	Description
OR3	FIFO 3 overrun flag. This bit is set to 1 if a write operation occurs when the corresponding FIFO full flag (FST[FULL3]) is set. Write a 1 to this bit to clear it. 0 FIFO 3 overrun condition has not occurred. 1 FIFO 3 overrun condition has occurred.
OF3	FIFO 3 threshold overflow flag. This bit is set to 1 if the number of words in FIFO 3 is higher than the value set in the FIFO Threshold Register (FTH[TH3]). 0 FIFO 3 overflow condition has not occurred. 1 FIFO 3 overflow condition has occurred.
EMP3	FIFO 3 empty flag. This bit is set to 1 if the FIFO 3 is empty. 0 FIFO 3 empty condition has not occurred. 1 FIFO 3 empty condition has occurred.
FULL3	FIFO 3 full flag. This bit is set to 1 if the FIFO 3 is full. 0 FIFO 3 full condition has not occurred. 1 FIFO 3 full condition has occurred.

Table 18-20. FST field descriptions (continued)

Field	Description
OR2	FIFO 2 overrun flag. This bit is set to 1 if a write operation occurs when the corresponding FIFO full flag (FST[FULL2]) is set. Write a 1 to this bit to clear it. 0 FIFO 2 overrun condition has not occurred. 1 FIFO 2 overrun condition has occurred.
OF2	FIFO 2 threshold overflow flag. This bit is set to 1 if the number of words in FIFO 2 is higher than the value set in the FIFO Threshold Register (FTH[TH2]). 0 FIFO 2 overflow condition has not occurred. 1 FIFO 2 overflow condition has occurred.
EMP2	FIFO 2 empty flag. This bit is set to 1 if the FIFO 2 is empty. 0 FIFO 2 empty condition has not occurred. 1 FIFO 2 empty condition has occurred.
FULL2	FIFO 2 full flag. This bit is set to 1 if the FIFO 2 is full. 0 FIFO 2 full condition has not occurred. 1 FIFO 2 full condition has occurred.
OR1	FIFO 1 overrun flag. This bit is set to 1 if a write operation occurs when the corresponding FIFO full flag (FST[FULL1]) is set. Write a 1 to this bit to clear it. 0 FIFO 1 overrun condition has not occurred. 1 FIFO 1 overrun condition has occurred.
OF1	FIFO 1 threshold overflow flag. This bit is set to 1 if the number of words in FIFO 1 is higher than the value set in the FIFO Threshold Register (FTH[TH1]). 0 FIFO 1 overflow condition has not occurred. 1 FIFO 1 overflow condition has occurred.
EMP1	FIFO 1 empty flag. This bit is set to 1 if the FIFO 1 is empty. 0 FIFO 1 empty condition has not occurred. 1 FIFO 1 empty condition has occurred.
FULL1	FIFO 1 full flag. This bit is set to 1 if the FIFO 1 is full. 0 FIFO 1 full condition has not occurred. 1 FIFO 1 full condition has occurred.
OR0	FIFO 0 overrun flag. This bit is set to 1 if a write operation occurs when the corresponding FIFO full flag (FST[FULL0]) is set. Write a 1 to this bit to clear it. 0 FIFO 0 overrun condition has not occurred. 1 FIFO 0 overrun condition has occurred.
OF0	FIFO 0 threshold overflow flag. This bit is set to 1 if the number of words in FIFO 0 is higher than the value set in the FIFO Threshold Register (FTH[TH0]). 0 FIFO 0 overflow condition has not occurred. 1 FIFO 0 overflow condition has occurred.
EMP0	FIFO 0 empty flag. This bit is set to 1 if the FIFO 0 is empty. 0 FIFO 0 empty condition has not occurred. 1 FIFO 0 empty condition has occurred.
FULL0	FIFO 0 full flag. This bit is set to 1 if the FIFO 0 is full. 0 FIFO 0 full condition has not occurred. 1 FIFO 0 full condition has occurred.

18.4.1.15 FIFO Right-Aligned Data *x* Register (FR*x*)

The FIFO Right-Aligned Data *x* Registers (FR*x*) store data coming from the ADC.

Address: Base + 0x0080 (FR0) Base + 0x0088 (FR2)
Base + 0x0084 (FR1) Base + 0x008C (FR3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	ADC	N_CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DATA											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-18. FIFO Right-aligned Data *x* Register (FR*x*)

Table 18-21. FR*x* field descriptions

Field	Description
ADC	This bit indicates from which ADC the data is coming. For CTU_0, these values apply: 0 The data is coming from ADC 1. 1 The data is coming from ADC 0. For CTU_1, these values apply: 0 The data is coming from ADC 3. 1 The data is coming from ADC 2.
N_CH	Number of stored channel.
DATA	Data of stored channel. Each read of this field deletes the current value and reloads the new pointed value.

18.4.1.16 FIFO Signed Left-Aligned Data *x* Register (FL*x*)

The FIFO Signed Left-Aligned Data *x* Registers (FL*x*) store data coming from the ADC.

Address: Base + 0x00A0 (FL0) Base + 0x00A8 (FL2)
Base + 0x00A4 (FL1) Base + 0x00AC (FL3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	ADC	N_CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	DATA												0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-19. FIFO Signed Left Aligned Data *x* Register (FL*x*)

Table 18-22. FL*x* field descriptions

Field	Description
ADC	This bit indicates from which ADC the data is coming. For CTU_0, these values apply: 0 The data is coming from ADC 1. 1 The data is coming from ADC 0. For CTU_1, these values apply: 0 The data is coming from ADC 3. 1 The data is coming from ADC 2.
N_CH	Number of stored channel.
DATA	Data of stored channel. Each read of this field deletes the current value and reloads the new pointed value.

NOTE

With a full FIFO, if concurrent FIFO read and write operations occur, then the order of the FIFO is not correct.

For example, FIFO is full and contains data A,B,C,D. Then there are POP and a PUSH requests in the same clock cycle. After the PUSH and POP operations instead of correct data B,C,D,E, the FIFO contains the data B,C,E,D. Data A is pushed out correctly, but data E and D are swapped.

Application software can detect the swap between E and D by reading the CTU.FLx.ADC and CTU.FLx.N_CH fields, unless E and D refer to the same ADC and same channel.

To reduce the risk of this issue 2 suggestions are given:

1) Lower the FIFO threshold to less than the size of the FIFO (probability is reduced, but can't be fully excluded).

2) Forbid 2 consecutive conversions from the same ADC and channel source to allow swap detection by reading the CTU.FLx.ADC and CTU.FLx.N_CH fields.

18.4.1.17 Cross Triggering Unit Error Flag Register (CTUEFR)

The CTUEFR manages the errors generated by the CTU.

Address: Base + 0x00C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	CS	ET_OE	ERROR_compare	T4_OE	T3_OE	T2_OE	T1_OE	ADC_OE	TGS_OSM	MRS_O	ICE	SM_TO	MRS_RE
W				w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-20. Cross Triggering Unit Error Flag Register (CTUEFR)

Table 18-23. CTUEFR field descriptions

Field	Description
CS	CS Counter status. This bit resets the self-test counter. 0 All the counter are at reset value. 1 One of the counter used in self-test mode is not in the reset state.
ET_OE	External trigger generation overrun error 0 Error has not occurred. 1 Error has occurred.
ERROR_compare	This bit is set if the counter reaches the TGSCCR.
T4_OE	ETIMER4 trigger generation overrun error 0 Error has not occurred. 1 Error has occurred.
T3_OE	ETIMER3 trigger generation overrun error 0 Error has not occurred. 1 Error has occurred.
T2_OE	ETIMER2 trigger generation overrun error 0 Error has not occurred. 1 Error has occurred.
T1_OE	ETIMER1 trigger generation overrun error 0 Error has not occurred. 1 Error has occurred.
ADC_OE	ADC command generation overrun error 0 Error has not occurred. 1 Error has occurred.
TGS_OSM	TGS overrun in sequential mode 0 Error has not occurred. 1 Error has occurred.
MRS_O	Master Reload Signal (MRS) overrun 0 Overrun has not occurred. 1 Error has occurred.

Table 18-23. CTUEFR field descriptions (continued)

Field	Description
ICE	Invalid command error 0 Error has not occurred. 1 Error has occurred. **Note:** If software is incorrectly programmed in Dual Conversion mode, such as by programming one or more ADC commands on the same physical channel (a channel that is shared between two ADCs, for example, Channel 11 on ADC_0 and ADC_1), the Invalid Command Error is not detected and this bit is not set. Software must ensure that in case of Dual Conversion Mode, if the channel number is the same for both ADCs then this number should not represent a shared ADC channel.
SM_TO	Trigger overrun (more than 8 EV) in TGS sequential mode 0 Overrun has not occurred. 1 Overrun has occurred.
MRS_RE	Master Reload Signal (MRS) reload error 0 Error has not occurred. 1 Error has occurred.

18.4.1.18 Cross Triggering Unit Interrupt Flag Register (CTUIFR)

The Cross Triggering Unit Interrupt Flag Register (CTUIFR) manages the CTU interrupts.

Address: Base + 0x00C2 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	Safety_error_B	Safety_error_A	ADC_I	T7_I	T6_I	T5_I	T4_I	T3_I	T2_I	T1_I	T0_I	MRS_I
W					w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-21. Cross Triggering Unit Interrupt Flag Register (CTUIFR)

Table 18-24. CTUIFR field descriptions

Field	Description
Safety_error_B	When this bit is set, the slice time between the start of conversion and the end of conversion is in the expected range. For CTU_0, this bit applies to ADC 1. For CTU_1, this bit applies to ADC 3.
Safety_error_A	When this bit is set, the slice time between the start of conversion and the end of conversion is in the expected range. For CTU_0, this bit applies to ADC 0. For CTU_1, this bit applies to ADC 2.
ADC_I	ADC command interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T7_I	Trigger 7 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 18-24. CTUIFR field descriptions (continued)

Field	Description
T6_I	Trigger 6 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T5_I	Trigger 5 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T4_I	Trigger 4 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T3_I	Trigger 3 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T2_I	Trigger 2 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T1_I	Trigger 1 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
T0_I	Trigger 0 interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.
MRS_I	Master Reload Signal (MRS) Interrupt flag. 0 Interrupt has not occurred. 1 Interrupt has occurred.

18.4.1.19 Cross Triggering Unit Interrupt/DMA Register (CTUIR)

The Cross Triggering Unit Interrupt/DMA Register (CTUIR) enables the trigger interrupts.

Address: Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_IE	T6_IE	T5_IE	T4_IE	T3_IE	T2_IE	T1_IE	T0_IE	0	0	Saf_cnt_b_en	Saf_cnt_a_en	DMA_DE	MRS_DMAE	MRS_IE	IEE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-22. Cross Triggering Unit Interrupt/DMA Register (CTUIR)

Table 18-25. CTUIR field description

Field	Description
T7_IE	Trigger 7 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T6_IE	Trigger 6 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T5_IE	Trigger 5 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T4_IE	Trigger 4 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T3_IE	Trigger 3 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T2_IE	Trigger 2 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T1_IE	Trigger 1 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
T0_IE	Trigger 0 interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
Saf_cnt_b_en	If this bit is set, the counter used to check the conversion time is enabled.
Saf_cnt_a_en	If this bit is set, the counter used to check the conversion time is enabled.
DMA_DE	If this bit is set, a DMA Done is like a write in the CTUCR[GRE] bit. If this bit is cleared, you have to write the GRE bit (if something is changed in the double buffered register) otherwise an error occurs.
MRS_DMAE	DMA transfer enable on Master Reload Signal (MRS) occurrence if CTUCR[GRE] is set.
MRS_IE	Master Reload Signal (MRS) interrupt enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.
IEE	Interrupt error enable. 0 Interrupt is not enabled. 1 Interrupt is enabled.

18.4.1.20 Control On-Time Register (COTR)

The Control On-Time Register (COTR) defines the on-time of the external trigger.

Address: Base + 0x00C6 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	COTR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-23. Control On-Time Register (COTR)

Table 18-26. COTR field description

Field	Description
COTR	Control On-Time and Guard Time for external trigger

18.4.1.21 Cross Triggering Unit Control Register (CTUCR)

The Cross Triggering Unit Control Register (CTUCR) manages the register reloads and generation of the software triggers.

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_SG	T6_SG	T5_SG	T4_SG	T3_SG	T2_SG	T1_SG	T0_SG	CTU_ADC_R	CTU_ODIS	DFE	CGRE	FGRE	MRS_SG	GRE	TGSISR_RE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-24. Cross Triggering Unit Control Register (CTUCR)

Table 18-27. CTUCR field descriptions

Field	Description
T7_SG	Trigger 7 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T6_SG	Trigger 6 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T5_SG	Trigger 5 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T4_SG	Trigger 4 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.

Table 18-27. CTUCR field descriptions (continued)

Field	Description
T3_SG	Trigger 3 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T2_SG	Trigger 2 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T1_SG	Trigger 1 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
T0_SG	Trigger 0 software generated You may program this field to 1, but it will always read 0. 0 Trigger has not been generated. 1 Trigger has been generated.
CTU_ADC_R	CTU/ADC state machine reset You may program this field to 1, but it will always read 0. 0 Reset has not been generated. 1 Reset has been generated.
CTU_ODIS	CTU output disable 0 CTU output is enabled. 1 CTU output is disabled.
DFE	Digital filter enable You may read this field and program it to 1; other operations are not allowed. 0 Digital filter is disabled. 1 Digital filter is enabled.
CGRE	Clear GRE bit 0 The GRE bit remains in its current state. 1 The GRE bit is cleared when this bit is set.
FGRE	Flag GRE 0 General Reload has not occurred. 1 General Reload has occurred.
MRS_SG	Master Reload Signal (MRS) software generated You may program this field to 1, but it will always read 0. 0 MRS has not been generated. 1 MRS has been generated.
GRE	General reload enable You may read this field and program it to 1; other operations are not allowed. 0 General reload is disabled. 1 General reload is enabled.
TGSISR_RE	TGS input selection register reload enable 0 Register reload is disabled. 1 Register reload is enabled.

18.4.1.22 Cross Triggering Unit Digital Filter Register (CTUDF)

The Cross Triggering Unit Digital Filter Register (CTUDF) specifies the value for the number of clockcycles for an external trigger not to be filtered.

Address: Base + 0x00CA Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	FILTERVALUE							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-25. Cross Triggering Unit Digital Filter Register (CTUDF)

Table 18-28. CTUDF field descriptions

Field	Description
FILTERVALUE	Digital Filter value The external signal is considered at 1 if it is latched at 1 at a time of *n* + 1, and is considered at 0 if it is latched at 0 at a time of *n* + 1, where *n* is the value in this bitfield.

18.4.1.23 Cross Triggering Unit Expected Value A Register (CTU_EXPECTED_A)

The Cross Triggering Unit Expected Value A Register (CTU_EXPECTED_A) holds the number of system clock (SYS_CLK) cycles needed for the conversion time. On CTU_0, this is the conversion time for ADC_0. On CTU_1, this is the time for ADC_2.

Address: Base + 0x00CC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VALUE															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 18-26. Cross Triggering Unit Expected Value A Register (CTU_EXPECTED_A)

Table 18-29. CTU_EXPECTED_A field descriptions

Field	Description
VALUE	This value is the number of system clock cycles needed for the conversion time.

18.4.1.24 Cross Triggering Unit Expected Value B Register (CTU_EXPECTED_B)

The Cross Triggering Unit Expected Value B Register (CTU_EXPECTED_B) holds the number of system clock cycles needed for the conversion time. On CTU_0, this is the conversion time for ADC_1. On CTU_1, this is the time for ADC_3.

Address: Base + 0x00CE Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VALUE															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 18-27. Cross Triggering Unit Expected Value B Register (CTU_EXPECTED_B)

Table 18-30. CTU_EXPECTED_B field descriptions

Field	Description
VALUE	This value is the number of system clock cycles needed for the conversion time.

18.4.1.25 Cross Triggering Unit Count Range Register (CTU_CNT_RANGE)

The Cross Triggering Unit Count Range Register (CTU_CNT_RANGE) is used with the CTU_EXPECTED registers to define a range for the number of system clock (SYS_CLK) cycles needed for a conversion that is not defined with a precision of one clock cycle.

Address: Base + 0x00D2 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VALUE							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-28. Cross Triggering Unit Count Range Register (CTU_CNT_RANGE)

Table 18-31. CTU_CNT_RANGE field descriptions

Field	Description
VALUE	Bits masked by this register field are ignored while comparing internal up counter to the CTU_EXPECTED_A/B register.

18.5 Functional description

18.5.1 Interaction with other peripherals

Figure 18-29 shows how the CTUs interact with these peripherals:

- CTU_0
  - ADC_0
  - ADC_1
  - DSPI_1
  - eTimer_0
  - eTimer_1
  - FlexPWM_0
  - FlexRay

- CTU_1
  - ADC_2
  - ADC_3
  - eTimer_1
  - eTimer_2
  - FlexPWM_1
  - PDI

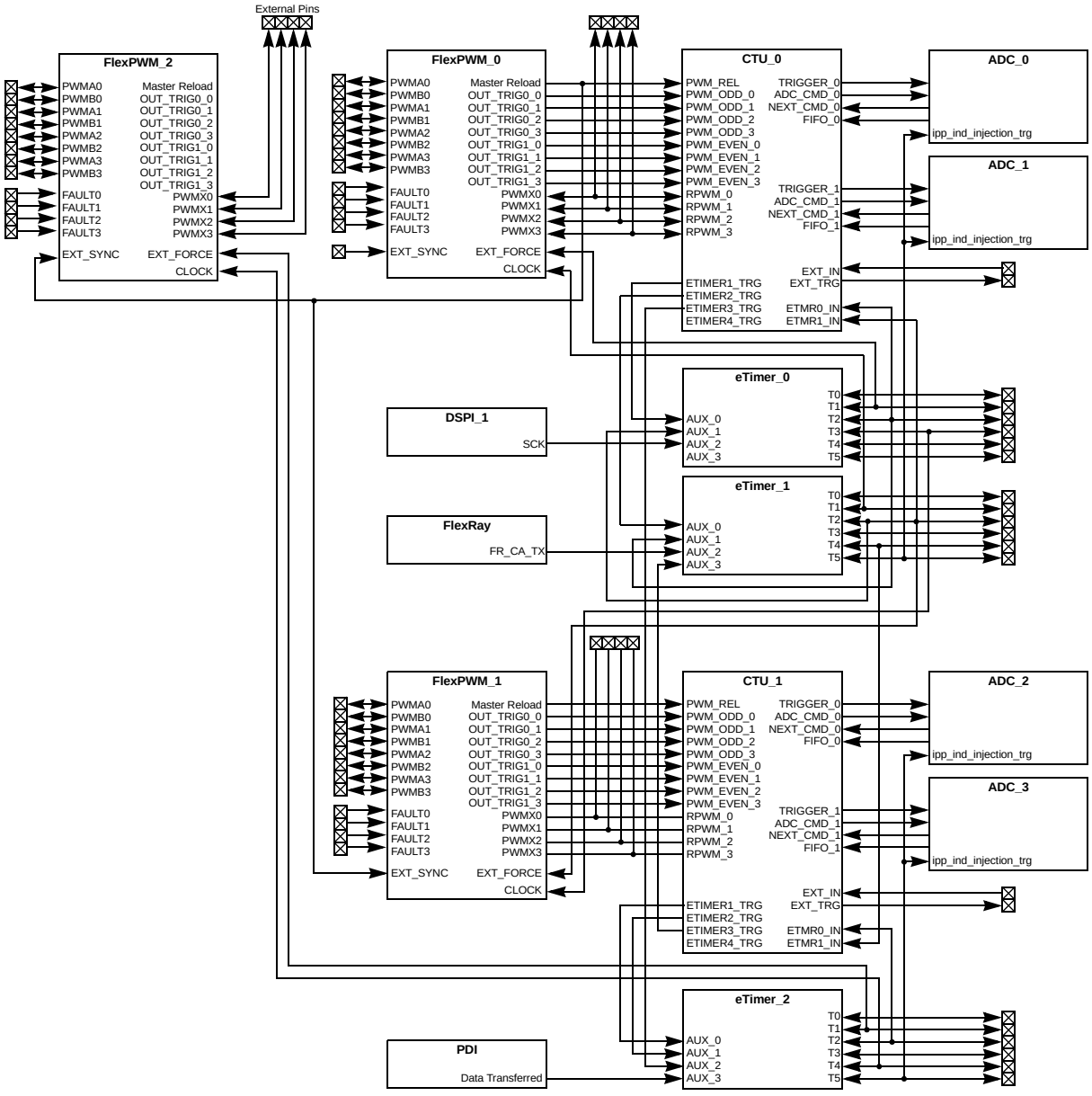


Figure 18-29. CTU interaction with other peripherals

18.5.2 Trigger events features

The trigger generator sub unit (TGS) has the capability to generate eight trigger events. Each trigger event has the following characteristics:

- The generation of trigger events is sequential in time.
- The Triggers List uses eight 16-bit double-buffered registers (T*x*CR).
- On each Master Reload Signal (MRS), the new Triggers List is loaded.
- The Triggers List is reloaded on an MRS occurrence only if the CTUCR[GRE] reload enable bit is set.

18.5.3 Trigger Generator Sub Unit (TGS)

The TGS has the following two modes:

- Triggered mode: each event source for the incoming signals can generate up to eight trigger event outputs. For the ADC, a commands list is entered by the CPU, and each event source can generate up to eight commands or streams of commands.
- Sequential mode: each event source for the incoming signals can generate one trigger event output, the next event source generates the next trigger event output, and so on in a predefined sequence. For the ADC, a commands list is entered by the CPU and the sequence of the selected incoming trigger events generate commands or stream of commands.

The TGS mode can be toggled between Triggered and Sequential mode with the TGSCR[TGS_M] bit.

18.5.4 TGS in Triggered Mode

The structure of the TGS in triggered mode is shown in Figure 18-30.

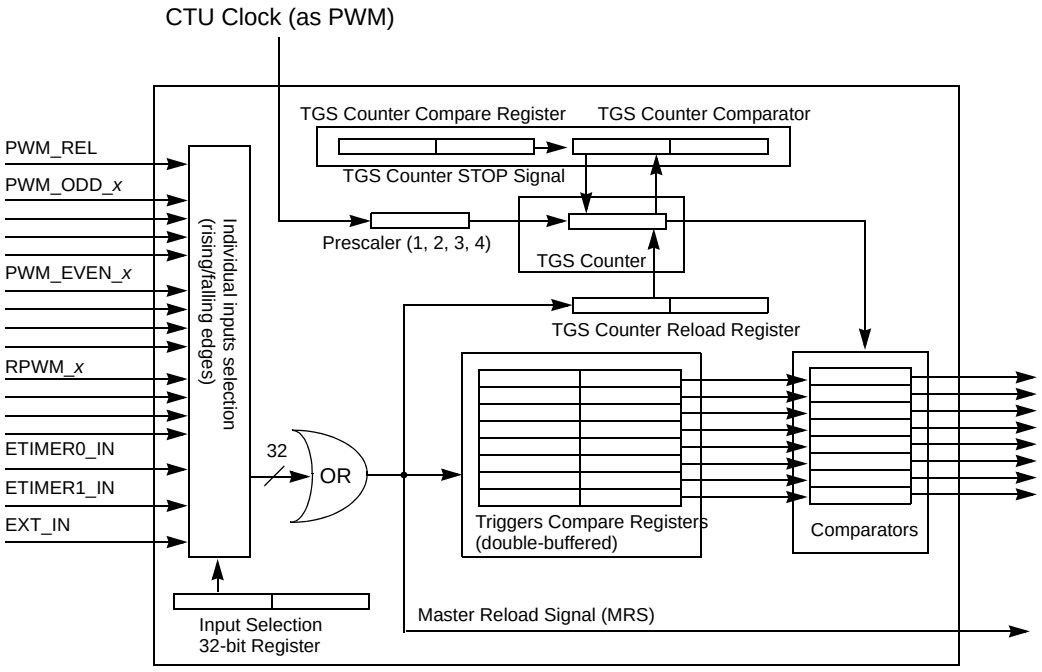


Figure 18-30. TGS in triggered mode

The TGS has 16 input signals that are selected by the input selection register (TGSISR), selecting the states inactive, rising, falling or both. Depending on the selection, as many as 32 input events can be enabled. These signals are OR-ed in order to generate the Master Reload Signal (MRS). The MRS, at the beginning of the control cycle “N” (defined by the MRS occurrence), pre-loads the TGS counter register, using the pre-load value written into the double-buffered register (TGSCRR), during the control cycle “N – 1”, and reloads all the double-buffered registers (Trigger Compare registers, TGSCR, TGSCRR itself, etc.).

The triggers list registers (T*x*CR) consist of eight compare registers. Each triggers list register is associated with a comparator. On reload (MRS occurrence), the comparators are disabled. One TGS clock cycle is necessary to enable them and to start the counting. The MRS is output together with individual trigger signals. The MRS can be performed by hardware or by software. The CTUCR[MRS_SG] bit, if set to 1, generates equivalent software MRS (i.e., resets/reloads the TGS Counter and reloads all double-buffered registers). This bit is cleared by each hardware or software MRS occurrence.

The TGS counter compare register and the TGS counter comparator are used to stop the TGS counter when it reaches the value stored in the TGS counter compare register before an MRS occurs.

The prescaler for TGS and SU can be 1,2,3,4 (PRES bits in the TGS Control Register).

An example timing for the TGS in Triggered Mode is shown in Figure 18-31. The red arrows indicate the MRS occurrences, while the black arrows indicate the trigger event occurrences, with the relevant delay in respect to the last MRS occurrence.

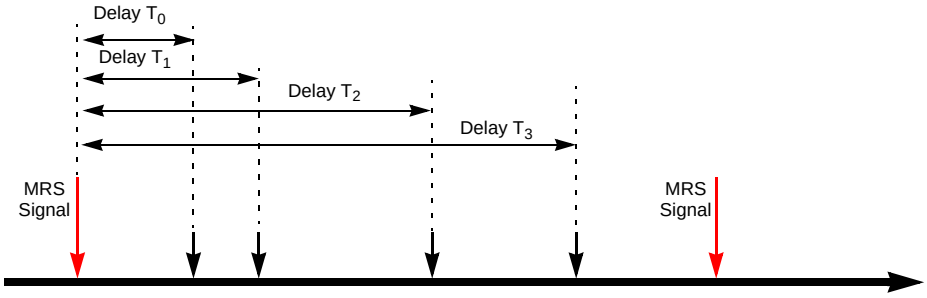


Figure 18-31. Example timing for TGS in triggered mode

18.5.5 TGS in sequential mode

The structure of the TGS in sequential mode is shown in Figure 18-32.

The 32 input events (16 signals with two edges for signal), which can be individually enabled, are OR-ed in order to generate the event signal (ES). The ES enables the reload of the TGS counter register and pilots the 3-bits counter, in order to select the next active trigger. One of the 32 input events can be selected, through the TGSCR[MRS_SM], to be the MRS, that enables the reload of the triggers list and resets the 3-bit counter (incoming events counter), i.e., the MRS is the signal linked with the control cycle defined as the time window between two consecutive MRSes. In this mode, each incoming event sequentially enables only one trigger event through the 3-bit counter and the MUX. The MUX is a selection switch which enables, according to the number of event signals occurred, only one of the eight trigger signals to the scheduler sub unit. Sequences of up to eight trigger events can be supported within this control cycle.

For the other features see the previous paragraphs.

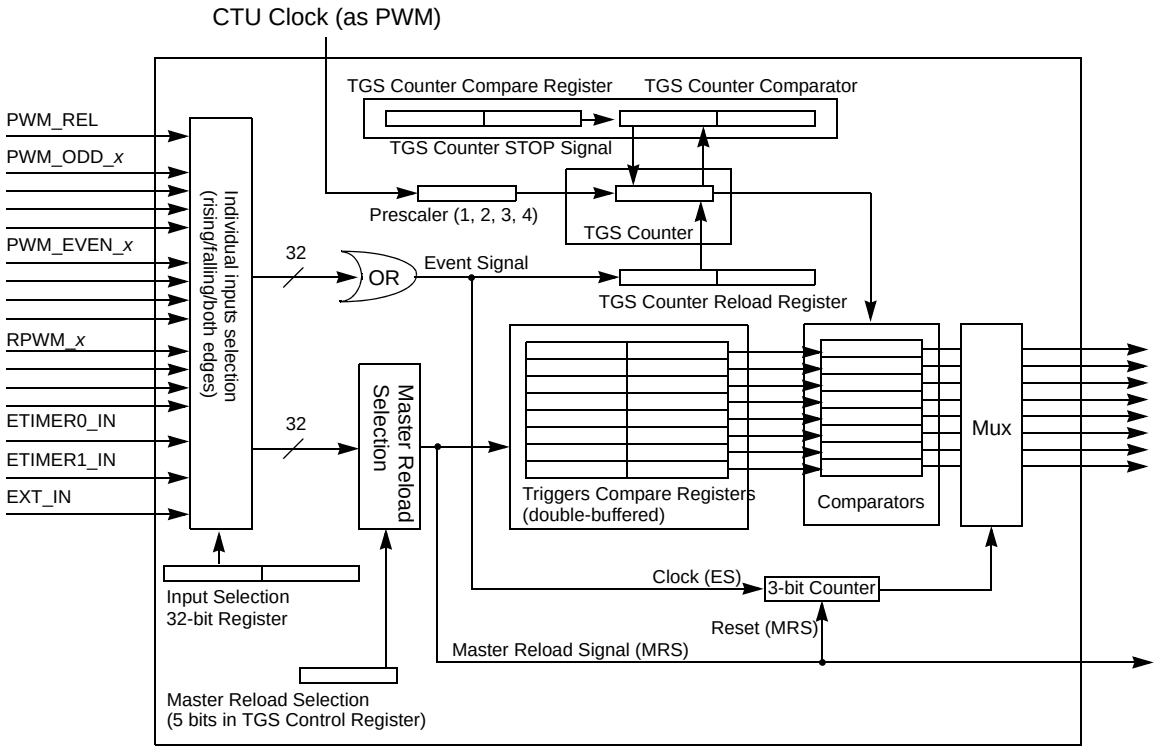


Figure 18-32. TGS in sequential mode

An example timing diagram for TGS in sequential mode is shown in Figure 18-33. The red arrows indicate the MRS occurrences and ES occurrences, while the black arrows indicate the trigger event occurrences with the relevant delay in respect to the ES occurrence. The first red arrow indicates the first ES occurrence, which is also the MRS.

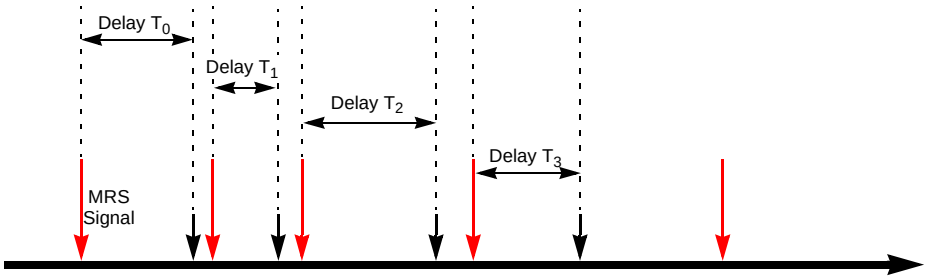


Figure 18-33. Example timing for TGS in sequential mode

18.5.6 TGS counter

The TGS counter is able to count from negative to positive, i.e., from 0x8000 to 0x7FFF. Figure 18-34 shows examples in order to explain the TGS counter counts. The compare operation to stop the TGS counter is not enabled during the first counting cycle, in order to allow the counting, if the value of the TGSCRR is the same as the value of the TGSCCR.

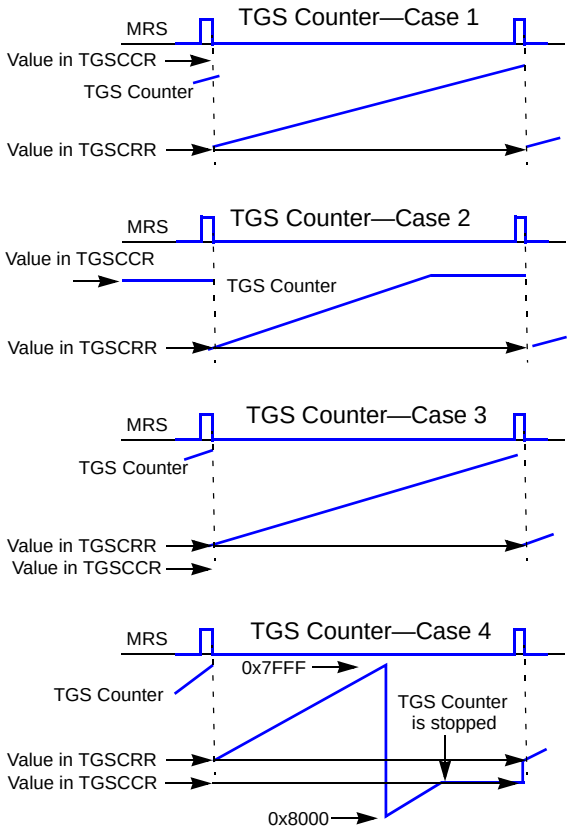


Figure 18-34. TGS counter cases

18.5.7 Debug mode

When the MCU is in debug mode, the CTU behavior is unaffected and remains dictated by the mode of the CTU.

18.6 Scheduler sub unit (SU)

The structure of the SU is shown in Figure 18-35.

The SU generates the trigger event output according to the occurred trigger event, and it has the same functionality in both TGS modes (triggered mode and sequential mode). Each of the following SU outputs:

- ADC command
- ETIMER1 pulse
- ETIMER2 pulse
- ETIMER3 pulse
- ETIMER4 pulse
- External trigger pulse

can be linked to any of eight trigger events by the Trigger Handler block. Each trigger event can be linked to one or more SU outputs.

If two events at the same time are linked to the same output only one output is generated and an error is provided. The output is generated using the trigger with the lowest index. For example, if trigger 0 and trigger 1 are linked to the ADC output and they occur together, an error is generated and the output linked with the trigger 0 is generated.

When a trigger is linked to the ADC, an associated ADC command (or stream of commands) is generated. The ADC Commands List Control Register CLCR*x* sets the assignment to an ADC command or to a stream of commands. When a trigger is linked to a timer or to the external trigger, a pulse with an active rising edge is generated. Additional features for the external triggers are available:

The external trigger output has:

- Pulse mode
- Toggle mode

In Toggle mode, each trigger event is linked to the external trigger, the external trigger pin toggles. The ON-Time for both modes (Pulse Mode and Toggle Mode) of the triggers is defined from a COTR register (Control On Time Register). A guard time is also defined from the same register at the same value of the ON-Time. A new trigger is generated only when the ON time + Guard Time has expired. The ON-Time and the Guard Time are only used for external triggers.

External signals can be asynchronous with MOTC_CLK, the motor control clock. For this reason, a programmable digital filter is available. The external signal is considered at 1 if it is latched at 1 at a time of $n + 1$, and is considered at 0 if it is latched at 0 at a time of $n + 1$, where n is the value in the Cross Triggering Unit Digital Filter Register (CTUDF).

Trigger events in the SU can be initiated by hardware or by software, and an additional software control is possible for each trigger event (as for the MRS), so 1 bit for each trigger event in the Cross Triggering Unit Control Register (CTUCR) generates an equivalent software trigger event. Each of these bits can be cleared by a respective hardware or software trigger event.

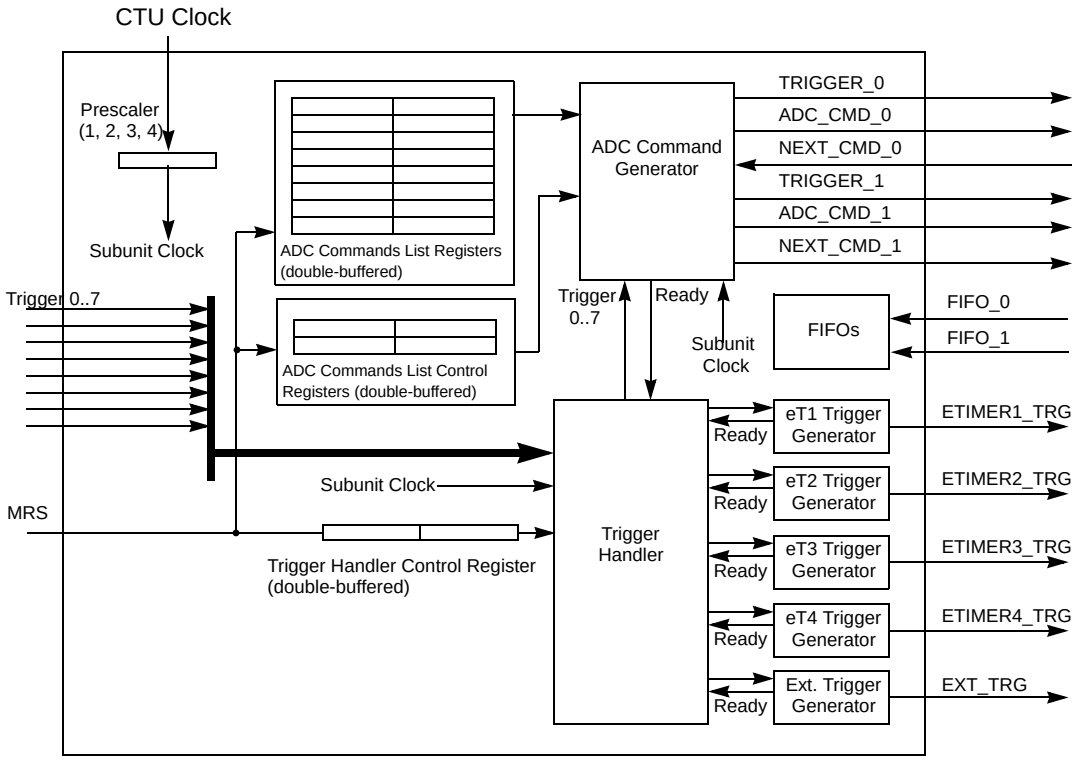


Figure 18-35. Scheduler sub unit

18.6.1 ADC commands list

The ADC can be controlled by the CPU (CPU Control Mode) or by the CTU (CTU Control Mode). The CTU can control the ADC by sending an ADC command only when the ADC is in CTU control mode. During the CTU control mode, the CPU is able to write to the ADC registers but it cannot start a new conversion. A control bit is allowed to select from the classic interface of the CTU control mode. Once selected, no change is possible unless a reset occurs.

The SU uses a Commands List in order to select the command to send to the ADC when a trigger event occurs. The commands list can hold 24 16-bit commands (see Section 18.6.2, ADC commands list format) and it is double-buffered, i.e., the commands list can be updated at any time between two consecutive MRS, but the changes become valid only after the next MRS occurs and a correct reload is performed. In order to manage the commands list, 5 bits are available in the CLCR*x* (ADC Commands List Control Register *x*), for the position of the first command in the list of commands for each trigger event. The number of commands piloted by the same trigger event is defined directly in the commands list. For each command, a bit defines whether or not it is the first command of a commands list.

18.6.2 ADC commands list format

The ADCs support the Single Conversion Mode (1 bit in the ADC command format allows selection of the conversion mode), and the Dual Conversion Mode (the sampling phases and the conversion phases are performed at the same time, and the storage of the results are performed in series). In both modes, the result of each conversion can be stored in one of the 4 available FIFOs. In dual conversion mode, both ADCs

must store the result of their conversion in the same FIFO. If the access to the FIFO is in the same clock cycle, ADC unit A has the priority. Otherwise, the first ADC that ends its conversion will write as first in the FIFO. Four analog channels are shared across the two ADCs and the total number of channels is 28 (12 + 12 + 4 shared channels), i.e., 16 channels for each ADC (12 + 4 shared channels). The dual conversion mode on the same physical channel is not allowed, but the dual conversion mode on the same channel number is allowed. According to this, if in dual conversion mode, the channel number is the same for both the ADCs and the selected channel is one of the shared channels, the CTU will detect an invalid command. In dual conversion mode, 4 bits for each ADC are used to select the channel number, and the conversion mode selection bit is used to select the dual conversion mode. If the single conversion mode is selected, 5 of the 8 bits reserved to select the channels in dual conversion mode are re-used to select the channel (4 bits) and the ADC unit (1 bit). See Section 18.4.1.10, Commands List Register x (CLRx).

The interrupt request bit is used as an interrupt request to the CPU when the ADC will complete the command with this bit set and it is only for CTU internal use.Before the next command to the CTU controls is sent, the value of the first command bit is checked to see if the current command is the first command of a new stream of consecutive commands or not. If not, the CTU sends the command. According to the previous considerations, the commands in the list allow control on:

- Channel A: number of ADC channel to sample from ADC unit A (4 bits)
- Channel B: number of ADC channel to sample from ADC unit B (4 bits)
- FIFO selection bits for the ADC unit A/B (2 bits)
- Conversion Mode selection bit:
  - 0 = Single Conversion Mode
  - 1 = Dual Conversion Mode
- First command bit (only for CTU internal use)
- Interrupt request bit (only for CTU internal use)

NOTE

The CTU FIFO order is non-deterministic. FIFO order may be reversed under certain conditions.

18.6.3 ADC results

ADC results can be stored in the channel relevant standard result register and/or in one of the 4 FIFOs: the different FIFOs allow to dispatch ADC results according to their type of acquisition (for example, phase currents, rotor position, ground-noise, other). Each FIFO has its own interrupt line and DMA request signal, plus an individual overflow error bit in the FIFO status register. The store location is specified in the ADC command, i.e., the FIFOs are available only in CTU Control Mode. Each entry of a FIFO is 32 bits. The size of the FIFOs are the following: FIFO0 and FIFO1: 16 entries (sized to avoid overflow during a full PWM period for current acquisitions); FIFO2 and FIFO3: 4 entries (low acquisition rate FIFOs). Results in each FIFO can be read by 16-bit read transaction (only the result is read in order to minimize the CPU load before computing on results) or by 32-bit read transaction (both the result and the channel number are read in order to avoid blind acquisitions). 5 bits in the upper 16 bits indicate the ADC unit (1 bit) and the channel number (4 bits). The result registers (only for the FIFOs) can be read from two

different addresses in the ADC memory map. The format of the result depends on the address from which it is read. The available formats are:

- Unsigned right-justified—conversion result is unsigned right-justified data. Bits [20:31] are used for 12-bit resolution, and bits [16:19] always return 0 when read. See Section 18.4.1.15, FIFO Right-Aligned Data x Register (FRx).
- Signed left-justified—conversion result is signed left-justified data. Bit [16] is reserved for sign and is always read as 0 for this ADC, bits [17:28] are used for 12-bit resolution, and bits [29:31] always return 0 when read. See Section 18.4.1.16, FIFO Signed Left-Aligned Data x Register (FLx).

18.7 Reload mechanism

Some CTU registers are double-buffered, and the reload is controlled by a reload enable bit, as the TGSISR_RE bit or the DFE bit, but for the most of the double-buffered registers, the reload is controlled by the MRS occurrence, and it is synchronized with the beginning of the CTU control period.

If the MRS occurs while the user is updating some double-buffered registers, for example, some registers of the Triggers List, the new Triggers List will be a mix of the old Triggers List and the new Triggers List, because the user has not ended the update of the Triggers List before the MRS occurrence.

In order to avoid this case, one bit is used to enable the reload operation, i.e., to inform the CTU that the user has ended updates to the double-buffered registers, and the reload can be performed without problems of mixed scenarios. In order to improve the coherency, the reload of all double-buffered registers is enabled by setting the General Reload Enable (GRE) bit in the CTU Control Register (CTUCR). The user must ensure that all intended double-buffered registers are updated before a new MRS occurrence. If an MRS occurs before the CTUCR[GRE] bit is set (for example, wrong application timing), the update is not performed, the previous values of all double-buffered registers remain active, the error flag is set (the MRS_RE bit in the CTU Error Flag Register) and, if enabled, CTU performs an interrupt request.

All the double-buffered registers use the CTUCR[GRE] bit to enable the reload when the MRS occurs. When the CTUCR[GRE] bit is set, the reload can be performed. When this bit is cleared, the reload is not performed. A correct reload resets the CTUCR[GRE] bit. All double-buffered registers cannot be written to while the CTUCR[GRE] bit remains set to 1. The CTUCR[GRE] bit can be reset by the occurrence of the next MRS (i.e., a correct reload) or by software setting the CTUCR[CGRE] bit.

The CTUCR[CGRE] bit is reset by hardware after the CTUCR[GRE] bit is reset. If the user sets the CTUCR[CGRE] bit and at the same time a MRS occurs, CGRE has the priority. Thus, CTUCR[GRE] is reset and the reload is not performed. In the same way, the CTUCR[GRE] bit has the priority when compared with the MRS occurrence, and the CTUCR[CGRE] has the priority compared with the CTUCR[GRE] (the two bits are in the same register so they can be set in the same time). MRS has the priority compared with the re-synchronization bit of the TGSISR.

In order to verify if a reload error occurs, the Flag GRE (FGRE) bit in the CTU Control Register (CTUCR) is used. When one of the double-buffered registers is written, CTUCR[FGRE] is set to 1 and it is reset by a correct reload. When the MRS occurs while CTUCR[FGRE] is 1 and CTUCR[GRE] is 1, a correct reload is performed (because all intended registers have been updated before the MRS occurs). If CTUCR[FGRE] is 1 and CTUCR[GRE] is 0, a reload is not performed, the error flag (MRS_RE) is set,

and (if enabled) an interrupt for an error is performed (in this case at least one register was written but the update has not ended before the MRS occurrence). If CTUCR[FGRE] is 0, a reload is not necessary because all the double-buffered registers are unchanged (see Figure 18-36).

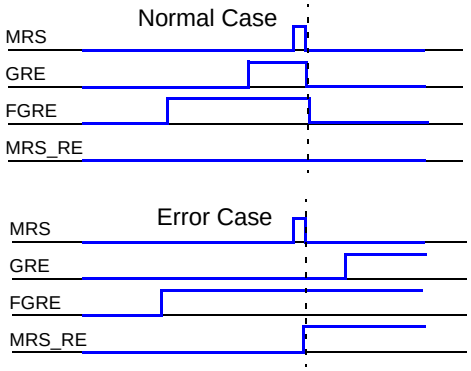


Figure 18-36. Reload error scenario

18.8 STOP mode

To reduce power consumption, it is also possible to enable a stop request from the MC_ME. The FIFOs are considered a lot like memory mapped registers. When the clock is started after a stop signal, some mistakes could occur. For example, a wrong trigger could be provided because it was programmed before the stop signal was performed, and some incorrect write operations into the FIFOs could happen. For this reason after a stop signal the FIFO must be empty. In order to avoid the problems linked to a wrong trigger, the CTU output can be disabled by setting the CTUCR[CTU_ODIS] bit and the ADC interface state machine can be reset by the CTUCR[CTU_ADC_R] (see Section 18.4.1.21, Cross Triggering Unit Control Register (CTUCR).

18.9 Interrupts and DMA requests

18.9.1 DMA support

The DMA can be used to configure the CTU registers. One DMA channel is reserved for performing a block transfer, and the MRS can be used as an optional DMA request signal (MRS_DMAE bit in the CTU Interrupt/DMA Register).

NOTE

If enabled, the DMA request on the MRS occurrence is performed only if a reload is performed, i.e., only if the CTUCR[GRE] bit is set.

Moreover, this CTU implementation requires DMA support for reading the data from the FIFOs. One DMA channel is available for each FIFO. Each FIFO can perform a DMA request when the number of words stored in the FIFO reaches the threshold value.

18.9.2 CTU faults and errors

Faults and errors that could occur during the programming include:

- An MRS occurs while user is updating the double-buffered registers and the CTUEFR[MRS_RE] bit is set.
- Receiving more than 8 Event Signals (EVs) before the next MRS occurs in TGS sequential mode and the CTUEFR[SM_TO] bit is set.
- A trigger event occurs during the time when the actions of the previous trigger event are not completed (the user must ensure that no trigger event will occur while the current trigger event is being processed—otherwise, the incoming trigger will be lost and an error will occur).

  There are four overrun flags (one for each type of output). The general mechanism is as in Figure 18-35.

  The Trigger Handler, when a trigger event occurs, and the corresponding Ready signal is high, presents the respective trigger signal (one cycle high time + one cycle low time) to the respective generator sub-block (ADC Command Generator, eT0 Trigger Generator, eT1 Trigger Generator or Ext. Trigger Generator). This generator sub-block then generates the requested signal. Until this real signal is generated (including guard time) the Ready signal is kept low.

  In the case of ADC command generator, the Ready signal is kept low until the last conversion in the batch is finished. The respective overrun flag is set at the following conditions:

  — Ready signal is low.

  — The rising edge of the respective trigger signal (from Trigger Handler to generator sub-block) occurs.

  This architecture allows user to pre-set, for example, a trigger to the eTimer1 in the middle of an ADC conversion, i.e., the SU will be considered busy only if a request to perform the same action that the SU is already performing occurs. One of the ADC_OE, T0_OE, T1_OE or ET_OE bit is set.
- Invalid (unrecognized) ADC command and the CTUEFR[ICE] bit is set.
- The MRS occurs before the enabled trigger events occur and the MRS_O bit is set.
- TGS overrun in sequential mode: a new incoming EV occurs before the trigger event selected by the previous EV occurs. The incoming EV sets an internal busy flag. The outgoing trigger event (all lines are OR-ed) resets this flag to 0. TGS Overrun in the sequential mode is generated under the following conditions:

  — TGS is in sequential mode.

  — There is an incoming EV while the busy flag is high. The TGS_OSM bit is set.

The faults/errors flags in the CTUEFR and in the CTUIFR can be cleared by writing a 1 while writing a 0 has no effect. The CTU does not support a write-protection mechanism.

18.9.3 CTU interrupt/DMA requests

The CTU can perform the following interrupt/DMA requests (15 interrupt lines):

- Error interrupt request (ERR_I; see Section 18.9.2, CTU faults and errors; 1 interrupt line)
- ADC command interrupt request (ADC_I; 1 interrupt line)
- Interrupt request on MRS occurrence (MRS_I; 1 interrupt line)

- Interrupt request on each trigger event occurrence (T0_I–T7_I; 1 interrupt line for each trigger event)
- FIFOs interrupt requests and/or DMA transfer request (FIFO0_I–FIFO3_I; 1 interrupt line for each FIFO)
- DMA transfer request on the MRS occurrence if CTUCR[GRE] bit is set (see Section 18.9.1, DMA support)

The interrupt flags are shown in Table 18-32.

Table 18-32. CTU interrupts

Category	Interrupt Name	IRQ# / Offset	Interrupt function / Bit name
Managed individually	MRS_I	IRQ193 / 0x0C10	MRS Interrupt flag — CTUIFR[MRS_I]
	T0_I	IRQ194 / 0x0C20	Trigger 0 interrupt flag — CTUIFR[T0_I]
	T1_I	IRQ195 / 0x0C30	Trigger 1 interrupt flag — CTUIFR[T1_I]
	T2_I	IRQ196 / 0x0C40	Trigger 2 interrupt flag — CTUIFR[T2_I]
	T3_I	IRQ197 / 0x0C50	Trigger 3 interrupt flag — CTUIFR[T3_I]
	T4_I	IRQ198 / 0x0C60	Trigger 4 interrupt flag — CTUIFR[T4_I]
	T5_I	IRQ199 / 0x0C70	Trigger 5 interrupt flag — CTUIFR[T5_I]
	T6_I	IRQ200 / 0x0C80	Trigger 6 interrupt flag — CTUIFR[T6_I]
	T7_I	IRQ201 / 0x0C90	Trigger 7 interrupt flag — CTUIFR[T7_I]
	ADC_I	IRQ206 / 0x0CE0	ADC command interrupt flag — CTUIFR[ADC_I]
ORed onto FIFO0_I (IRQ202)	FIFO_FULL0	IRQ202 / 0x0CA0	FIFO 0 full interrupt flag — FST[FULL0]
	FIFO_EMPTY0		FIFO 0 empty interrupt flag — FST[EMP0]
	FIFO_OVERFLOW0		FIFO 0 overflow interrupt flag — FST[OF0]
	FIFO_OVERRUN0		FIFO 0 overrun interrupt flag — FST[OR0]
ORed onto FIFO1_I (IRQ203)	FIFO_FULL1	IRQ203 / 0x0CB0	FIFO 1 full interrupt flag — FST[FULL1]
	FIFO_EMPTY1		FIFO 1 empty interrupt flag — FST[EMP1]
	FIFO_OVERFLOW1		FIFO 1 overflow interrupt flag — FST[OF1]
	FIFO_OVERRUN1		FIFO 1 overrun interrupt flag — FST[OR1]
ORed onto FIFO2_I (IRQ204)	FIFO_FULL2	IRQ204 / 0x0CC0	FIFO 2 full interrupt flag — FST[FULL2]
	FIFO_EMPTY2		FIFO 2 empty interrupt flag — FST[EMP2]
	FIFO_OVERFLOW2		FIFO 2 overflow interrupt flag — FST[OF2]
	FIFO_OVERRUN2		FIFO 2 overrun interrupt flag — FST[OR2]

Table 18-32. CTU interrupts (continued)

Category	Interrupt Name	IRQ# / Offset	Interrupt function / Bit name
ORed onto FIFO3_I (IRQ205)	FIFO_FULL3	IRQ205 / 0x0CD0	FIFO 3 full interrupt flag — FST[FULL3]
	FIFO_EMPTY3		FIFO 3 empty interrupt flag — FST[EMP3]
	FIFO_OVERFLOW 3		FIFO 3 overflow interrupt flag — FST[OF3]
	FIFO_OVERRUN3		FIFO 3 overrun interrupt flag — FST[OR3]
ORed onto ERR_I (IRQ207)	MRS_RE	IRQ207 / 0x0CF0	Master Reload Signal Reload Error — CTUEFR[MRS_RE]
	SM_TO		Trigger Overrun (more than 8 EV) in TGS Sequential Mode — CTUEFR[SM_TO]
	ICE		Invalid Command Error interrupt flag — CTUEFR[ICE]
	MRS_O		Master Reload Signal Overrun interrupt flag — CTUEFR[MRS_O]
	TGS_OSM		TGS Overrun in Sequential Mode interrupt flag — CTUEFR[TGS_OSM]
	ADC_OE		ADC command generation Overrun Error interrupt flag — CTUEFR[ADC_OE]
	T0_OE		Timer 0 trigger generation Overrun Error interrupt flag — CTUEFR[T0_OE]
	T1_OE		Timer 1 trigger generation Overrun Error interrupt flag — CTUEFR[T1_OE]
	T2_OE		Timer 2 trigger generation Overrun Error interrupt flag — CTUEFR[T2_OE]
	ET_OE		External Trigger generation Overrun Error interrupt flag — CTUEFR[ET_OE]

This page is intentionally left blank.

Chapter 19
Enhanced Direct Memory Access (eDMA)

19.1 Introduction

The eDMA module is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor via 32 programmable channels. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations and the actual data movement operations, along with a local memory containing the *transfer control descriptors* (TCD) for the channels. This SRAM-based implementation minimizes the overall module size.

Figure 19-1 is a block diagram of the eDMA module.

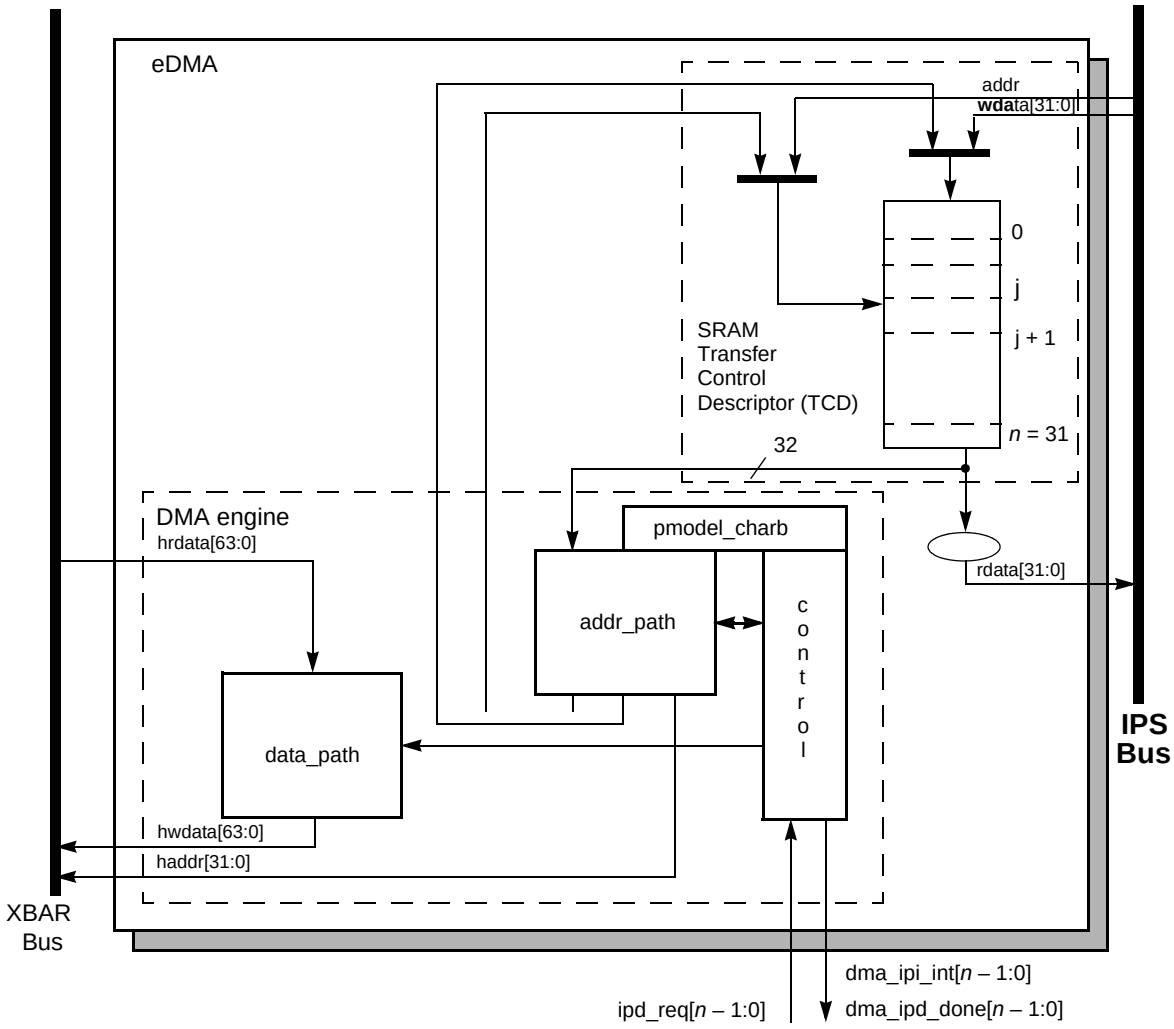


Figure 19-1. eDMA block diagram

19.1.1 Overview

The eDMA is a highly programmable data transfer engine that has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The eDMA hardware supports:

- 32-channel implementation
- Connections to the XBAR for bus mastering the data movement, slave bus for programming the module
  - Support for 64-bit XBAR datapath widths
- 32-byte transfer control descriptor per channel stored in local memory
- 32 bytes of data registers, used as temporary storage to support burst transfers

Throughout this chapter, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

19.1.2 Features

The eDMA module supports the following features:

- 32 channels support independent 8-, 16-, or 32-bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to post-increment or remain constant
- Each transfer is initiated by a peripheral, CPU, or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the INTC on completion of a single value or block transfer
- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a “minor” byte transfer count
  - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
    - Independent channel linking at end of minor loop and/or major loop
  - Peripheral hardware requests (one per channel, assigned in the DMA Channel Mux)
  - For all three methods, *one service request per execution of the minor loop is required*
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count

  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather DMA processing
- Support for complex data structures
- Support to cancel transfers via software or hardware

The structure of the transfer control descriptor is fundamental to the operation of the eDMA module. It is defined below in a ‘C’ pseudo-code specification, where `int` refers to a 32-bit variable (unless noted otherwise) and `short` is a 16-bit variable.

NOTE

To compile these structures, change any periods '.' in the variable name to underscores '_'.

```
typedef union {
    struct {              /* citer.e_link = 1 */
        unsigned short citer.linkch:6; /* link channel number, */
        unsigned short citer:9;     /* current (“major”) iteration count */
    } minor_link_enabled;           /* channel link at end of the minor loop */
    struct {                        /* citer.e_link = 0 */
        unsigned short citer:15;    /* current (“major”) iteration count */
    } minor_link_disabled;          /* no linking at end of the minor loop */
} t_minor_link_citer;

typedef union {
    struct {              /* biter.e_link = 1 */
        unsigned short biter.linkch:6; /* link channel number, */
        unsigned short biter:9;     /* beginning (“major”) iteration count */
    } init_minor_link_enabled;      /* channel link at end of the minor loop */
    struct {                        /* biter.e_link = 0 */
        unsigned short biter:15;    /* beginning (“major”) iteration count */
    } init_minor_link_disabled;     /* no linking at end of the minor loop */
} t_minor_link_biter;

typedef struct {
         unsigned int      saddr;   /* source address */
         unsigned int      smod:5;  /* source address modulo */
         unsigned int      ssize:3; /* source transfer size */
         unsigned int      dmod:5;  /* destination address modulo */
         unsigned int      dsize:3; /* destination transfer size */
         short             soff;    /* signed source address offset */
         unsigned int      nbytes;  /* inner (“minor”) byte count */
         int               slast;   /* last source address adjustment */
         unsigned int      daddr;   /* destination address */
         unsigned short    citer.e_link:1;/* enable channel linking on minor loop */
         t_minor_link_citerminor_link_citer;/* conditional current iteration count */
         short             doff;    /* signed destination address offset */
         int               dlast_sga;/* last destination address adjustment, or
                                        scatter/gather address (if e_sg = 1) */
         unsigned short    biter.e_link:1;/* beginning channel link enable */
         t_minor_link_biterminor_link_biter;/* beginning (“major”) iteration count */
         unsigned int      bwc:2;   /* bandwidth control */
         unsigned int      major.linkch:6;/* link channel number */
         unsigned int      done:1;  /* channel done */
```

```
        unsigned int     active:1;/* channel executing */
        unsigned int     major.e_link:1;/* enable channel linking on major loop*/
        unsigned int     e_sg:1;  /* enable scatter/gather descriptor */
        unsigned int     d_req:1; /* disable ipd_req when done */
        unsigned int     int_half:1;/* interrupt on citer = (biter >> 1) */
        unsigned int     int_maj:1;/* interrupt on major loop completion */
        unsigned int     start:1; /* explicit channel start */
} tcd                            /* transfer_control_descriptor */
```

The basic operation of a channel is defined as:

1. The channel is initialized by software loading the transfer control descriptor into the eDMA's programming model, memory-mapped through the IPS space, and implemented as local memory.
2. The channel requests service; either explicitly by software, a peripheral request or a linkage from another channel.

NOTE

The major loop executes one iteration per service request.

3. The contents of the transfer control descriptor for the activated channel is read from the local memory and loaded into the eDMA engine's internal register file.
4. The eDMA engine executes the data transfer defined by the transfer control descriptor, reading from the source and writing to the destination. The number of iterations in the minor loop is automatically calculated by the eDMA engine. The number of iterations within the minor loop is a function of the number of bytes to transfer (nbytes), the source size (ssize) and the destination size (dsize). The completion of the minor loop is equal to one iteration of the major loop.
5. At the conclusion of the minor loop's execution, certain fields of the transfer control descriptor are written back to the local TCD memory.

The process (steps 2–5) is repeated until the outer major loop's iteration count is exhausted. At that time, additional processing steps are completed, for example, the optional assertion of an interrupt request signaling the transfer's completion, final adjustments to the source and destination addresses, etc. A more detailed description of the channel processing is listed in the pseudo-code below. This simplified example is intended to represent basic data transfers. Detailed processing associated with the error handling is omitted.

```
/* the given DMAchannel is requesting service by the software assertion of the
   tcd[channel].start bit, the assertion of an enabled ipd_req from a device, or
   the implicit assertion of a channel-to-channel link */

/* begin by reading the transfer control descriptor from the local RAM
   into the local dma_engine registers */
dma_engine        = read_from_local_memory [channel];
dma_engine.active = 1;             /* set active flag */
dma_engine.done   = 0;             /* clear done flag */

/* check the transfer control descriptor for consistency */
if (dma_engine.config_error == 0) {

    / * begin execution of the inner “minor” loop transfers */
    {

        /* convert the source transfer size into a byte count */
```

```
switch (dma_engine.ssize) {
case 0:                         /* 8-bit transfer */
    src_xfr_size = 1;
    break;
case 1:                         /* 16-bit transfer */
    src_xfr_size = 2;
    break;
case 2:                         /* 32-bit transfer */
    src_xfr_size = 4;
    break;
case 3:                         /* 64-bit transfer */
    src_xfr_size = 8;
    break;
case 4:                         /* 16-byte burst transfer */
    src_xfr_size = 16;
    break;
case 5:                         /* 32-byte burst transfer */
    src_xfr_size = 32;
    break;
}

/* convert the destination transfer size into a byte count */
switch (dma_engine.dsize) {
case 0:                         /* 8-bit transfer */
    dest_xfr_size = 1;
    break;
case 1:                         /* 16-bit transfer */
    dest_xfr_size = 2;
    break;
case 2:                         /* 32-bit transfer */
    dest_xfr_size = 4;
    break;
case 3:                         /* 64-bit transfer */
    dest_xfr_size = 8;
    break;
case 4:                         /* 16-byte burst transfer */
    dest_xfr_size = 16;
    break;
case 5:                         /* 32-byte burst transfer */
    dest_xfr_size = 32;
    break;
}

/* determine the larger of the two transfer sizes, this value reflects */
/* the number of bytes transferred per read->write sequence. */
/* number of iterations of the minor loop = nbytes / xfer_size */
if (dma_engine.ssize < dma_engine.dsize)
    xfr_size = dest_xfer_size;
 else
    xfr_size = src_xfer_size;

/* process the source address, READ data into the buffer*/

/* read “xfr_size” bytes from the source */
/* if the ssize < dsize, do multiple reads to equal the dsize */
/* if the ssize => dsize, do a single read of source data */
number_of_source_reads = xfer_size / src_xfer_size;
```

```
 for (number_of_source_reads) {
     dma_engine.data = read_from_amba-ahb (dma_engine.saddr, src_xfr_size);

     /* generate the next-state source address */
     /* sum the current saddr with the signed source offset */
     ns_addr = dma_engine.saddr + (int) dma_engine.soff; }

     /* if enabled, apply the power-of-2 modulo to the next-state addr */
     if (dma_engine.smod != 0)
         address_select = (1 << dma_engine.smod) - 1; }
     else
         address_select = 0xffff_ffff;

     dma_engine.saddr = ns_addr           &  address_select
                 | dma_engine.saddr & ~address_select; }
}


 /* process the destination address, WRITE data from buffer */

 /* write “xfr_size” bytes to the destination */
 /* if the dsize < ssize, do multiple writes to equal the ssize */
 /* if the dsize => ssize, do a single write of dest data */
 number_of_dest_writes = xfer_size / dest_xfer_size;

 for (number_of_dest_writes) {
     write_to_amba-ahb (dma_engine.daddr, dest_xfr_size) = dma_engine.data;

     /* generate the next-state destination address */
     /* sum the current daddr with the signed destination offset */
     ns_addr = dma_engine.daddr + (int) dma_engine.doff;

     /* if enabled, apply the power-of-2 modulo to the next-state dest addr */
     if (dma_engine.dmod != 0)
         address_select = (1 << dma_engine.dmod) - 1;
     else
         address_select = 0xffff_ffff;

     dma_engine.daddr = ns_addr           &  address_select
                 | dma_engine.daddr & ~address_select;
 }
 if (cancel_transfer)
     break;

 /* check for a higher priority channel to service if: */
 /* 1) preemption is enabled */
 /* 2) in fixed arbitration mode */
 /* 3) a higher priority channel is requesting service */
 /* 4) not already servicing a preempting channel */
 if ((DCHPRIn.ecp = 1)  &  fixed_arbitration_mode
     higher_pri_request & ~current_channel_is_preempt)
     service_preempt_channel;

 /* the bandwidth control field determines when the next read/write occurs */
 if (dma_engine.bwc > 1)
     stall_dma_engine (1 << dma_engine.bwc);
```

```
        /* decrement the minor loop byte count */
        dma_engine.nbytes = dma_engine.nbytes - xfr_size;

    }while (dma_engine.nbytes > 0) /* end of minor inner loop */

    dma_engine.citer--;            /* decrement major loop iteration count */

    /* if the major loop is not yet exhausted, update certain TCD values in the RAM */
    if (dma_engine.citer != 0) {
        write_to_local_memory [channel].saddr = dma_engine.saddr;
        write_to_local_memory [channel].daddr = dma_engine.daddr;
        write_to_local_memory [channel].citer = dma_engine.citer;

                        /* if minor loop linking is enabled, make the channel link */
        if (dma_engine.citer.e_link)
            TCD[citer.linkch].start = 1;     /* specified channel service req */

        /* check for interrupt assertion if half of the major iterations are done */
        if (dma_engine.int_half && (dma_engine.citer == (dma_engine.biter >> 1)))
            generate_interrupt (channel);

        dma_engine.active = 0;         /* clear the channel busy flag */

     }
     else { /* major loop is complete, dma_engine.citer == 0 */
       /* since the major loop is complete, perform the final address adjustments */

       /* sum the current {src,dst} addresses with “last” adjustment */
       write_to_local_memory [channel].saddr = dma_engine.saddr + dma_engine.slast;
       write_to_local_memory [channel].daddr = dma_engine.daddr + dma_engine.dlast;
       /* restore the major iteration count to the beginning value */
       write_to_local_memory [channel].citer = dma_engine.biter;

       /* check for interrupt assertion at completion of the major iteration */
       if (dma_engine.int_maj)
         generate_interrupt (channel);

        /* check if the ipd_req is to be disabled at completion of the major iteration */
       if (dma_engine.d_req)
           DMAERQ [channel] = 0;

       /* check for a scatter/gather transfer control descriptor */
       if (dma_engine.e_sg) {
         /* load new transfer control descriptor from the address defined by dlast_sga */
           write_to_local_memory [channel] =
                                    read_from_amba-ahb(dma_engine.dlast_sga,32);
       }
       if (dma_engine.major.e_link)
           TCD[major.linkch].start = 1;       /* specified channel service req */

       dma_engine.active = 0;               /* clear the channel busy flag */
       dma_engine.done = 1;                 /* set the channel done flag */

}
else { /* configuration error detected, abort the channel */
       dma_engine.error_status = error_type; /* record the error */
```

```
        dma_engine.active = 0;                /* clear the channel busy flag */
        /* check for interrupt assertion on error */
        if (dma_engine.int_err)
            generate_interrupt (channel);
}
```

For more details, see Section 19.2.1, Register descriptions, and Section 19.3, Functional description.

19.2 Memory map and register description

The eDMA's programming model is partitioned into two sections, both mapped into the slave bus space: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory.

Reading an *unimplemented* register bit or memory location returns the value of 0. Write the value of 0 to *unimplemented* register bits. Any access to a *reserved* memory location results in a bus error. *Reserved* memory locations are indicated in the memory map. For 16- and 32-channel implementations, reserved memory also includes the high order "H" registers containing channels 63–32 data (i.e., DMAERQH, DMAEEIH, DMAINTH, DMAERRH).

Many of the control registers have a bit width that matches the number of channels implemented in the module, i.e., 32 bits in size. Registers associated with a 32-channel design are implemented as two 32-bit registers, and include an "H" and "L" suffixes, signaling the "high" and "low" portions of the control function.

NOTE

The MPC5675K devices implement a 32-channel eDMA module. Thus, none of the high order "H" registers are implemented. Only the low order "L" registers are present on the MPC5675K. However, the register names retain the "L" suffixes and "Low" names to maintain compatibility with existing software.

The eDMA module does not include any logic to provide access control. Rather, this function is supported using the standard access control logic provided by the PBRIDGE controller. Table 19-1 lists the base addresses for the eDMA modules. Table 19-2 lists the registers in each DMA module. Table 19-3 is a 32-bit view of the eDMA's memory map.

Table 19-1. eDMA module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	DMA_0	0xFFF4_4000
	DMA_1	
Decoupled Parallel Mode (DPM)	DMA_0	0xFFF4_4000 (same as LSM)
	DMA_1	0x8FF4_4000

Table 19-2. DMA memory map

Offset from DMA_BASE)	Register	Access[1]	Reset value[2]	Location
0x0000	DMA Control Register (DMACR)	R/W	0x0000_0400	on page 498
0x0004	DMA Error Status (DMAES)	R	0x0000_0000	on page 500
0x0008	Reserved			
0x000C	DMA Enable Request Low (DMAERQL, Channels 31–00)	R/W	0x0000_0000	on page 503
0x0010	Reserved			
0x0014	DMA Enable Error Interrupt Low (DMAEEIL, Channels 31–00)	R/W	0x0000_0000	on page 503
0x0018	DMA Set Enable Request (DMASERQ)	W	0x00	on page 504
0x0019	DMA Clear Enable Request (DMACERQ)	W	0x00	on page 505
0x001A	DMA Set Enable Error Interrupt (DMASEEI)	W	0x00	on page 505
0x001B	DMA Clear Enable Error Interrupt (DMACEEI)	W	0x00	on page 506
0x001C	DMA Clear Interrupt Request (DMACINT)	W	0x00	on page 507
0x001D	DMA Clear Error (DMACERR)	W	0x00	on page 507
0x001E	DMA Set Start Bit (DMASSRT)	W	0x00	on page 508
0x001F	DMA Clear Done Status Bit (DMACDNE)	W	0x00	on page 509
0x0020	Reserved			
0x0024	DMA Interrupt Request Low (DMAINTL, Channels 31–00)	R/W	0x0000_0000	on page 509
0x0028	Reserved			
0x002C	DMA Error Low (DMAERRL, Channels 31–00)	R/W	0x0000_0000	on page 510
0x0030	Reserved			
0x0034	DMA Hardware Request Status Low (DMAHRSL, Channels 31–00)	R/W	0x0000_0000	on page 511
0x0038	DMA General Purpose Output Register (DMAGPOR)	R/W	0x0000_0000	on page 512
0x003C–0x00FF	Reserved			
0x0100	DMA Channel 0 Priority (DCHPRI0)	R/W	0x00	on page 512
0x0101	DMA Channel 1 Priority (DCHPRI1)	R/W	0x01	on page 512
0x0102	DMA Channel 2 Priority (DCHPRI2)	R/W	0x02	on page 512
0x0103	DMA Channel 3 Priority (DCHPRI3)	R/W	0x03	on page 512
0x0104	DMA Channel 4 Priority (DCHPRI4)	R/W	0x04	on page 512
0x0105	DMA Channel 5 Priority (DCHPRI5)	R/W	0x05	on page 512
0x0106	DMA Channel 6 Priority (DCHPRI6)	R/W	0x06	on page 512
0x0107	DMA Channel 7 Priority (DCHPRI7)	R/W	0x07	on page 512

Table 19-2. DMA memory map (continued)

Offset from DMA_BASE)	Register	Access[1]	Reset value[2]	Location
0x0108	DMA Channel 8 Priority (DCHPRI8)	R/W	0x08	on page 512
0x0109	DMA Channel 9 Priority (DCHPRI9)	R/W	0x09	on page 512
0x010A	DMA Channel 10 Priority (DCHPRI10)	R/W	0x0A	on page 512
0x010B	DMA Channel 11 Priority (DCHPRI11)	R/W	0x0B	on page 512
0x010C	DMA Channel 12 Priority (DCHPRI12)	R/W	0x0C	on page 512
0x010D	DMA Channel 13 Priority (DCHPRI13)	R/W	0x0D	on page 512
0x010E	DMA Channel 14 Priority (DCHPRI14)	R/W	0x0E	on page 512
0x010F	DMA Channel 15 Priority (DCHPRI15)	R/W	0x0F	on page 512
0x0110	DMA Channel 16 Priority (DCHPRI16)	R/W	0x10	on page 512
0x0111	DMA Channel 17 Priority (DCHPRI17)	R/W	0x11	on page 512
0x0112	DMA Channel 18 Priority (DCHPRI18)	R/W	0x12	on page 512
0x0113	DMA Channel 19 Priority (DCHPRI19)	R/W	0x13	on page 512
0x0114	DMA Channel 20 Priority (DCHPRI20)	R/W	0x14	on page 512
0x0115	DMA Channel 21 Priority (DCHPRI21)	R/W	0x15	on page 512
0x0116	DMA Channel 22 Priority (DCHPRI22)	R/W	0x16	on page 512
0x0117	DMA Channel 23 Priority (DCHPRI23)	R/W	0x17	on page 512
0x0118	DMA Channel 24 Priority (DCHPRI24)	R/W	0x18	on page 512
0x0119	DMA Channel 25 Priority (DCHPRI25)	R/W	0x19	on page 512
0x011A	DMA Channel 26 Priority (DCHPRI26)	R/W	0x1A	on page 512
0x011B	DMA Channel 27 Priority (DCHPRI27)	R/W	0x1B	on page 512
0x011C	DMA Channel 28 Priority (DCHPRI28)	R/W	0x1C	on page 512
0x011D	DMA Channel 29 Priority (DCHPRI29)	R/W	0x1D	on page 512
0x011E	DMA Channel 30 Priority (DCHPRI30)	R/W	0x1E	on page 512
0x011F	DMA Channel 31 Priority (DCHPRI31)	R/W	0x1F	on page 512
0x0120–0x0FFF	Reserved			
0x1000–0x11FC	TCD00–TCD15	R/W	—[3]	on page 513
0x1200–0x13FC	TCD16–TCD31	R/W	—[3]	on page 513
0x1400–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 See register for more information.

Table 19-3. eDMA 32-bit memory map

DMA Offset	Register			
0x0000	DMA Control Register (DMACR)			
0x0004	DMA Error Status (DMAES)			
0x0008	Reserved			
0x000C	DMA Enable Request Low (DMAERQL, Channels 31–00)			
0x0010	Reserved			
0x0014	DMA Enable Error Interrupt Low (DMAEEIL, Channels 31–00)			
0x0018	DMA Set Enable Request (DMASERQ)	DMA Clear Enable Request (DMACERQ)	DMA Set Enable Error Interrupt (DMASEEI)	DMA Clear Enable Error Interrupt (DMACEEI)
0x001C	DMA Clear Interrupt Request (DMACINT)	DMA Clear Error (DMACERR)	DMA Set Start Bit (DMASSRT)	DMA Clear Done Status Bit (DMACDNE)
0x0020	Reserved			
0x0024	DMA Interrupt Request Low (DMAINTL, Channels 31–00)			
0x0028	Reserved			
0x002C	DMA Error Low (DMAERRL, Channels 31–00)			
0x0030	Reserved			
0x0034	DMA Hardware Request Status Low (DMAHRSL, Channels 31–00)			
0x0038	DMA General Purpose Output Register (DMAGPOR)			
0x003C–0x00FF	Reserved			
0x0100	DMA Channel 0 Priority (DCHPRI0)	DMA Channel 1 Priority (DCHPRI1)	DMA Channel 2 Priority (DCHPRI2)	DMA Channel 3 Priority (DCHPRI3)
0x0104	DMA Channel 4 Priority (DCHPRI4)	DMA Channel 5 Priority (DCHPRI5)	DMA Channel 6 Priority (DCHPRI6)	DMA Channel 7 Priority (DCHPRI7)
0x0108	DMA Channel 8 Priority (DCHPRI8)	DMA Channel 9 Priority (DCHPRI9)	DMA Channel 10 Priority (DCHPRI10)	DMA Channel 11 Priority (DCHPRI11)
0x010C	DMA Channel 12 Priority (DCHPRI12)	DMA Channel 13 Priority (DCHPRI13)	DMA Channel 14 Priority (DCHPRI14)	DMA Channel 15 Priority (DCHPRI15)
0x0110	DMA Channel 16 Priority (DCHPRI16)	DMA Channel 17 Priority (DCHPRI17)	DMA Channel 18 Priority (DCHPRI18)	DMA Channel 19 Priority (DCHPRI19)
0x0114	DMA Channel 20 Priority (DCHPRI20)	DMA Channel 21 Priority (DCHPRI21)	DMA Channel 22 Priority (DCHPRI22)	DMA Channel 23 Priority (DCHPRI23)
0x0118	DMA Channel 24 Priority (DCHPRI24)	DMA Channel 25 Priority (DCHPRI25)	DMA Channel 26 Priority (DCHPRI26)	DMA Channel 27 Priority (DCHPRI27)
0x011C	DMA Channel 28 Priority (DCHPRI28)	DMA Channel 29 Priority (DCHPRI29)	DMA Channel 30 Priority (DCHPRI30)	DMA Channel 31 Priority (DCHPRI31)
0x0120–0x013F	Reserved			
0x0140–0x0FFF	Reserved			
0x1000–0x11FF	TCD00–TCD15			
0x1200–0x13FF	TCD16–TCD31			
0x1400–0x17FF	Reserved			

19.2.1 Register descriptions

19.2.1.1 DMA Control Register (DMACR)

The 32-bit DMACR defines the basic operating configuration of the eDMA module.

The eDMA module arbitrates channel service requests in groups of 16 channels. This 32-channel configuration has two groups (1,0). Group 1 contains channels 31–16, and group 0 contains channels 15–0.

Arbitration within a group can be configured to use either a fixed-priority or a round-robin selection. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see Section 19.2.1.17, DMA Channel n Priority (DCHPRIn) register). In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through without regard to priority.

The group priorities operate in a similar fashion. In group fixed-priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 3 is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRP*n*Pri registers. All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error is reported. Unused group priority registers, per configuration, are unimplemented in the DMACR. In group round-robin mode, the group priorities are ignored and the groups are cycled through without regard to priority.

Minor loop offsets are address offset values added to the final source address (saddr) or destination address (daddr) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (mloff) is added to the final source address (saddr), or the final destination address (daddr), or both prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (slast and dlast_sga) are used to compute the next saddr and daddr values.

When minor loop mapping is enabled (DMACR[EMLM] = 1), TCD*n* word2 is redefined. A portion of TCD*n* word2 specifies multiple fields: a source enable bit (smloe) to specify the minor loop offset should be applied to the source address (saddr) upon minor loop completion, a destination enable bit (dmloe) to specify the minor loop offset should be applied to the destination address (daddr) upon minor loop completion, and the sign-extended minor loop offset value (mloff). The same offset value (mloff) is used for both source and destination minor loop offsets. When either minor loop offset is enabled (smloe set or dmloe set), the nbytes field is reduced to 10 bits. When both minor loop offsets are disabled (smloe cleared and dmloe cleared), the nbytes field is a 30-bit vector.

When minor loop mapping is disabled (DMACR[EMLM] = 0), all 32 bits of TCD*n* word2 are assigned to the nbytes field. See Section 19.2.1.18, Transfer Control Descriptor (TCD), for more details.

See Figure 19-2 and Table 19-4 for the DMACR definition.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CX	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	GRP1PRI		GRP0PRI		EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 19-2. DMA Control Register (DMACR)

Table 19-4. DMACR field descriptions

Field	Description
CX	Cancel Transfer. 0 Normal operation. 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.
ECX	Error Cancel Transfer. 0 Normal operation. 1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the DMAES register and generating an optional error interrupt (see Section 19.2.1.2, DMA Error Status (DMAES)).
GRP1PRI	Channel Group 1 Priority. Group 1 priority level when fixed priority group arbitration is enabled.
GRP0PRI	Channel Group 0 Priority. Group 0 priority level when fixed priority group arbitration is enabled.
EMLM	Enable Minor Loop Mapping. 0 Minor loop mapping disabled. TCD*n*.word2 is defined as a 32-bit nbytes field. 1 Minor loop mapping enabled. When set, the TCD*n*.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.
CLM	Continuous Link Mode. 0 A minor loop channel link made to itself goes through channel arbitration before being activated again. 1 A minor loop channel link made to itself does not go through channel arbitration before being activated again. Upon minor loop completion, the channel is activated again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
HALT	Halt DMA Operations. 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.

Table 19-4. DMACR field descriptions (continued)

Field	Description
HOE	Halt On Error. 0 Normal operation. 1 Any error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.
ERGA	Enable Round-robin Group Arbitration. 0 Fixed priority arbitration is used for selection among the groups. 1 Round-robin arbitration is used for selection among the groups.
ERCA	Enable Round-robin Channel Arbitration. 0 Fixed priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
EDBG	Enable Debug. 0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the eDMA module to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the ipg_debug input is negated or the EDBG bit is cleared.

19.2.1.2 DMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast_sga) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.citer.e_link bit does not equal the TCD.biter.e_link bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction that is already pipelined after errant access, has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write is performed using the data captured during the bus error. If a bus error occurs

on the last write prior to switching to the next read sequence, the read sequence is performed before the channel is terminated due to the destination bus error.

A transfer may be cancelled by software via the DMACR[CX] bit or hardware via the **dma_cancel_xfer** input signal. When a cancel transfer request is recognized, the eDMA engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the eDMA engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting the DMACR[ECX] or asserting the **dma_err_cancel_xfer** input), the channel number is loaded into the ERRCHN field and the ECX and VLD bits are set are set in the DMAES register. In addition, an error interrupt may be generated if enabled. See Section 19.2.1.14, DMA Error Low (DMAERRL) register, for error interrupt details.

The occurrence of any type of error causes the eDMA engine to immediately stop, and the appropriate channel bit in the DMA Error register to be asserted. At the same time, the details of the error condition are loaded into the DMAES register. The major loop complete indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. See Figure 19-3 and Table 19-5 for the DMAES definition.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	0	ERRCHN					SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-3. DMA Error Status (DMAES) Register

Table 19-5. DMAES field descriptions

Field	Description
VLD	Logical OR of all DMAERRH and DMAERRL status bits. 0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
ECX	Transfer cancelled. 0 No cancelled transfers. 1 The last recorded entry was a cancelled transfer via the error cancel transfer input.

Table 19-5. DMAES field descriptions (continued)

Field	Description
GPE	Group Priority Error. 0 No group priority error. 1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique.
CPE	Channel Priority Error. 0 No channel priority error. 1 The last recorded error was a configuration error among the group priorities. All group priorities are not unique.
ERRCHN	Error Channel Number or Cancelled Channel Number. The channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled.
SAE	Source Address Error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.saddr field. TCD.saddr is inconsistent with TCD.ssize.
SOE	Source Offset Error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.soff field. TCD.soff is inconsistent with TCD.ssize.
DAE	Destination Address Error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.daddr field. TCD.daddr is inconsistent with TCD.dsize.
DOE	Destination Offset Error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.doff field. TCD.doff is inconsistent with TCD.dsize.
NCE	Nbytes/Citer Configuration Error. 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the TCD.nbytes or TCD.citer fields. TCD.nbytes is not a multiple of TCD.ssize and TCD.dsize, or TCD.citer is equal to zero, or TCD.citer.e_link is not equal to TCD.biter.e_link.
SGE	Scatter/Gather Configuration Error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.dlast_sga field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.e_sg is enabled. TCD.dlast_sga is not on a 32-byte boundary.
SBE	Source Bus Error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

19.2.1.3 DMA Enable Request Low (DMAERQL) Register

The DMA Enable Request Low (DMAERQL) register provides a bit map for the 32 implemented channels to enable the request signal for each channel. DMAEQRL covers channels 31-00. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the DMASERQ and DMACERQ registers. The DMA{S,C}ERQ registers are provided so that the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAERQL register.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the DMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request. See Figure 19-4 and Table 19-6 for the DMAERQ definition.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ31	ERQ30	ERQ29	ERQ28	ERQ27	ERQ26	ERQ25	ERQ24	ERQ23	ERQ22	ERQ21	ERQ20	ERQ19	ERQ18	ERQ17	ERQ16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-4. DMA Enable Request Low (DMAERQL) Register

Table 19-6. DMAERQL field descriptions

Field	Description
ERQ*n*	Enable DMA Request *n*. 0 The DMA request signal for channel *n* is disabled. 1 The DMA request signal for channel *n* is enabled.

As a given channel completes the processing of its major iteration count, a flag in the transfer control descriptor may affect the ending state of the DMAERQ bit for that channel. If the TCD.d_req bit is set, then the corresponding DMAERQ bit is cleared, disabling the DMA request; else if the d_req bit is cleared, the state of the DMAERQ bit is unaffected.

19.2.1.4 DMA Enable Error Interrupt Low (DMAEEIL) Register

The DMAEEIL register provides a bit map for the 32 implemented channels to enable the error interrupt signal for each channel. DMAEEIL covers channels 31-00. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to the DMASEEI and DMACEEI registers. The DMA{S,C}EEI registers are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAEEIL registers.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See Figure 19-5 and Table 19-7 for the DMAEEI definition.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-5. DMA Enable Error Interrupt Low (DMAEEIL) Register

Table 19-7. DMAEEIL field descriptions

Field	Description
EEI*n*	Enable Error Interrupt *n*. 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

19.2.1.5 DMA Set Enable Request (DMASERQ) register

The DMASERQ register provides a simple memory-mapped mechanism to set a given bit in the DMAERQL register to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQL register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAERQL to be asserted. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-6 and Table 19-8 for the DMASERQ definition.

Address: Base + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SERQ						
Reset	0	0	0	0	0	0	0	0

Figure 19-6. DMA Set Enable Request (DMASERQ) register

Table 19-8. DMASERQ field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
SERQ	Set Enable Request. 0–31 Set the corresponding bit in DMAERQL. 32–63 Reserved 64–127 Set all bits in DMAERQL.

19.2.1.6 DMA Clear Enable Request (DMACERQ)

The DMACERQ register provides a simple memory-mapped mechanism to clear a given bit in the DMAERQL register to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERQL to be zeroed, disabling all DMA request inputs. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-7 and Table 19-9 for the DMACERQ definition.

Address: Base + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERQ						
Reset	0	0	0	0	0	0	0	0

Figure 19-7. DMA Clear Enable Request (DMACERQ) Register

Table 19-9. DMACERQ field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
CERQ	Clear Enable Request. 0–31 Clear the corresponding bit in DMAERQL. 32–63 Reserved 64–127 Clear all bits in DMAERQL.

19.2.1.7 DMA Set Enable Error Interrupt (DMASEEI) register

The DMA Set Enable Error Interrupt (DMASEEI) register provides a simple memory-mapped mechanism to set a given bit in the DMAEEIL register to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEIL register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEIL to be asserted. If bit 0 (NOP) is set, the command is ignored. This allows multiple

byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-8 and Table 19-10 for the DMASEEI definition.

Address: Base + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SEEI						
Reset	0	0	0	0	0	0	0	0

Figure 19-8. DMA Set Enable Error Interrupt (DMASEEI) Register

Table 19-10. DMASEEI field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
SEEI	Set Enable Error Interrupt. 0–31 Set the corresponding bit in DMAEEIL. 32–63 Reserved 64–127 Set all bits in DMAEEIL.

19.2.1.8 DMA Clear Enable Error Interrupt (DMACEEI) register

The DMA Clear Enable Error Interrupt (DMACEEI) register provides a simple memory-mapped mechanism to clear a given bit in the DMAEEIL register to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEIL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAEEIL to be zeroed, disabling all DMA request inputs. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-9 and Table 19-11 for the DMACEEI definition.

Address: Base + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CEEI						
Reset	0	0	0	0	0	0	0	0

Figure 19-9. DMA Clear Enable Error Interrupt (DMACEEI) Register

Table 19-11. DMACEEI field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
CEEI	Clear Enable Error Interrupt. 0–31 Clear the corresponding bit in DMAEEIL. 32–63 Reserved 64–127 Clear all bits in DMAEEIL.

19.2.1.9 DMA Clear Interrupt Request (DMACINT)

The DMACINT register provides a simple memory-mapped mechanism to clear a given bit in the DMAINTL register to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMAINTL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAINTL to be zeroed, disabling all DMA interrupt requests. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-10 and Table 19-12 for the DMACINT definition.

Address: Base + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CINT						
Reset	0	0	0	0	0	0	0	0

Figure 19-10. DMA Clear Interrupt Request (DMACINT) Fields

Table 19-12. DMACINT field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
CINT	Clear Interrupt Request. 0–31 Clear the corresponding bit in DMAINTL. 32–63 Reserved 64–127 Clear all bits in DMAINTL.

19.2.1.10 DMA Clear Error (DMACERR) register

The DMA Clear Error (DMACERR) register provides a simple memory-mapped mechanism to clear a given bit in the DMAERRL register to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMAERRL register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERRL to be zeroed, clearing all channel error indicators. If bit 0 (NOP) is set,

the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Figure 19-11 and Table 19-13 for the DMACERR definition.

Address: Base + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERR						
Reset	0	0	0	0	0	0	0	0

Figure 19-11. DMA Clear Error (DMACERR) Register

Table 19-13. DMACERR field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
CERR	Clear Error Indicator. 0–31 Clear the corresponding bit in DMAERRL. 32–63 Reserved 64–127 Clear all bits in DMAERRL.

19.2.1.11 DMA Set START Bit (DMASSRT) register

The DMA Set START Bit (DMASSRT) register provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Table 19-30 for the TCD START bit definition.

Address: Base + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SSRT						
Reset	0	0	0	0	0	0	0	0

Figure 19-12. DMA Set START Bit (DMASSRT) Register

Table 19-14. DMASSRT field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
SSRT	Set START Bit (Channel Service Request). 0–31 Set the corresponding channel's TCD.start. 32–63 Reserved 64–127 Set all TCD.start bits.

19.2.1.12 DMA Clear DONE Status (DMACDNE) register

The DMA Clear DONE Status (DMACDNE) register provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding Transfer Control Descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. If bit 0 (NOP) is set, the command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes. See Table 19-30 for the TCD DONE bit definition.

Address: Base + 0x001F Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CDNE						
Reset	0	0	0	0	0	0	0	0

Figure 19-13. DMA Clear DONE Status (DMACDNE) Register

Table 19-15. DMACDNE field descriptions

Field	Description
NOP	No Operation. 0 Normal operation. 1 No operation, ignore bits 6-0.
CDNE	Clear DONE Status Bit. 0–31 Clear the corresponding channel's DONE bit. 32–63 Reserved 64–127 Clear all TCD DONE bits.

19.2.1.13 DMA Interrupt Request Low (DMAINTL) register

The DMA Interrupt Request Low (DMAINTL) register provides a bit map for the 32 implemented channels signaling the presence of an interrupt request for each channel. DMAINTL covers channels 31-00. The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the DMACINT register. On writes to the DMAINT, a one in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no affect on the corresponding channel's current interrupt status. The DMACINT register is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the DMAINTL registers. See Figure 19-14 and Table 19-16 for the DMAINT definition.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT15	INT14	INT13	INT12	INT11	INT10	INT09	INT08	INT07	INT06	INT05	INT04	INT03	INT02	INT01	INT00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-14. DMA Interrupt Request Low (DMAINTL) Register

Table 19-16. DMAEEIL field descriptions

Field	Description
INT*n*	DMA Interrupt Request *n*. 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

19.2.1.14 DMA Error Low (DMAERRL) register

The DMA Error Low (DMAERRL) register provides a bit map for the 32 implemented channels signaling the presence of an error for each channel. DMAERRL covers channels 31-00. The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, then logically summed across groups of 16 or 32 channels to form several group error interrupt requests, which is then routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the DMACERR register in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the DMAEEI register. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the DMACERR register. On writes to the DMAERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The DMACERR register is provided so the error indicator for a *single* channel can easily be cleared. See Figure 19-15 and Table 19-17 for the DMAERR definition.

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR31	ERR30	ERR29	ERR28	ERR27	ERR26	ERR25	ERR24	ERR23	ERR22	ERR21	ERR20	ERR19	ERR18	ERR17	ERR16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-15. DMA Error Low (DMAERRL) Register

Table 19-17. DMAERRL field descriptions

Field	Description
ERR*n*	DMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

19.2.1.15 DMA Hardware Request Status Low (DMAHRSL) register

The DMA Hardware Request Status Low (DMAHRSL) register provides a bit map for the 32 implemented channels to show the current hardware request status for each channel. DMAHRSL covers channels 31-00. Hardware request status reflects the current state of the registered and qualified (via the DMAERQ field) ipd_req lines as seen by the DMA2's arbitration logic. This view into the hardware request signals may be used for debug purposes.

See Figure 19-16 and Table 19-18 for the DMAHRS definition.

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS31	HRS30	HRS29	HRS28	HRS27	HRS26	HRS25	HRS24	HRS23	HRS22	HRS21	HRS20	HRS19	HRS18	HRS17	HRS16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS15	HRS14	HRS13	HRS12	HRS11	HRS10	HRS09	HRS08	HRS07	HRS06	HRS05	HRS04	HRS03	HRS02	HRS01	HRS00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-16. DMA Hardware Request Status Low (DMAHRSL) register

Table 19-18. DMAHRSL field descriptions

Field	Description
HRS*n*	DMA Hardware Request Status *n*. 0 A hardware service request for channel *n* is not present. 1 A hardware service request for channel *n* is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the DMAERQ*n* bit.

19.2.1.16 DMA General Purpose Output Register (DMAGPOR)

The DMA General Purpose Output Register (DMAGPOR) indicates to the Cache Coherency module that writes from this DMA are marked as global. Each bit of the DMAGPOR register corresponds to a particular channel (32 channels in all).

See Figure 19-17 and Table 19-19 for the DMAGPOR definition.

Address: Base + 0x0038 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	GBLW31	GBLW30	GBLW29	GBLW28	GBLW27	GBLW26	GBLW25	GBLW24	GBLW23	GBLW22	GBLW21	GBLW20	GBLW19	GBLW18	GBLW17	GBLW16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	GBLW15	GBLW14	GBLW13	GBLW12	GBLW11	GBLW10	GBLW09	GBLW08	GBLW07	GBLW06	GBLW05	GBLW04	GBLW03	GBLW02	GBLW01	GBLW00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-17. DMA General Purpose Output Register (DMAGPOR)

Table 19-19. DMAGPOR field descriptions

Field	Description
GBLW*n*	Global Write Indicator *n*. 0 Writes from channel *n* are considered non-global. 1 Writes from channel *n* are considered global.

19.2.1.17 DMA Channel *n* Priority (DCHPRI*n*) register

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value, i.e., 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0–15. When read, the GRPPRI bits of the DCHPRI*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the DCHPRI*n* registers. The group priority is assigned in the DMACR. See Figure 19-2 and Table 19-4 for the DMACR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRI*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes.

A channel's ability to preempt another channel can be disabled by setting the DPA bit in the DCHPRI*n* register. When a channel's preempt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel. See Figure 19-18 and Table 19-20 for the DCHPRI*n* definition.

Address: Base + 0x0100 to 0x011F.
See Table 19-2.

Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	DPA	GRPPRI		CHPRI			
W								
Reset	0	0	—[1]	—[1]	—[1]	—[1]	—[1]	—[1]

Figure 19-18. DMA Channel *n* Priority (DCHPRI*n*) register

[1] Defaults to channel *n* after reset.

Table 19-20. DCHPRI*n* field descriptions

Field	Description
ECP	Enable Channel Preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
DPA	Disable Preempt Ability. 0 Channel *n* can suspend a lower priority channel. 1 Channel *n* cannot suspend any channel, regardless of channel priority.
GRPPRI	Channel *n* Current Group Priority. Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read-only; writes are ignored.
CHPRI	Channel *n* Arbitration Priority. Channel priority when fixed-priority arbitration is enabled.

19.2.1.18 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The TCD structure was previously discussed in detail in Section 19.1.2, Features. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1, ... channel 31. The

definitions of the TCD are presented as eight 32-bit values. Table 19-21 is a 32-bit view of the basic TCD structure.

Table 19-21. TCD*n* 32-bit memory structure

DMA Offset	TCD*n* Field		
0x1000 + (32 × *n*) + 0x00	Source Address (saddr)		
0x1000 + (32 × *n*) + 0x04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)	
0x1000 + (32 × *n*) + 0x08	Signed Minor Loop Offset (smloe, dmloe, mloff)		Inner “Minor” Byte Count (nbytes)
0x1000 + (32 × *n*) + 0x0C	Last Source Address Adjustment (slast)		
0x1000 + (32 × *n*) + 0x10	Destination Address (daddr)		
0x1000 + (32 × *n*) + 0x14	Current “Major” Iteration Count (citer)	Signed Destination Address Offset (doff)	
0x1000 + (32 × *n*) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)		
0x1000 + (32 × *n*) + 0x1C	Beginning “Major” Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)	

Figure 19-19 and Table 19-22 define word 0 of the TCD*n* structure, the saddr field.

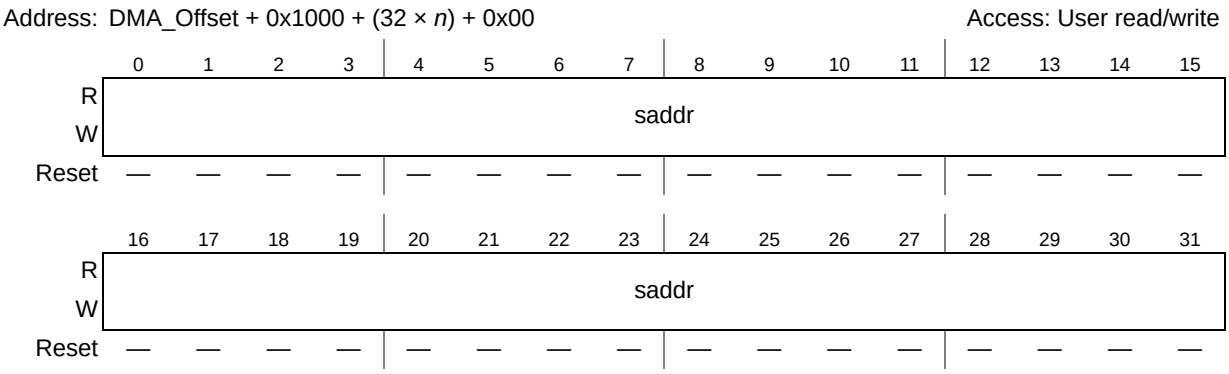


Figure 19-19. TCD*n* Word 0 (TCD*n*.saddr) field

Table 19-22. TCD*n* Word 0 (TCD*n*.saddr) field description

Field	Description
saddr	Source address. Memory address pointing to the source data.

Figure 19-20 and Table 19-23 define word 1 of the TCD*n* structure, the soff and transfer attribute fields.

Address: DMA_Offset + 0x1000 + (32 × *n*) + 0x04 Access: User read/write

Figure 19-20. TCD*n* Word 1 (TCD*n*.{soff,smod,ssize,dmod,dsize}) fields

Table 19-23. TCD*n* Word 1 (TCD*n*.{soff,smod,ssize,dmod,dsize}) field description

Field	Description
smod	Source address modulo. 0 Source address modulo feature is disabled. non-0 The value defines a specific address bit that is selected to be either the value after saddr + soff calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, the queue should be based at a 0-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as ((1 << smod[4:0]) - 1) where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, the soff is typically set to the transfer size to implement post-increment addressing with the smod function constraining the addresses to a 0-modulo-size range.
ssize	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 16-byte (32-bit AHB bus, WRAP4 burst) 100 Reserved (64-bit AHB bus) 101 32-byte 110 Reserved 111 Reserved Likewise, the attempted specification of a 16-byte source size in a 64-bit AMBA AHB bus implementation generates a configuration error.
dmod	Destination address modulo. See the smod definition.
dsize	Destination data transfer size. See the ssize definition.
soff	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

Figure 19-21 and Table 19-24 define word 2 of the TCD*n* structure, the nbytes field.

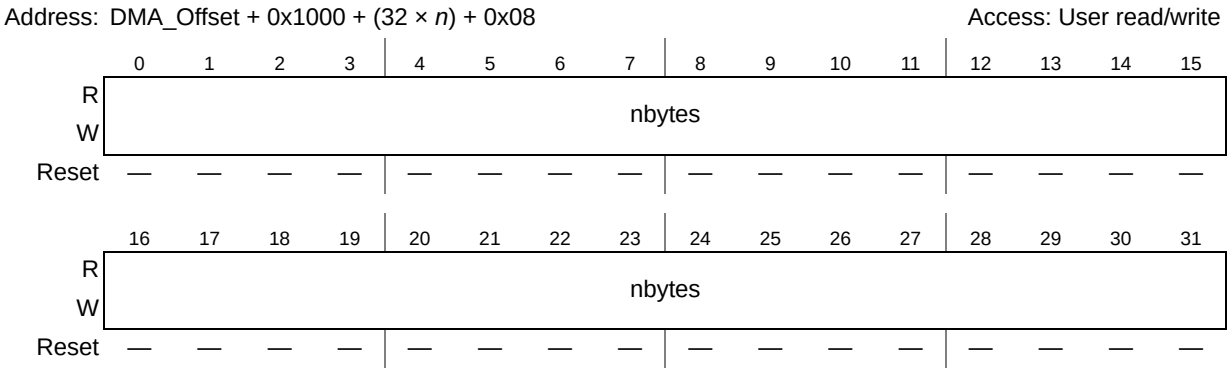


Figure 19-21. TCD*n* Word 2 (TCD*n*.nbytes) Field (DMACR[EMLM] = 0) field

Table 19-24. TCD*n* Word 2 (TCD*n*.nbytes) Field (DMACR[EMLM] = 0) field description

Field	Description
nbytes	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.

When minor loop mapping (DMACR[EMLM] = 1) is enabled, TCD word2 is redefined as four fields: a source minor loop offset enable, a destination minor loop offset enable, a minor loop offset field, and an nbytes field.

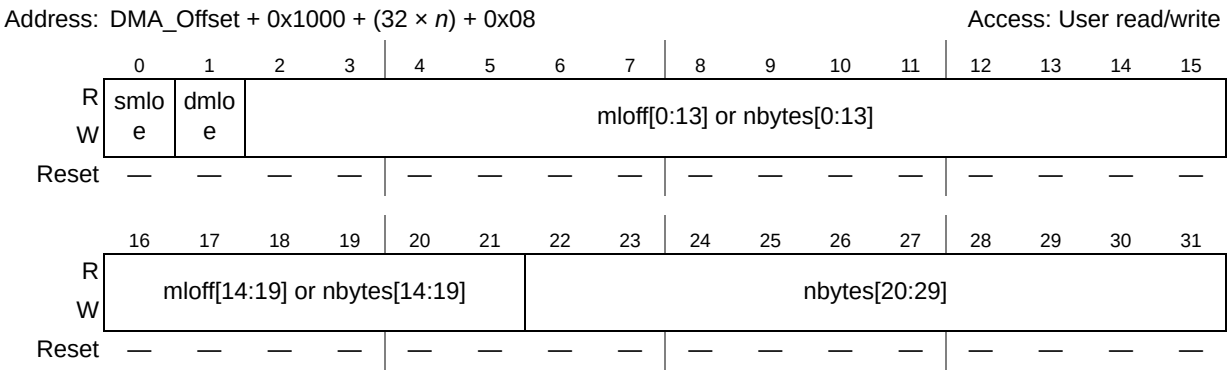


Figure 19-22. TCD*n* Word 2 (TCD*n*.nbytes) Field (DMACR[EMLM] = 1) field

Table 19-25. TCD*n* Word 2 (TCD*n*.nbytes) Field (DMACR[EMLM] = 1) field description

Field	Description
smloe	Source minor loop offset enable. This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.
dmloe	Destination minor loop offset enable. This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.
nbytes[0:19] or mloff[0:19]	Inner “minor” byte transfer count, or Minor loop offset. If both smloe and dmloe are cleared, this field is part of the byte transfer count. If either smloe or dmloe are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.
nbytes	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.

Figure 19-23 and Table 19-26 define word 3 of the TCD*n* structure, the slast field.

Address: DMA_Offset + 0x1000 + (32 × *n*) + 0x0C Access: User read/write

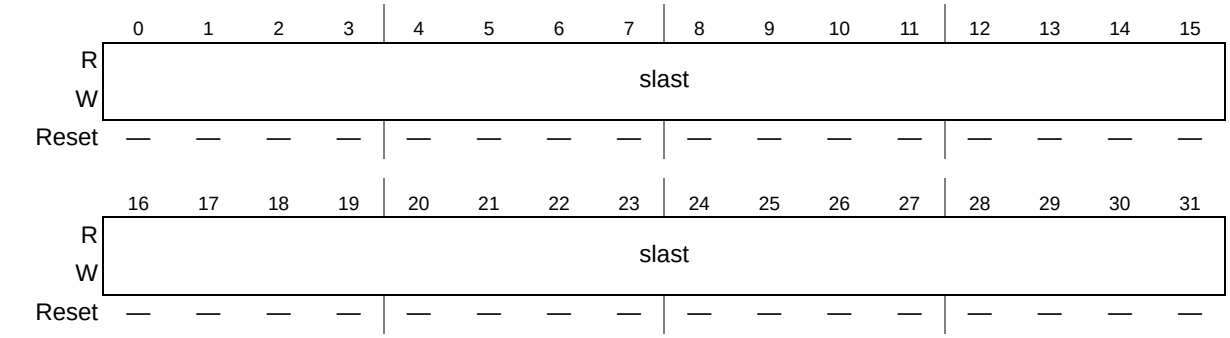


Figure 19-23. TCD*n* Word 3 (TCD*n*.slast) field

Table 19-26. TCD*n* Word 3 (TCD*n*.slast) field description

Field	Description
slast	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.

Figure 19-24 and Table 19-27 define word 4 of the TCD*n* structure, the daddr field.

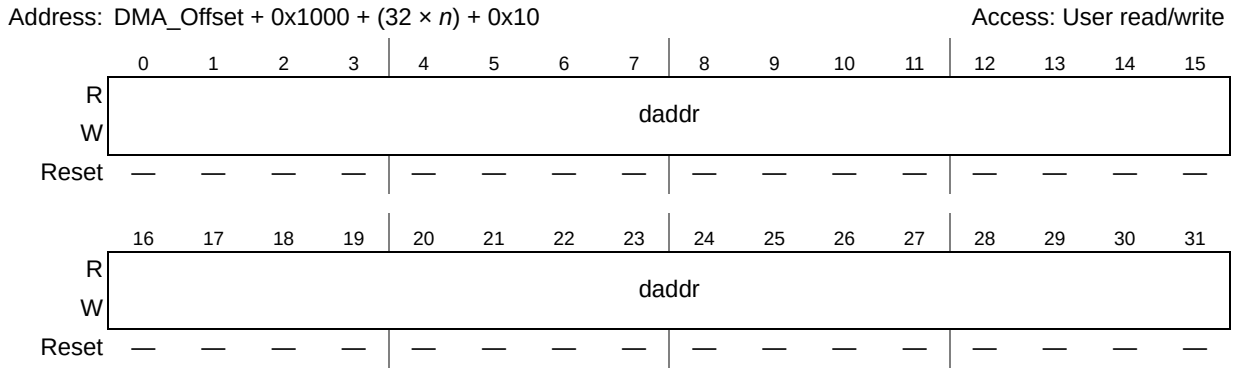


Figure 19-24. TCD*n* Word 4 (TCD*n*.daddr) field

Table 19-27. TCD*n* Word 4 (TCD*n*.daddr) field description

Field	Description
daddr	Destination address. Memory address pointing to the destination data.

Figure 19-25 and Table 19-28 define word 5 of the TCD*n* structure, the citer and doff fields.

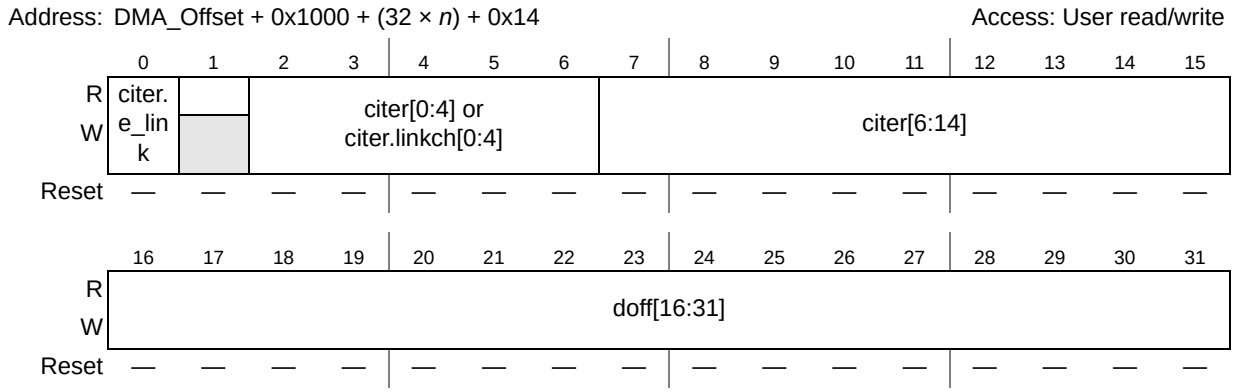


Figure 19-25. TCD*n* Word 5 (TCD*n*.{citer,doff}) fields

Table 19-28. TCD*n* Word 5 (TCD*n*.{citer,doff}) field description

Field	Description
citer.e_link	Enable channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by citer.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. If channel linking is disabled, the citer value is extended to 15 bits in place of a link channel number. If the “major” loop is exhausted, this link mechanism is suppressed in favor of the major.e_link channel linking. *This bit must be equal to the biter.e_link bit, otherwise a configuration error is reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.

Table 19-28. TCD*n* Word 5 (TCD*n*.{citer,doff}) field description (continued)

Field	Description

citer[0:4] or citer.linkch[0:4]	Current “major” iteration count or Link channel number.
	if (TCD.citer.e_link = 0) then
	No channel-to-channel linking (or chaining) is performed after the inner “minor” loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field.
	else
	After the “minor” loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by citer.linkch[0:4] by setting that channel’s TCD.start bit.
	
	The value contained in citer.linkch[0:4] must not exceed the number of implemented channels.

citer[6:14]	Current “major” iteration count. This 9- or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. Once the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the citer field from the beginning iteration count (biter) field. When the citer field is initially loaded by software, it must be set to the same value as that contained in the biter field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
doff[16:31]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Figure 19-26 and Table 19-29 define word 6 of the TCD*n* structure, the dlast_sga field.

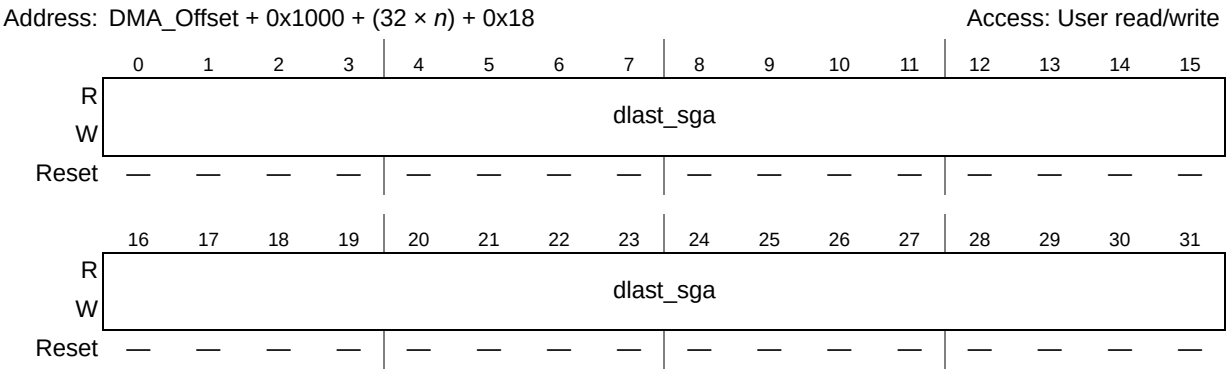


Figure 19-26. TCD*n* Word 6 (TCD*n*.dlast_sga) field

Table 19-29. TCD*n* Word 6 (TCD*n*.dlast_sga) field description

Field	Description

dlast_sga	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather).
	
	if (TCD.e_sg = 0) then
	Adjustment value added to the destination address at the completion of the outer major iteration count.
	
	This value can be applied to “restore” the destination address to the initial value, or adjust the address to reference the next data structure.
	
	else
	This address points to the beginning of a 0-modulo-32 region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32, else a configuration error is reported.

Figure 19-27 and Table 19-30 define word 7 of the TCD*n* structure, the biter and control/status fields.

Address: DMA_Offset + 0x1000 + (32 × *n*) + 0x1C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	biter															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	bwc		0	major.linkch[0:4]					done	active	major.e_link	e_sg	d_req	int_half	int_maj	start
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-27. TCD*n* Word 7 (TCD*n*.{biter,control/status}) fields

Table 19-30. TCD*n* Word 7 (TCD*n*.{biter,control/status}) field description

Field	Description
biter.e_link	Enable channel-to-channel linking on minor loop complete. This is the initial value copied into the citer.e_link field when the major loop is completed. The citer.e_link field controls channel linking during channel execution. *This bit must be equal to the citer.e_link bit, otherwise a configuration error is reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.

Table 19-30. TCD*n* Word 7 (TCD*n*.{biter,control/status}) field description (continued)

Field	Description

biter[0:4] or biter.linkch[0:4]	Beginning “major” iteration count or Beginning Link channel number. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution.
	if (TCD.biter.e_link = 0) then
	No channel-to-channel linking (or chaining) is performed after the inner “minor” loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit biter field.
	else
	After the “minor” loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by biter.linkch[5:0] by setting that channel’s TCD.start bit.
	The value contained in biter.linkch[5:0] must not exceed the number of implemented channels.

biter[6:14]	Beginning “major” iteration count. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the iteration count and linking during channel execution. This 9- or 15-bit count represents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16-bit biter entry is reloaded into the 16-bit citer entry. When the biter field is initially loaded by software, it must be set to the same value as that contained in the citer field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
bwc[0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA module. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, ... sequences until the minor count is exhausted. This field forces the eDMA module to stall after the completion of each read/write access to control the bus request bandwidth seen by the platform’s cross-bar arbitration switch. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the eDMA module as seen by the cross-bar arbitration switch for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No eDMA engine stalls. 01 Dynamic priority elevation. 10 eDMA engine stalls for 4 cycles after each r/w. 11 eDMA engine stalls for 8 cycles after each r/w.

major.linkch [0:4]	Link channel number.
	if (TCD.major.e_link = 0) then
	No channel-to-channel linking (or chaining) is performed after the outer “major” loop counter is exhausted.
	else
	After the “major” loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting that channel’s TCD.start bit.
	The value contained in major.linkch[5:0] must not exceed the number of implemented channels.

done	Channel done. This flag indicates the eDMA module has completed the outer major loop. It is set by the eDMA engine as the citer count reaches zero; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
active	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner minor loop completes or if any error condition is detected.

Table 19-30. TCD*n* Word 7 (TCD*n*.{biter,control/status}) field description (continued)

Field	Description
major.e_link	Enable channel-to-channel linking on major loop complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. *To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.done bit is set.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
e_sg	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. *To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.done bit is set.* 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The dlast_sga field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
d_req	Disable request. If this flag is set, the eDMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero. 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the outer major loop is complete.
int_half	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (citer == (biter >> 1)). This halfway point interrupt request is provided to support double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when biter values are less than two. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
int_maj	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
start	Channel start. If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

19.3 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

19.3.1 eDMA microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into the following submodules:

- eDMA engine
  - *addr_path*: This module implements registered versions of two channel transfer control descriptors: channel “x” and channel “y,” and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. Once a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by DCHPRI*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other addr_path.channel_{x,y}. Once the inner minor loop completes execution, the addr_path hardware writes the new values for the TCD*n*.{saddr, daddr, citer} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCD*n*.citer field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.
  - *data_path*: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The AMBA-AHB read data bus is the primary input, and the AHB write data bus is the primary output.

    The addr_ and data_path modules directly support the 2-stage pipelined AMBA-AHB bus. The addr_path module represents the 1st stage of the bus pipeline (the address phase), while the data_path module implements the 2nd stage of the pipeline (the data phase).
  - *pmodel_charb*: This module implements the first section of the eDMA module’s programming model as well as the channel arbitration logic. The programming model registers are connected to the IPS bus (not shown). The ipd_req[*n*] inputs and dma_ipi_int[*n*] outputs are also connected to this module (via the control logic).
  - *control*: This module provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner ‘minor loop’ byte count has been moved. For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.
- transfer_control_descriptor local memory
  - *memory controller*: This logic implements the required dual-ported controller, handling accesses from both the eDMA engine as well as references from the IPS bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the IPS transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.

— *memory array*: The TCD is implemented using a single-ported, synchronous compiled RAM memory array

19.3.2 DMA basic data flow

The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 19-28, the first segment involves the channel service request. In the diagram, this example uses the assertion of the ipd_req[*n*] signal to request service for channel *n*. Channel service request via software and the TCD*n*.start bit follows the same basic flow as an ipd_req. The ipd_req[*n*] input signal is registered internally and then routed to through the eDMA engine, first through the control module, then into the programming model/channel arbitration (pmodel_charb) module. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path (addr_path) and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the dma_engine.addr_path.channel_{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the dma_engine.addr_path.channel_{x,y} registers.

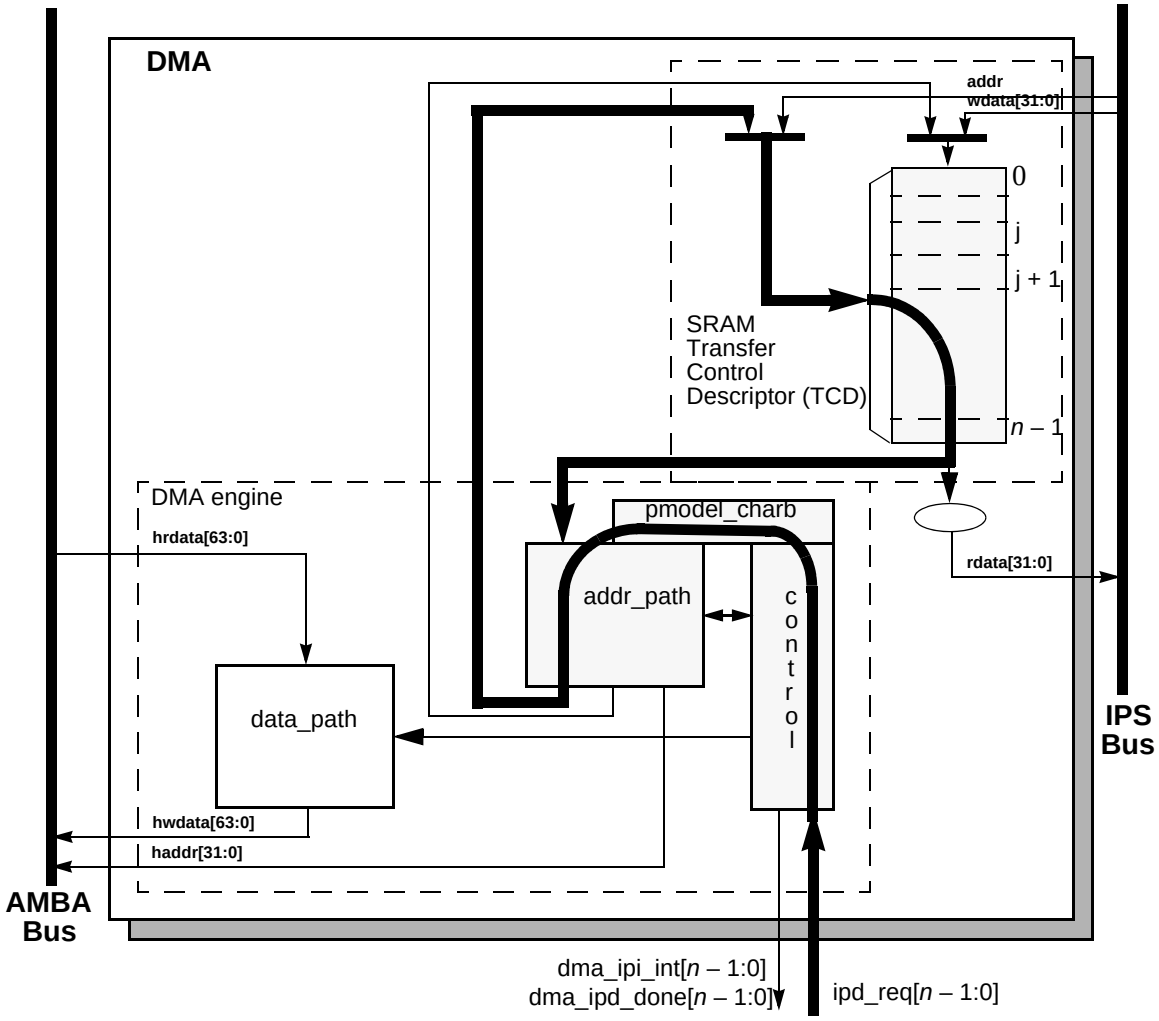


Figure 19-28. DMA operation, part 1

In the second part of the basic data flow as shown in Figure 19-29, the modules associated with the data transfer (addr_path, data_path and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data_path module until it is gated onto the AMBA-AHB bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The dma_ipd_done[*n*] signal is asserted at the end of the minor byte count transfer.

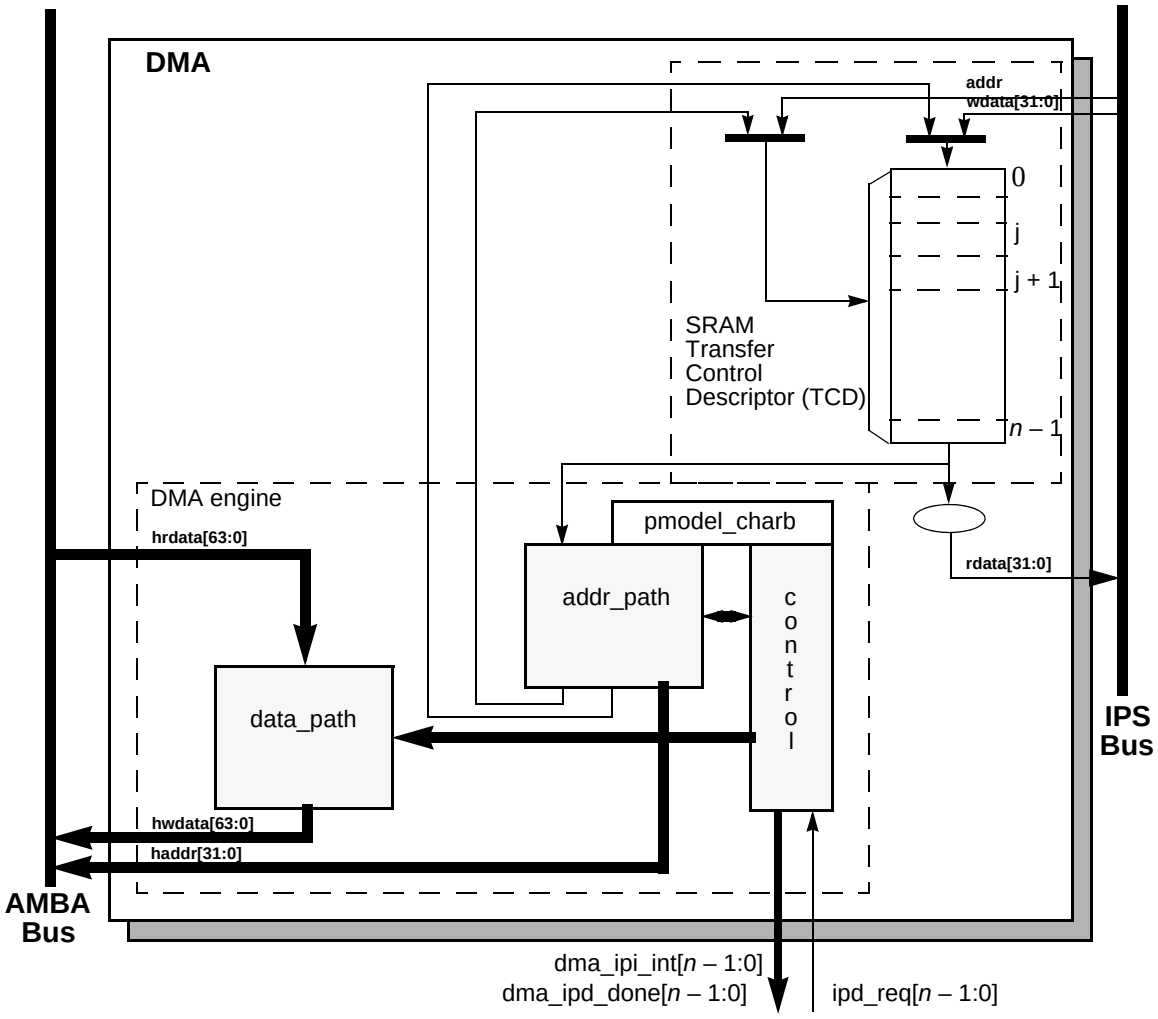


Figure 19-29. DMA operation, part 2

Once the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the addr_path logic performs the required updates to certain fields in the channel's TCD, for example, saddr, daddr, citer. When the outer major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the biter field into the citer. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 19-30.

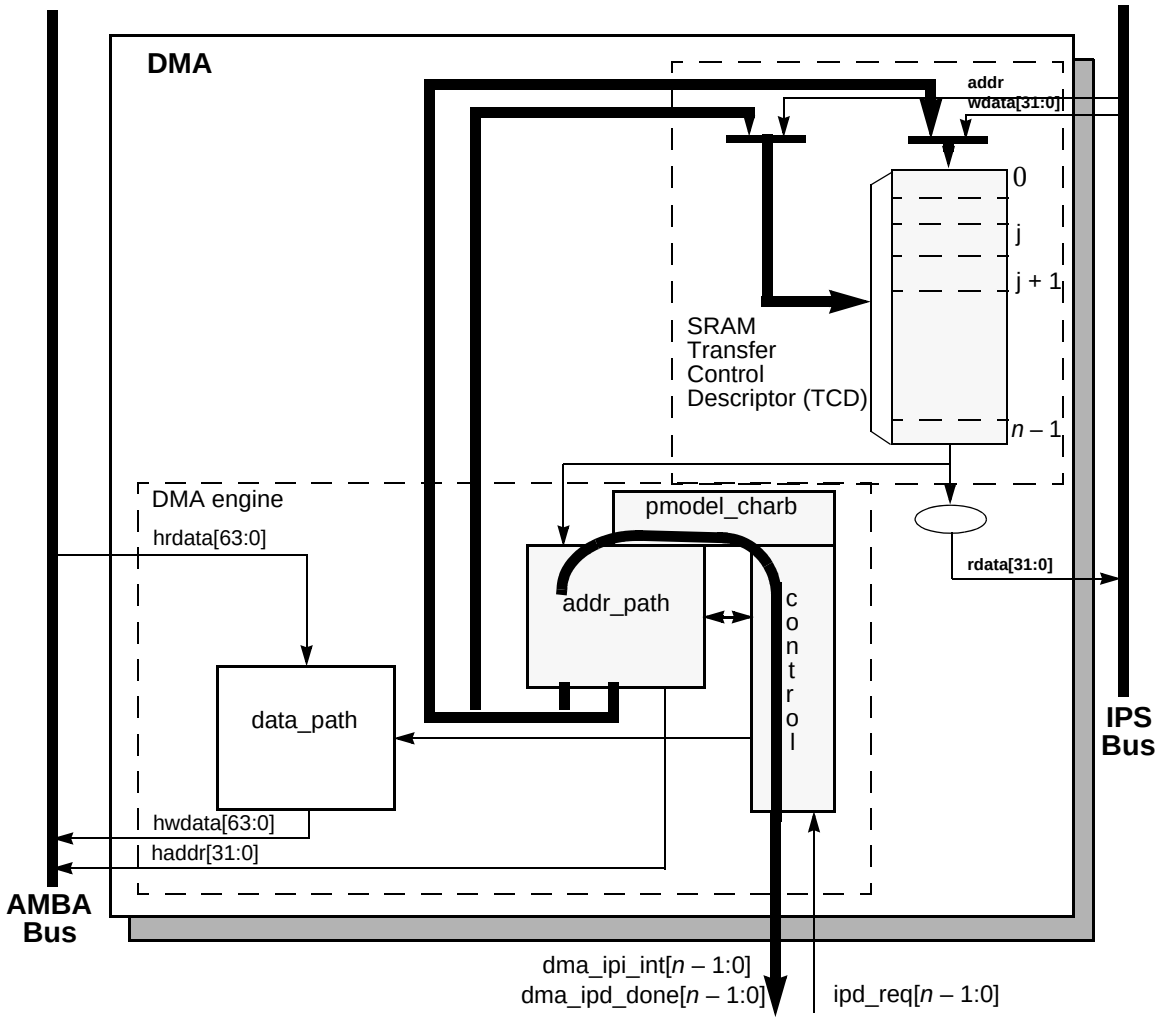


Figure 19-30. DMA operation, part 3

19.3.3 DMA performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA module. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests that can be serviced in a fixed time is a more interesting metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA module also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in Table 19-31. The following assumptions apply to Table 19-31 and Table 19-32:

- SRAM can be accessed with zero wait-states when viewed from the AMBA-AHB data phase
- All IPS accesses are 32 bits in size

Table 19-31 presents a peak transfer rate comparison, measured in megabytes per second. In this table, the SRAM-to-SRAM transfers occur at the native platform datapath width, i.e., either 32- or 64-bits per access. For all transfers involving the IPS bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

Table 19-31. DMA peak transfer rates (MB/s)

Platform speed, width	SRAM-to-SRAM	32-bit IPS-to-SRAM	SRAM-to-32-bit IPS
66.7 MHz, 32-bit	133.3	66.7	53.3
66.7 MHz, 64-bit[1]	266.7	—	—
83.3 MHz, 32-bit	166.7	83.3	66.7
83.3 MHz, 64-bit	333.3	—	—
100.0 MHz, 32-bit	200.0	100.0	80.0
100.0 MHz, 64-bit	400.0	—	—
133.3 MHz, 32-bit	266.7	133.3	106.7
133.3 MHz, 64-bit	533.3	—	—
150.0 MHz, 32-bit	300.0	150.0	120.0
150.0 MHz, 64-bit	600.0	—	—

1 64-bit writes to peripherals are not supported.

The second performance metric is a measure of the number of DMA requests that can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single IPS-mapped operand to/from the platform SRAM. The same timing assumptions used in the previous example apply to this calculation. In particular, this metric also reflects the time required to activate the channel. The eDMA design supports the following hardware service request sequence:

- Cycle 1: ipd_req[*n*] is asserted.
- Cycle 2: The ipd_req[*n*] is registered locally in the eDMA module and qualified (TCD.start bit initiated requests start at this point with the registering of the IPS write to TCD word7).
- Cycle 3: Channel arbitration begins.
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5–6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the eDMA engine is 64 bits, so the entire descriptor can be accessed in four cycles.
- Cycle 7: The first AMBA-AHB read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the platform's crossbar switch, arbitration at the system bus may insert an additional cycle of delay here.
- Cycle 8–?: The last part of the TCD is read in. This cycle represents the first data phase for the read, and the address phase for the destination write.
  The exact timing from this point is a function of the response times for the channel's read and write accesses. In this case of an IPS read and a platform SRAM write, the combined data phase time is 4 cycles. For an SRAM read and IPS write, it is 5 cycles.
- Cycle ? + 1: This cycle represents the data phase of the last destination write.

- Cycle ? + 2: The eDMA engine completes the execution of the inner minor loop and prepares to write back the required TCD*n* fields into the local memory. TCD word7 is read and checked for channel linking or scatter/gather requests.
- Cycle ? + 3: The appropriate fields in the first part of the TCD*n* are written back into the local memory.
- Cycle ? +4 : The fields in the second part of the TCD*n* are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle ? + 5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the AHB system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with IPS-to-SRAM (4 cycles) and SRAM-to-IPS (5 cycles), DMA requests can be processed every 11.5 cycles (4 + (4 + 5) ÷ 2 + 3). This is the time from Cycle 4 to Cycle "? + 5". The resulting peak request rate, as a function of the platform frequency, is shown in Table 19-32. This metric represents millions of requests per second.

Table 19-32. DMA peak request rate (MReq/sec)

Platform speed	Request rate (zero wait state)	Request rate (with wait states)
66.6 MHz	7.4	5.8
83.3 MHz	9.2	7.2
100.0 MHz	11.1	8.7
133.3 MHz	14.8	11.6
150.0 MHz	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

$$PEAKreq = freq \div [\ entry + (1 + read\_ws) + (1 + write\_ws) + exit\ ] \qquad \textit{Eqn. 19-1}$$

where:

PEAKreq	**peak request rate**
freq	**platform frequency**
entry	**channel startup (4 cycles)**
read_ws	**wait states seen during the system bus read data phase**
write_ws	**wait states seen during the system bus write data phase**
exit	**channel shutdown (3 cycles)**

For example: consider a platform with the following characteristics:

- Platform SRAM can be accessed with one wait-state when viewed from the AMBA-AHB data phase
- All IPS reads require 2 wait states, and IPS writes 3 wait-states, again viewed from the system bus data phase

- Platform operates at 150 MHz

For an SRAM to IPS transfer:

PEAKreq = 150 MHz ÷ [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For an IPS to SRAM transfer:

PEAKreq = 150 MHz ÷ [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/sec

Assuming an even distribution of the two transfer types, the average Peak Request Rate would be:

PEAKreq = (11.5 Mreq/sec + 12.5 Mreq/sec) ÷ 2 = 12.0 Mreq/sec

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (where no channel is executing, DMA is idle) are:

- 11 cycles for a software (TCD.start bit) request
- 12 cycles for a hardware (ipd_req signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the ipd_req signals. For the peak request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

NOTE

When channel linking or scatter/gather is enabled, a two cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

19.4 Initialization/application information

19.4.1 DMA initialization

A typical initialization of the eDMA module is:

1. Write the DMACR register if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRI*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the DMAEEI registers if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the DMAERQ register.
6. Request channel service by either software (setting the TCD.start bit) or by hardware (slave device asserting its ipd_req signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine reads the entire TCD for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.saddr) to the destination (as defined by the destination address, TCD.daddr) continue until

the specified number of bytes (TCD.nbytes) have been transferred. When the transfer is complete, the eDMA engine's local TCD.saddr, TCD.daddr, and TCD.citer are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed (for example, interrupts, major loop channel linking, and scatter/gather operations, if enabled).

19.4.2 DMA programming errors

The eDMA module performs various tests on the Transfer Control Descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of two errors; Group Priority Error and Channel Priority Error, GPE and CPE in the DMAES register respectively.

For all error types other than Group or Channel Priority Errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

The sequence listed below is correct. For item 2, the dma_ipd_ack{done} lines assert only if the selected channel is requesting service via the ipd_req signal. The typical application enables error interrupts for all channels. So the user gets an error interrupt, but the channel number for the DMAERR register and the error interrupt request line may be wrong because they reflect the selected channel.

Channel priority errors are identified within a group after that group has been selected as the active group. For example:

1. The eDMA module is configured for fixed group and fixed channel arbitration modes.
2. Group1 is the highest priority and all channels are unique in that group.
3. Group0 is the next highest priority and has two channels with the same priority level.
4. If Group1 has any service requests, those requests are executed.
5. Once all of Group1 requests have completed, Group0 becomes the next active group.
6. If Group0 has a service request, then an undefined channel in Group0 is selected and a channel priority error occurs.
7. This repeats until the all the Group0 requests have been removed or a higher priority Group1 request comes in.

A group priority error is global and any request in any group causes a group priority error.

In general, if priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

19.4.3 DMA arbitration mode considerations

19.4.3.1 Fixed group arbitration, fixed channel arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA module is programmed so the channels within one group use "fixed"

priorities, and that group is assigned the highest “fixed” priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller — i.e., no other groups will be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request.

The advantage of this scenario is that latency can be small for channels that need to be serviced quickly.

Preemption is available in this scenario only.

19.4.3.2 Round-robin group arbitration, fixed channel arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with an service request and rotating through to the lowest group number containing a service request.

Once the channel request is serviced, the group round-robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues round-robin, always servicing the highest priority channel in the next group in the sequence, or just skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round-robin service rate, then that channel always is serviced before lower priority channels in the same group, and thus the lower priority channels never are serviced.

The advantage of this scenario is that no one group uses all the DMA bandwidth.

The highest priority channel selection latency is potentially greater than fixed/fixed arbitration.

Excessive request rates on high priority channels can prevent the servicing of lower priority channels in the same group.

19.4.3.3 Round-robin group arbitration, round-robin channel arbitration

Groups are serviced as described in Section 19.4.3.2, Round-robin group arbitration, fixed channel arbitration, but this time channels are serviced in channel number order. Only one channel is serviced from each requesting group for each round-robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round-robin manner, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are lost, but at least one channel will be serviced.

This scenario configures all channels to be serviced at some point, regardless of the request rates. However, the potential latency could be quite high.

All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

19.4.3.4 Fixed group arbitration, round-robin channel arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in Section 19.4.3.1, Fixed group arbitration, fixed channel arbitration, but all the channels in the highest priority group are serviced.

Service latency ia short on the highest priority group, but can become much longer as the group priority decreases.

19.4.4 DMA transfer

19.4.4.1 Single request

To perform a transfer of *n* bytes of data with one activation, set the major loop to one (TCD.citer = TCD.biter = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. Once the transfer is complete, the TCD.done bit is e set and an interrupt is generated (if properly enabled).

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA module is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

Table 19-33. Example TCD entry for single request

Entry	Value
TCD.citer	= TCD.biter = 1
TCD.nbytes	= 16
TCD.saddr	= 0x1000
TCD.soff	= 1
TCD.ssize	= 0
TCD.slast	= –16
TCD.daddr	= 0x2000
TCD.doff	= 4
TCD.dsize	= 2
TCD.dlast_sga	= –16
TCD.int_maj	= 1

Table 19-33. Example TCD entry for single request (continued)

Entry	Value
TCD.start	= 1 (TCD.word7 should be written last after all other fields have been initialized)
All other TCD fields	= 0

This generates the following sequence of events:

1. IPS write to the TCD.start bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a. read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b. write_word(0x2000) → *first iteration of the minor loop*
   c. read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d. write_word(0x2004) → *second iteration of the minor loop*
   e. read_byte(0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B)
   f. write_word(0x2008) → *third iteration of the minor loop*
   g. read_byte(0x100C), read_byte(0x100D), read_byte(0x100E), read_byte(0x100F)
   h. write_word(0x200C) → *last iteration of the minor loop* → *major loop complete*
6. eDMA engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 1 (TCD.biter)
7. eDMA engine writes: TCD.active = 0, TCD.done = 1, DMAINT[*n*] = 1
8. The channel retires.

The eDMA module goes idle or services next channel.

19.4.4.2 Multiple requests

The next example is the same as the previous example, with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA module is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests is enabled in the DMAERQ register, channel service requests are initiated by the slave device.

Table 19-34. Example TCD entry for multiple requests

Entry	Value
TCD.citer	= TCD.biter = 2

Table 19-34. Example TCD entry for multiple requests (continued)

Entry	Value
TCD.slast	= –32
TCD.dlast_sga	= –32

This would generate the following sequence of events:

1. First hardware (ipd_req) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a. read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b. write_word(0x2000) → *first iteration of the minor loop*
   c. read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d. write_word(0x2004) → *second iteration of the minor loop*
   e. read_byte(0x1008), read_byte(0x1009), read_byte(0x100A), read_byte(0x100B)
   f. write_word(0x2008) → *third iteration of the minor loop*
   g. read_byte(0x100C), read_byte(0x100D), read_byte(0x100E), read_byte(0x100F)
   h. write_word(0x200C) → *last iteration of the minor loop*
6. eDMA engine writes: TCD.saddr = 0x1010, TCD.daddr = 0x2010, TCD.citer = 1
7. eDMA engine writes: TCD.active = 0
8. The channel retires → *one iteration of the major loop*

The eDMA module goes idle or services next channel.

9. Second hardware (ipd_req) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a. read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
    b. write_word(0x2010) → *first iteration of the minor loop*
    c. read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
    d. write_word(0x2014) → *second iteration of the minor loop*
    e. read_byte(0x1018), read_byte(0x1019), read_byte(0x101A), read_byte(0x101B)
    f. write_word(0x2018) → *third iteration of the minor loop*

g. read_byte(0x101C), read_byte(0x101D), read_byte(0x101E), read_byte(0x101F)

h. write_word(0x201C) → *last iteration of the minor loop* → *major loop complete*

14. eDMA engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 2 (TCD.biter)
15. eDMA engine writes: TCD.active = 0, TCD.done = 1, DMAINT[*n*] = 1
16. The channel retires → *major loop complete*

The eDMA module goes idle or services the next channel.

19.4.5 TCD status

19.4.5.1 Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.citer field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the TCD.start bit AND the TCD.active bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.start was written to a one. Polling the TCD.active bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.start = 1, TCD.active = 0, TCD.done = 0 (channel service request via software)
2. TCD.start = 0, TCD.active = 1, TCD.done = 0 (channel is executing)
3. TCD.start = 0, TCD.active = 0, TCD.done = 0 (channel has completed the minor loop and is idle)
   or
   TCD.start = 0, TCD.active = 0, TCD.done = 1 (channel has completed the major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.citer field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. ipd_req asserts (channel service request via hardware)
2. TCD.start = 0, TCD.active = 1, TCD.done = 0 (channel is executing)
3. TCD.start = 0, TCD.active = 0, TCD.done = 0 (channel has completed the minor loop and is idle)
   or
   TCD.start = 0, TCD.active = 0, TCD.done = 1 (channel has completed the major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.done bit.

The TCD.start bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

19.4.5.2 Active channel TCD reads

The eDMA module reads back the 'true' TCD.saddr, TCD.daddr, and TCD.nbytes values if read while a channel is executing. The 'true' values of the saddr, daddr, and nbytes are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (saddr and daddr) and nbytes (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

19.4.5.3 Preemption status

Preemption is only available when *fixed* arbitration is selected for *both* group and channel arbitration modes. Preemption is applicable when a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (constantly rotating) when round-robin arbitration mode is selected.

The TCD.active bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.active bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

The worst case latency when switching to a preempt channel is the summation of:

- Arbitration latency (2 cycles)
- Bandwidth control stalls (if enabled)
- The time to execute two read/write sequences (including AHB bus holds; a system dependency driven by the slave devices or the crossbar)

19.4.6 Channel linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.start bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.citer.e_link field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

Table 19-35. Example TCD entry for channel linking

Entry	Value
TCD.citer.e_link	= 1
TCD.citer.linkch	= 0xC
TCD.citer value	= 0x4

Table 19-35. Example TCD entry for channel linking (continued)

Entry	Value
TCD.major.e_link	= 1
TCD.major.linkch	= 0x7

will execute as:

1. Minor loop done → set channel 12 TCD.start bit
2. Minor loop done → set channel 12 TCD.start bit
3. minor loop done → set channel 12 TCD.start bit
4. minor loop done, major loop done → set channel 7 TCD.start bit

When minor loop linking is enabled (TCD.citer.e_link = 1), the TCD.citer field uses a 9-bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.citer.e_link = 0), the TCD.citer field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.citer.linkch field are concatenated onto the citer value to increase the range of the citer.

NOTE

The TCD.citer.e_link bit and the TCD.biter.e_link bit must equal or a configuration error is reported. The citer and biter vector widths must be equal to calculate the major loop, half-way done interrupt point.

19.4.7 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

19.4.7.1 Dynamic priority changing

The following two options are recommended for dynamically changing ***channel*** priority levels:

1. Switch to round-robin channel arbitration mode, change the channel priorities, then switch back to fixed arbitration mode.
2. Disable all the channels within a group, then change the channel priorities within that group only, then enable the appropriate channels.

The following two options are available for dynamically changing ***group*** priority levels:

1. Switch to round-robin group arbitration mode, change the group priorities, then switch back to fixed arbitration mode.
2. Disable ALL channels, change the group priorities, then enable the appropriate channels.

19.4.7.2 Dynamic channel linking

Dynamic channel linking is the process of changing the TCD.major.e_link during channel execution. These bits are read from the TCD local memory at the ***end*** of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link request:

1. Set the TCD.major.e_link bit.
2. Read back the TCD.major.e_link bit.
3. Test the TCD.major.e_link request status:
   a. If the bit is set, the dynamic link attempt was successful.
   b. If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

For this request, the TCD local memory controller forces the TCD.major.e_link to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set indicating the major loop is complete.

NOTE

The user must clear the TCD.done bit before writing the TCD.major.e_link. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

19.4.7.3 Dynamic scatter/gather

Dynamic scatter/gather is the process of setting the TCD.e_sg bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic scatter/gather operation by enabling the TCD.e_sg bit at the same time the eDMA engine is retiring the channel. The TCD.e_sg would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired.

Two methods for this coherency model are shown in the following subsections. Method 1 has the advantage of reading the major.linkch field and the e_sg bit with a single read. For both dynamic channel linking and scatter/gather requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.word7 if that channel's TCD.done bit is set indicating the major loop is complete.

NOTE

The user must clear the TCD.done bit before writing the TCD.major.e_link or TCD.e_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

19.4.7.3.1 Method 1 (channel not using major loop channel linking)

For a channel not using major loop channel linking, the coherency model in Table 19-36 may be used for a dynamic scatter/gather request.

When the TCD.major.e_link bit is zero, the TCD.major.linkch field is not used by the eDMA. In this case, the TCD.major.linkch bits may be used for other purposes. This method uses the TCD.major.linkch field as a TCD indentification (ID).

Table 19-36. Coherency model for method 1

Step	Action
1	When the descriptors are built, write a unique TCD ID in the TCD.major.linkch field for each TCD associated with a channel using dynamic scatter/gather.
2	Write 1b to theTCD.d_req bit. **Note:** Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offest value.
3	Write theTCD.dlast_sga field with the scatter/gather address.
4	Write 1b to the TCD.e_sg bit.
5	Read back the 16 bit TCD control/status field.
6	Test the TCD.e_sg request status and TCD.major.linkch value: • If e_sg = 1b, the dynamic link attempt was successful. • If e_sg = 0b and the major.linkch (ID) did not change, the attempted dynamic link did not succeed (the channel was already retiring). • If e_sg = 0b and the major.linkch (ID) changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).

19.4.7.3.2 Method 2 (channel using major loop linking)

For a channel using major loop channel linking, the coherency model in Table 19-37 may be used for a dynamic scatter/gather request. This method uses the TCD.dlast_sga field as a TCD indentification (ID).

Table 19-37. Coherency model for method 2

Step	Action
1	Write 1b to theTCD.d_req bit. **Note:** Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offest value.
2	Write theTCD.dlast_sga field with the scatter/gather address.

Table 19-37. Coherency model for method 2 (continued)

Step	Action
3	Write 1b to the TCD.e_sg bit.
4	Read back the TCD.e_sg bit.
5	Test the TCD.e_sg request status: • If e_sg = 1b, the dynamic link attempt was successful. • If e_sg = 0b, read the 32 bit TCD dlast_sga field. • If e_sg = 0b and the dlast_sga did not change, the attempted dynamic link did not succeed (the channel was already retiring). • If e_sg = 0b and the dlast_sga changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).

19.4.8 Hardware request release timing

This section provides a timing diagram for deasserting the ipd_req hardware request signal. Figure 19-31 shows two read write sequences with grey indicating the release of the ipd_req hardware request signal.

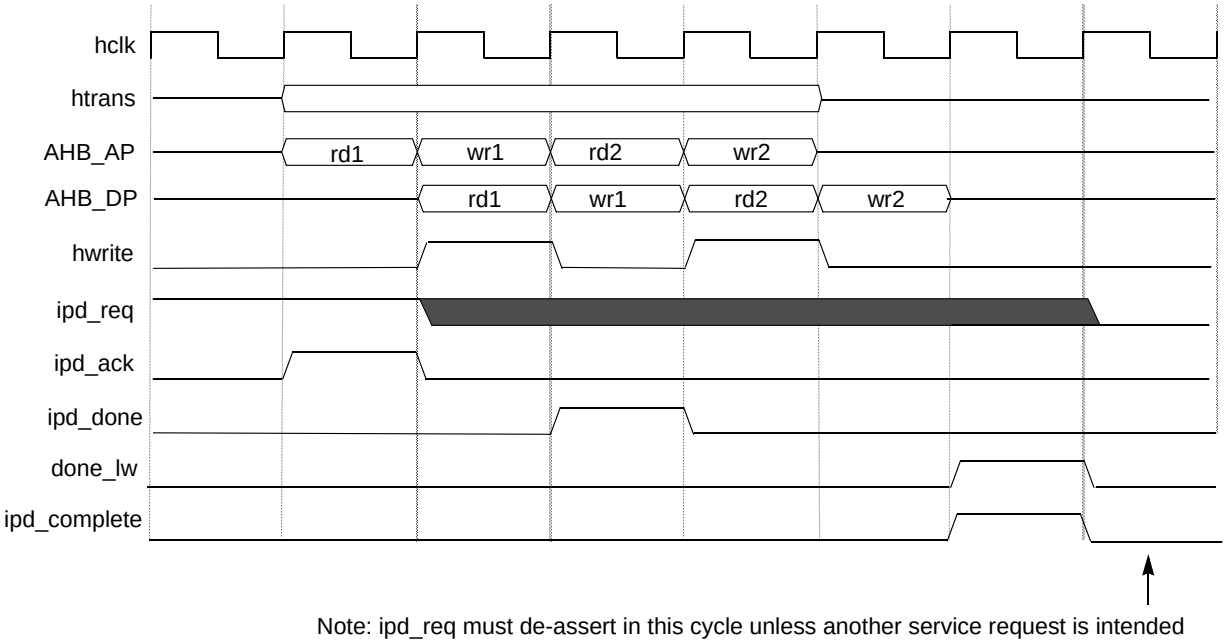


Figure 19-31. ipd_req hardware handshake

Chapter 20
eDMA Channel Mux (DMACHMUX)

20.1 Introduction

20.1.1 Overview

The DMACHMUX allows routing 52 DMA peripheral sources (called slots) to 32 DMA channels. This is illustrated in Figure 20-1.

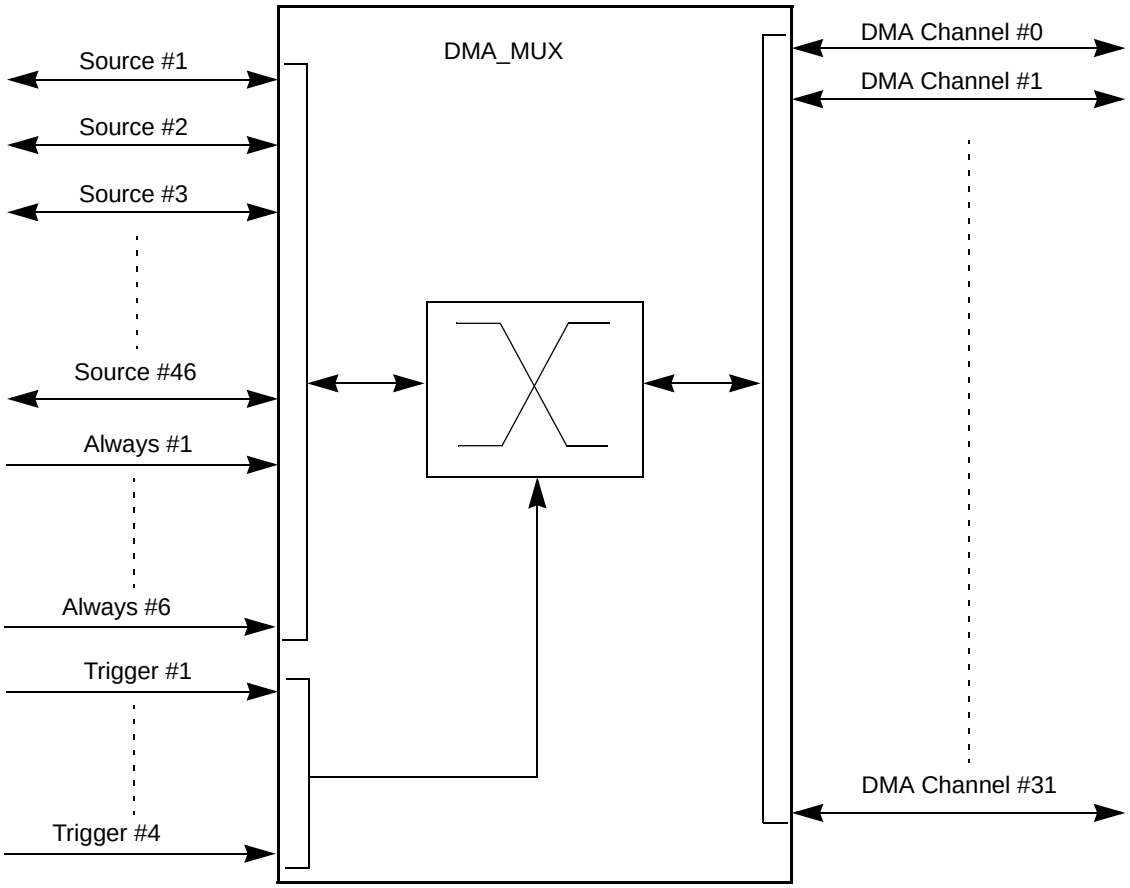


Figure 20-1. DMACHMUX block diagram

20.1.2 Features

The DMACHMUX provides these features:

- 46 peripheral slots (plus 6 always-on slots) can be routed to 32 channels
- 32 independently selectable DMA channels routers
  - The first 4 channels additionally provide a trigger functionality

- Each channel router can be assigned to one of 46 possible peripheral DMA slots or to one of the 6 always-on slots.

20.1.3 Modes of operation

The following operation modes are available:

- Disabled Mode

  In this mode, the DMA channel is disabled. Since disabling and enabling of DMA channels is done primarily via the DMA configuration registers, this mode is used mainly as the reset state for a DMA channel in the DMACHMUX. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (for example, changing the period of a DMA trigger).
- Normal Mode

  In this mode, a DMA source (such as DSPI transmit or DSPI receive for example) is routed directly to the specified DMA channel. The operation of the DMACHMUX in this mode is completely transparent to the system.
- Periodic Trigger Mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the Periodic Interrupt Timer (PIT). This mode is only available for channels 0-4.

20.2 External signal description

20.2.1 Overview

The DMACHMUX has no external pins.

20.3 Memory map and register description

This section provides a detailed description of all memory-mapped registers in the DMACHMUX.

Table 20-1 shows the memory map for the DMACHMUX. All addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMACHMUX.

Table 20-1. DMACHMUX module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	DMACHMUX_0	0xFFFD_C000
	DMACHMUX_1	
Decoupled Parallel Mode (DPM)	DMACHMUX_0	0xFFFD_C000 (same as LSM)
	DMACHMUX_1	0xC3FA_C000

Table 20-2. DMACHMUX memory map

Offset from DMACHMUX_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Channel #0 Configuration (CHCONFIG0)	R/W	0x00	on page 546
0x0001	Channel #1 Configuration (CHCONFIG1)	R/W	0x00	on page 546
0x0002	Channel #2 Configuration (CHCONFIG2)	R/W	0x00	on page 546
0x0003	Channel #3 Configuration (CHCONFIG3)	R/W	0x00	on page 546
0x0004	Channel #4 Configuration (CHCONFIG4)	R/W	0x00	on page 546
0x0005	Channel #5 Configuration (CHCONFIG5)	R/W	0x00	on page 546
0x0006	Channel #6 Configuration (CHCONFIG6)	R/W	0x00	on page 546
0x0007	Channel #7 Configuration (CHCONFIG7)	R/W	0x00	on page 546
0x0008	Channel #8 Configuration (CHCONFIG8)	R/W	0x00	on page 546
0x0009	Channel #9 Configuration (CHCONFIG9)	R/W	0x00	on page 546
0x000A	Channel #10 Configuration (CHCONFIG10)	R/W	0x00	on page 546
0x000B	Channel #11 Configuration (CHCONFIG11)	R/W	0x00	on page 546
0x000C	Channel #12 Configuration (CHCONFIG12)	R/W	0x00	on page 546
0x000D	Channel #13 Configuration (CHCONFIG13)	R/W	0x00	on page 546
0x000E	Channel #14 Configuration (CHCONFIG14)	R/W	0x00	on page 546
0x000F	Channel #15 Configuration (CHCONFIG15)	R/W	0x00	on page 546
0x0010	Channel #16 Configuration (CHCONFIG16)	R/W	0x00	on page 546
0x0011	Channel #17 Configuration (CHCONFIG17)	R/W	0x00	on page 546
0x0012	Channel #18 Configuration (CHCONFIG18)	R/W	0x00	on page 546
0x0013	Channel #19 Configuration (CHCONFIG19)	R/W	0x00	on page 546
0x0014	Channel #20 Configuration (CHCONFIG20)	R/W	0x00	on page 546
0x0015	Channel #21 Configuration (CHCONFIG21)	R/W	0x00	on page 546
0x0016	Channel #22 Configuration (CHCONFIG22)	R/W	0x00	on page 546
0x0017	Channel #23 Configuration (CHCONFIG23)	R/W	0x00	on page 546
0x0018	Channel #24 Configuration (CHCONFIG24)	R/W	0x00	on page 546
0x0019	Channel #25 Configuration (CHCONFIG25)	R/W	0x00	on page 546
0x001A	Channel #26 Configuration (CHCONFIG26)	R/W	0x00	on page 546
0x001B	Channel #27 Configuration (CHCONFIG27)	R/W	0x00	on page 546
0x001C	Channel #28 Configuration (CHCONFIG28)	R/W	0x00	on page 546
0x001D	Channel #29 Configuration (CHCONFIG29)	R/W	0x00	on page 546
0x001E	Channel #30 Configuration (CHCONFIG30)	R/W	0x00	on page 546

Table 20-2. DMACHMUX memory map (continued)

Offset from DMACHMUX_BASE	Register	Access[1]	Reset Value[2]	Location
0x001F	Channel #31 Configuration (CHCONFIG31)	R/W	0x00	on page 546
0x0020–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit READ/WRITE to address ‘Base + 0x00’, but performing a 32-bit access to address ‘Base + 0x01’ is illegal.

20.3.1 Register descriptions

The following memory-mapped registers are available in the DMACHMUX.

20.3.1.1 Channel Configuration Registers (CHCONFIG#*n*)

Each of the DMA channels can be independently enabled/disabled and associated with one of the DMA slots (peripheral slots or always-on slots) in the system.

Address: Base + #*n* Access: User read/write

	0	1	2	3	4	5	6	7
R	ENBL	TRIG	SOURCE					
W								
Reset	0	0	0	0	0	0	0	0

Figure 20-2. Channel Configuration Registers (CHCONFIG#*n*)

Table 20-3. CHCONFIG*n* field descriptions

Field	Description
ENBL	DMA Channel Enable. ENBL enables the DMA Channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMACHMUX. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled.
TRIG	DMA Channel Trigger Enable (for triggered channels only). TRIG enables the periodic trigger capability for the DMA Channel. 0 Triggering is disabled. If triggering is disabled and the ENBL bit is set, the DMA Channel routes the specified source to the DMA channel. 1 Triggering is enabled.
SOURCE	DMA Channel Source (slot). SOURCE specifies which DMA source, if any, is routed to a particular DMA channel (see Section 20.4, DMACHMUX request source slot mapping).

Table 20-4. Channel and trigger enabling

ENBL	TRIG	Function	Mode
0	X	DMA Channel is disabled	Disabled Mode
1	0	DMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	DMA Channel is enabled with triggering	Periodic Trigger Mode

NOTE

Setting multiple CHCONFIG registers with the same source value will result in unpredictable behavior.

NOTE

Before changing the trigger or source settings a DMA channel must be disabled via the CHCONFIG[*#n*].ENBL bit.

20.4 DMACHMUX request source slot mapping

Table 20-5 defines the mapping of the DMACHMUX source slots to the interrupt request sources on the device. Table 20-5 is valid for both instantiations of the DMACHMUX module.

Table 20-5. DMACHMUX source slot mapping

DMACHMUX source slot #	Source module	Source resource
1	DSPI_0	DSPI_TFFF
2	DSPI_0	DSPI_RFDF
3	DSPI_1	DSPI_TFFF
4	DSPI_1	DSPI_RFDF
5	DSPI_2	DSPI_TFFF
6	DSPI_2	DSPI_RFDF
7	CTU	CTU
8	CTU	FIFO0
9	CTU	FIFO1
10	CTU	FIFO2
11	CTU	FIFO3
12	FlexPWM_0	comp_val
13	FlexPWM_0	capt
14	eTimer_0	DREQ0
15	eTimer_0	DREQ1
16	eTimer_1	DREQ0
17	eTimer_1	DREQ1

Table 20-5. DMACHMUX source slot mapping (continued)

DMACHMUX source slot #	Source module	Source resource
18	eTimer_2	DREQ0
19	eTimer_2	DREQ1
20	ADC_0	DMA
21	ADC_1	DMA
22	LINFlexD_0	Transmit
23	LINFlexD_0	Receive
24	LINFlexD_1	Transmit
25	LINFlexD_1	Receive
26	FlexPWM_1	comp_val
27	FlexPWM_1	capt
28	CTU_1	CTU
29	CTU_1	FIFO0
30	CTU_1	FIFO1
31	CTU_1	FIFO2
32	CTU_1	FIFO3
33	ADC_2	DMA
34	ADC_3	DMA
35	LINFlexD_2	Transmit
36	LINFlexD_2	Receive
37	LINFlexD_3	Transmit
38	LINFlexD_3	Receive
39	FLEXPWM_2	comp_val
40	FLEXPWM_2	capt
41	IIC_0	Transmit
42	IIC_0	Receive
43	IIC_1	Transmit
44	IIC_1	Receive
45	IIC_2	Transmit
46	IIC_2	Receive
47–57	Not used	
58	Always Requestor	—
59	Always Requestor	—
60	Always Requestor	—

Table 20-5. DMACHMUX source slot mapping (continued)

DMACHMUX source slot #	Source module	Source resource
61	Always Requestor	—
62	Always Requestor	—
63	Always Requestor	—

20.5 DMACHMUX trigger inputs

Table 20-6 defines the signal sources for the trigger support of the first #TRG channels of the DMACHMUX. Table 20-6 is valid for both instantiations of the DMACHMUX module.

Table 20-6. DMACHMUX trigger sources

Source Module	Source Signal	DMACHMUX Channel Trigger #
PIT	Trigger Channel 0	0
PIT	Trigger Channel 1	1
PIT	Trigger Channel 2	2
PIT	Trigger Channel 3	3

20.6 Functional description

The primary purpose of the DMACHMUX is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMACHMUX is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in Section 20.7.2, Enabling and configuring sources, is followed, the configuration of the DMACHMUX may be changed during the normal operation of the system.

Functionally, the DMACHMUX channels may be divided into two classes:

- Channels that implement the normal routing functionality plus periodic triggering capability
- Channels that implement only the normal routing functionality

20.6.1 DMA channels with periodic triggering capability

Besides the normal routing functionality, the first 4 channels of the DMACHMUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the Periodic Interrupt Timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. For more information, see Chapter 42, Periodic Interrupt Timer (PIT).

NOTE

Because of the dynamic nature of the system (for example, DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

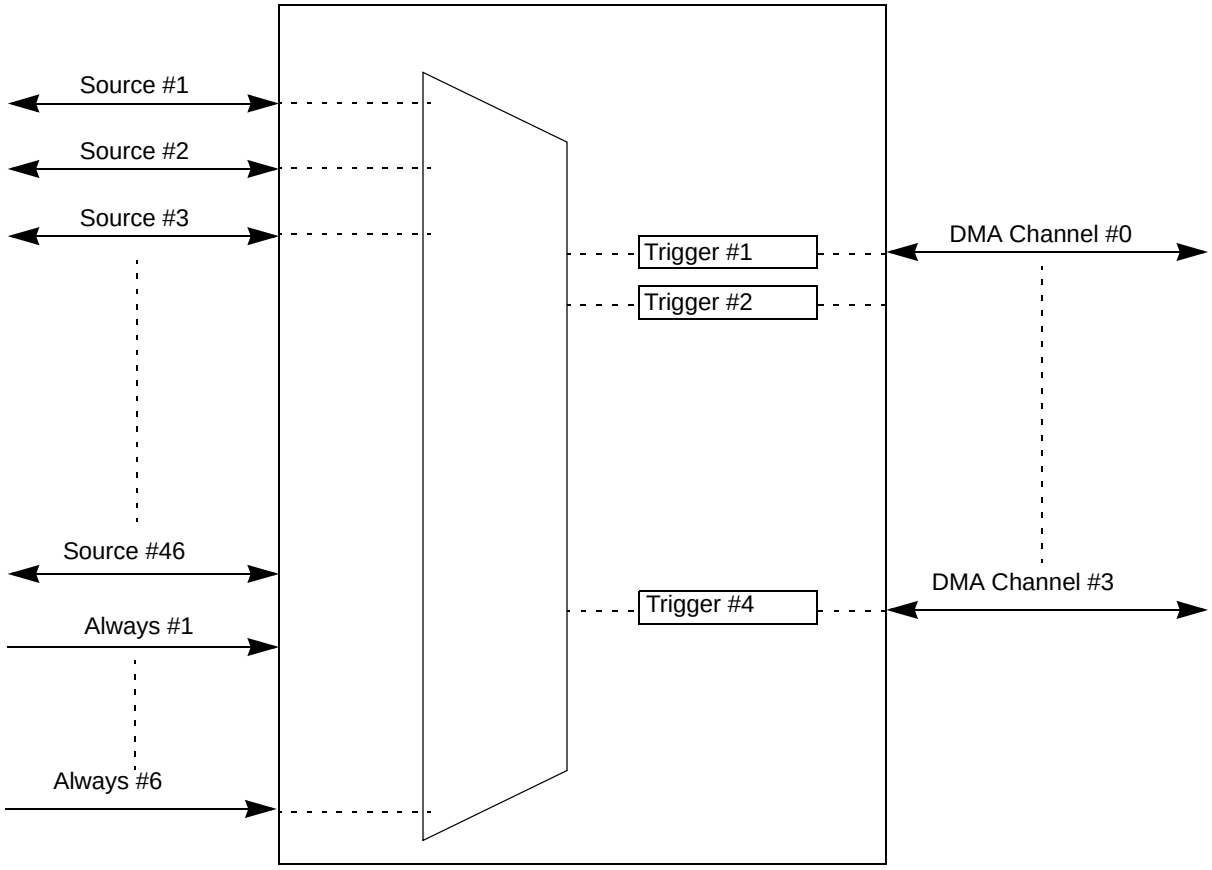


Figure 20-3. DMACHMUX triggered channels

The DMA channel triggering capability allows the system to “schedule” regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen. This is illustrated in Figure 20-4.

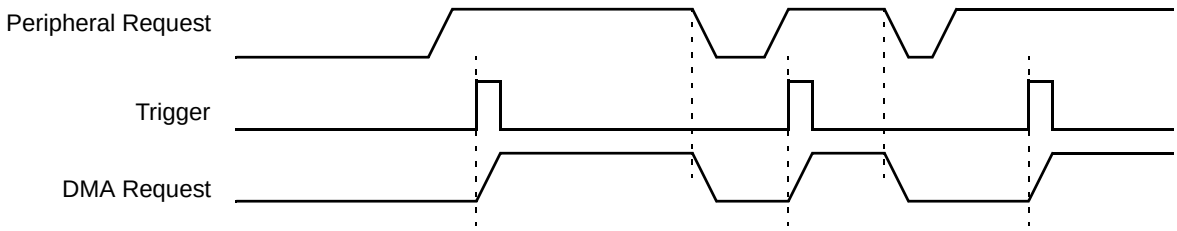


Figure 20-4. DMACHMUX channel triggering: normal operation

Once the DMA request has been serviced, the peripheral negates its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that trigger is ignored. This situation is illustrated in Figure 20-5.

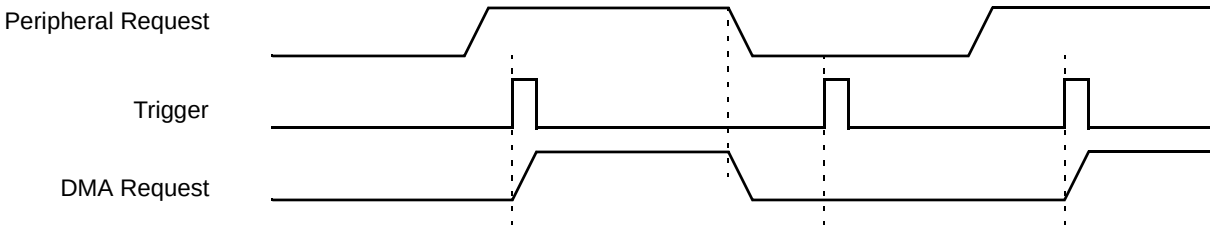


Figure 20-5. DMACHMUX channel triggering: ignored trigger

This triggering capability may be used with any peripheral that supports DMA transfers, and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. Once set up, the SPI requests DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5 µs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO Ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (i.e.-resolution, range of values, etc.) can be found in Chapter 42, Periodic Interrupt Timer (PIT).

20.6.2 DMA channels with no triggering capability

The other channels of the DMA Mux provide the normal routing functionality as described in Section 20.1.3, Modes of operation.

20.6.3 “Always enabled” DMA sources

In addition to the peripherals that can be used as DMA sources, there are 6 additional DMA sources that are “always enabled.” Unlike the peripheral DMA sources, where the peripheral controls the flow of data during DMA transfers, the “always enabled” sources provide no such “throttling” of the data transfers. These sources are most useful in the following cases:

- Doing DMA transfers to/from GPIO—Moving data from/to one or more GPIO pins, either un-throttled (i.e., as fast as possible) or periodically (using the DMA triggering capability).

- Doing DMA transfers from memory to memory—Moving data from memory to memory, typically as fast as possible, sometimes with software activation.
- Doing DMA transfers from memory to the external bus (or vice versa)—Similar to memory to memory transfers, this is typically done as quickly as possible.
- Any DMA transfer that requires software activation—Any DMA transfer that should be explicitly started by software.

In cases where software should initiate the start of a DMA transfer, an “always enabled” DMA source can be used to provide maximum flexibility. When activating a DMA channel via software, subsequent executions of the minor loop require a new “start” event be sent. This can either be a new software activation, or a transfer request from the DMACHMUX. The options for doing this are:

- Transfer all data in a single minor loop. By configuring the DMA to transfer all of the data in a single minor loop (i.e., major loop counter = 1), no re-activation of the channel is necessary. The disadvantage to this option is the reduced granularity in determining the load that the DMA transfer will incur on the system. For this option, the DMA channel should be disabled in the DMACHMUX.
- Use explicit software re-activation. In this option, the DMA is configured to transfer the data using both minor and major loops, but the processor is required to re-activate the channel (by writing to the DMA registers) *after every minor loop*. For this option, the DMA channel should be disabled in the DMACHMUX.
- Use an “always enabled” DMA source. In this option, the DMA is configured to transfer the data using both minor and major loops, and the DMACHMUX does the channel re-activation. For this option, the DMA channel should be enabled and pointing to an “always enabled” source. Note that the re-activation of the channel can be continuous (DMA triggering is disabled) or can use the DMA triggering capability. In this manner, it is possible to execute periodic transfers of packets of data from one source to another, without processor intervention.

20.7 Initialization/application information

20.7.1 Reset

The reset state of each individual bit is shown within the Register Description section (Section 20.3.1, Register descriptions). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

20.7.2 Enabling and configuring sources

Example 20-1. Enabling a source with periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first four DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.

4. Configure the corresponding timer.
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 20-2. Configuring source #5 Transmit for use with DMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Configure a timer for the desired trigger interval.
4. Write 0xC5 to CHCONFIG2 (Base Address + 0x02).

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFDC000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
/* This can be replicated for 32 channels ! */

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Example 20-3. Enabling a source without periodic triggering

1. Determine with which DMA channel the source will be associated. Note that only the first four DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.

4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

Example 20-4. Configuring source #5 transmit for use with DMA channel 2, with no periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFDC000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
/* This can be replicated for 32 channels ! */

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

20.7.2.1 Disabling a source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

Example 20-5. Switching the source of a DMA channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source.
2. Clear the ENBL and TRIG bits of the DMA channel.

3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 20-6. Switching DMA channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the transfers to peripheral slot 7. This example assumes channel 8 doesn't have triggering capability.
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08).
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). (In this example, setting the TRIG bit would have no effect, due to the assumption that channels 8 does not support the periodic triggering functionality).

The following code example illustrates steps #2 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFFDC000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
/* This can be replicated for 32 channels ! */

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

Chapter 21
Deserial Serial Peripheral Interface (DSPI)

21.1 Introduction

21.1.1 Overview

The DSPI module provides a synchronous serial bus for communication between the MPC5675K device and an external peripheral device. The DSPI supports device pin count reduction through serialization of internal signals transmitted over the SPI block.

The MPC5675K device has three DSPIs, each of which can be used as a master SPI with several slave selects.

Figure 21-1 is a block diagram of the DSPI module.

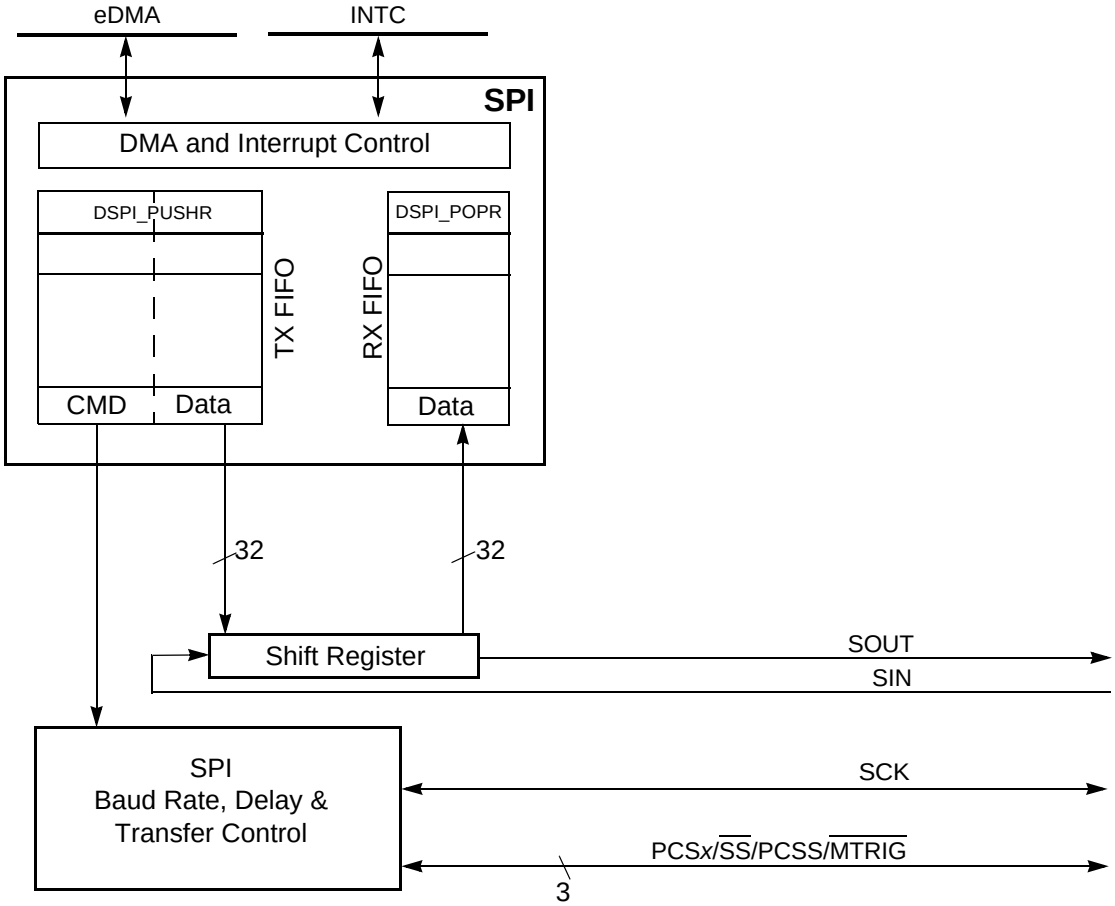


Figure 21-1. DSPI block diagram

21.1.2 Features

The DSPI provides these features:

- Full-duplex, synchronous transfers
- Master or slave operation
  - Data streaming operation in the slave mode with continuous slave selection
- Buffered transmit operation using the TX FIFO with 5 entries
- Buffered receive operation using the RX FIFO with 16 entries (5 entries on DSPI2)
- Programmable transfer attributes on a per-frame basis:
  - 4 transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Various programmable delays
  - Programmable serial frame size of 4 to 16 bits, expandable by software control
  - Programmable master bit rates
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- As many as 8 chip select lines available, depending on package and pin multiplexing
- 4 clock and transfer attributes registers (CTAR)
- Chip select strobe available as alternate function on one of the chip select pins for de-glitching
- FIFOs for buffering 5 transfers on the transmit side and 16 transfers on receive side (5 transfers on DSPI2)
- Queueing operation possible through use of the eDMA
- General-purpose I/O functionality on pins when not used for SPI
- In slave mode, continuos receive and transmit without de-asserting chip select

21.1.3 DSPI configurations

The DSPI module on this device operates only in the SPI configuration.

21.1.3.1 SPI configuration

The SPI Configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with internal FIFOs supporting external queues operation. Transmit data and received data reside in separate FIFOs. The host CPU or a DMA controller read the received data from the receive FIFO and write transmit data to the transmit FIFO.

For queued operations the SPI queues can reside in system RAM, external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished by a DMA controller or host CPU. Figure 21-2 shows a system example with DMA, DSPI and external queues in system RAM.

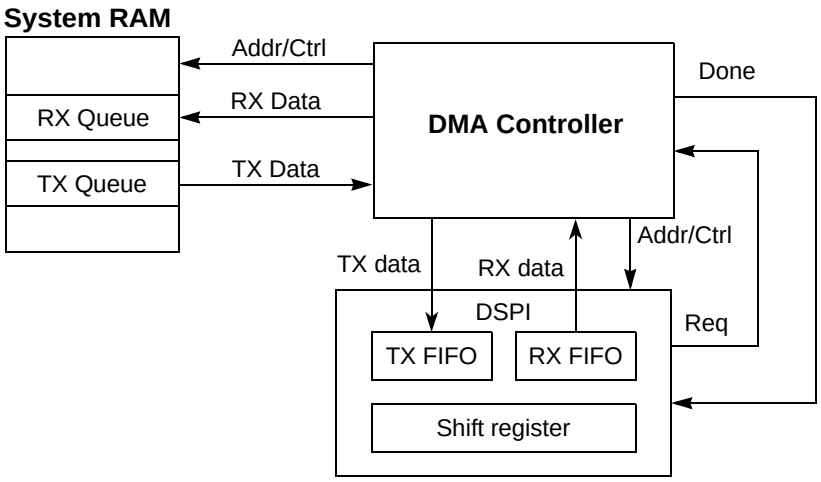


Figure 21-2. DSPI with Queues and eDMA

21.1.4 Modes of operation

The DSPI has five modes of operation that can be divided into two categories:

- Module-specific modes:
  - Master mode
  - Slave mode
  - Module disable mode
- Device-specific modes:
  - External stop mode
  - Debug mode

The DSPI enters module-specific modes when the host writes a DSPI register. The device-specific modes are controlled by signals external to the DSPI. The device-specific modes are modes that the entire device may enter in parallel to the DSPI block-specific modes.

21.1.4.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK signal and the PCS*x* signals are controlled by the DSPI and configured as outputs.

21.1.4.2 Slave mode

The slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The SCK signal and the PCS0/$\overline{SS}$ signal are configured as inputs and driven by an SPI bus master.

21.1.4.3 Module disable mode

The module disable mode can be used for power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in the module disable mode.

21.1.4.4 External stop mode

The external stop mode is used for device power management. The DSPI supports the Peripheral Bus stop mode mechanism. When a request is made to enter external stop mode, the DSPI block acknowledges the request and completes the transfer in progress. When the DSPI reaches the frame boundary it signals that the system clock to the DSPI module may be shut off.

21.1.4.5 Debug mode

The debug mode is used for system development and debugging. In this mode, you can read the last data from the FIFO without any changes. The DSPI_MCR[FRZ] bit controls DSPI behavior in the debug mode. If the FRZ bit is set, the DSPI stops all serial transfers when the device in the debug mode. If the FRZ bit is cleared, the device debug mode has no effect on the DSPI.

21.2 External signal description

21.2.1 Overview

Table 21-1 lists the DSPI signals on the device. Table 21-2 lists the chip select (CS) signals implemented on each DSPI interface on the device.

Table 21-1. DSPI signal properties

Name	I/O Type	Function	
		Master mode	Slave mode
PCS0/$\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
PCS1–PCS3	Output	Peripheral Chip Select 1–3	Unused
SIN	Input	Serial Data In	Serial Data In
SOUT	Output	Serial Data Out	Serial Data Out
SCK	Output / Input	Serial Clock (output)	Serial Clock (input)

Table 21-2. CS Lines implemented on device

DSPI	CS lines implemented							
	CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7
DSPI_0	X	X	X	X	X	X	X	X
DSPI_1	X	X	X	X	—	—	—	—
DSPI_2	X	X	X	X	—	—	—	—

21.2.2 Detailed signal description

21.2.2.1 PCS0/$\overline{SS}$ — Peripheral Chip Select/Slave Select

In master mode, the PCS0 signal is a Peripheral Chip Select output that selects which slave device the current transmission is intended for.

In slave mode, the active low $\overline{SS}$ signal is a Slave Select input signal that allows an SPI master to select the DSPI as the target for transmission.

21.2.2.2 PCS1–PCS3 — Peripheral Chip Selects 1–3

In master mode, PCS1–PCS3 are Peripheral Chip Select output signals.

In slave mode these signals are not used.

21.2.2.3 SIN — Serial Input

SIN is a serial data input signal.

21.2.2.4 SOUT — Serial Output

SOUT is a serial data output signal.

21.2.2.5 SCK — Serial Clock

SCK is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK is an input from an external bus master.

21.3 Memory map and register description

21.3.1 Memory map

Register accesses to memory addresses that are reserved or undefined result in a transfer error. Write accesses to the DSPI_POPR register also result in a transfer error.

Table 21-3 shows the base addresses for the DSPI modules. Addresses are the same for Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). Table 21-4 shows the memory map for the DSPI registers.

Table 21-3. DSPI module base addresses

Mode	Module	Module base address
LSM and DPM	DSPI_0	0xFFF9_0000
	DSPI_1	0xFFF9_4000
	DSPI_2	0xFFF9_8000

Table 21-4. DSPI memory map

Offset from DSPI_BASE	Register	Access[1]	Reset value[2]	Location
0x0000	DSPI Module Configuration Register (DSPI_MCR)	R/W	0x0000_0001	on page 563
0x0004	DSPI Hardware Configuration Register (DSPI_HCR)[3]	R	0xC34U_0000[4]	on page 565
0x0008	DSPI Transfer Count Register (DSPI_TCR)	R/W	0x0000_0000	on page 566
0x000C	DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0)	R/W	0x7800_0000	on page 566
0x0010	DSPI Clock and Transfer Attributes Register 1 (DSPI_CTAR1)	R/W	0x7800_0000	on page 566
0x0014	DSPI Clock and Transfer Attributes Register 2 (DSPI_CTAR2)	R/W	0x7800_0000	on page 566
0x0018	DSPI Clock and Transfer Attributes Register 3 (DSPI_CTAR3)	R/W	0x7800_0000	on page 566
0x001C–0x002B	Reserved			
0x002C	DSPI Status Register (DSPI_SR)	R/W	0x0200_0000	on page 571
0x0030	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)	R/W	0x0000_0000	on page 573
0x0034	DSPI Push TX FIFO Register (DSPI_PUSHR)	R/W	0x0000_0000	on page 574
0x0038	DSPI Pop RX FIFO Register (DSPI_POPR)	R/W	0x0000_0000	on page 576
0x003C	DSPI Transmit FIFO Register 0 (DSPI_TXFR0)	R	0x0000_0000	on page 576
0x0040	DSPI Transmit FIFO Register 1 (DSPI_TXFR1)	R	0x0000_0000	on page 576
0x0044	DSPI Transmit FIFO Register 2 (DSPI_TXFR2)	R	0x0000_0000	on page 576
0x0048	DSPI Transmit FIFO Register 3 (DSPI_TXFR3)	R	0x0000_0000	on page 576
0x004C	DSPI Transmit FIFO Register 4 (DSPI_TXFR4)	R	0x0000_0000	on page 576
0x0050–0x007B	Reserved			
0x007C	DSPI Receive FIFO Register 0 (DSPI_RXFR0)	R	0x0000_0000	on page 577
0x0080	DSPI Receive FIFO Register 1 (DSPI_RXFR1)	R	0x0000_0000	on page 577
0x0084	DSPI Receive FIFO Register 2 (DSPI_RXFR2)	R	0x0000_0000	on page 577
0x0088	DSPI Receive FIFO Register 3 (DSPI_RXFR3)	R	0x0000_0000	on page 577
0x008C	DSPI Receive FIFO Register 4 (DSPI_RXFR4)	R	0x0000_0000	on page 577
0x0090	DSPI Receive FIFO Register 5 (DSPI_RXFR5)[5]	R	0x0000_0000	on page 577
0x0094	DSPI Receive FIFO Register 6 (DSPI_RXFR6)[5]	R	0x0000_0000	on page 577
0x0098	DSPI Receive FIFO Register 7 (DSPI_RXFR7)[5]	R	0x0000_0000	on page 577
0x009C	DSPI Receive FIFO Register 8 (DSPI_RXFR8)[5]	R	0x0000_0000	on page 577
0x00A0	DSPI Receive FIFO Register 9 (DSPI_RXFR9)[5]	R	0x0000_0000	on page 577

Table 21-4. DSPI memory map (continued)

Offset from DSPI_BASE	Register	Access[1]	Reset value[2]	Location
0x00A4	DSPI Receive FIFO Register 10 (DSPI_RXFR10)[5]	R	0x0000_0000	on page 577
0x00A8	DSPI Receive FIFO Register 11 (DSPI_RXFR11)[5]	R	0x0000_0000	on page 577
0x00AC	DSPI Receive FIFO Register 12 (DSPI_RXFR12)[5]	R	0x0000_0000	on page 577
0x00B0	DSPI Receive FIFO Register 13 (DSPI_RXFR13)[5]	R	0x0000_0000	on page 577
0x00B4	DSPI Receive FIFO Register 14 (DSPI_RXFR14)[5]	R	0x0000_0000	on page 577
0x00B8	DSPI Receive FIFO Register 15 (DSPI_RXFR15)[5]	R	0x0000_0000	on page 577
0x00BC–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 The DSPI_HCR register provides parametrization information about the particular instance of the DSPI module.

4 One or more bits (indicated by “U”) are specific to the DSPI module. See register for more information.

5 This register is implemented on DSPI0 and DSPI1, but not on DSPI2.

21.3.2 Register descriptions

21.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI Module Configuration Register (DSPI_MCR) contains bits that configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR are allowed to be changed, while the DSPI is in the RUNNING state.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MDIS	DIS_TXF	DIS_RXF	0	0	SMPL_PT		0	0	0	0	0	0	PES	HALT
W					CLR_TXF	CLR_RXF										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 21-3. DSPI Module Configuration Register (DSPI_MCR)

Table 21-5. DSPI_MCR field descriptions

Field	Description
MSTR	Master/Slave Mode Select. The MSTR bit configures the DSPI for either master mode or slave mode. 0 DSPI is in slave mode. 1 DSPI is in master mode.
CONT_SCKE	Continuous SCK Enable. The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See Section 21.4.5, Continuous serial communications clock, for details. 0 Continuous SCK disabled. 1 Continuous SCK enabled.
DCONF	DSPI Configuration. The DCONF field selects between the three different configurations of the DSPI: 00 SPI. 01 Reserved. 10 Reserved. 11 Reserved.
FRZ	Freeze. The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the device enters Debug mode. 0 Do not stop serial transfers. 1 Stop serial transfers.
MTFE	Modified Timing Format Enable. The MTFE bit enables a modified transfer format to be used. See Section 21.4.4.4, Modified SPI transfer format (MTFE = 1, CPHA = 1), for more information. 0 Modified SPI transfer format disabled. 1 Modified SPI transfer format enabled.
PCSSE	Peripheral Chip Select Strobe Enable. The PCSSE bit enables the PCS5/$\overline{\text{PCSS}}$ to operate as a PCS Strobe output signal. See Section 21.4.3.5, Peripheral Chip Select Strobe Enable (PCSS), for more information. 0 PCS5/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select 5 signal. 1 PCS5/$\overline{\text{PCSS}}$ is used as an active-low PCS Strobe signal.
ROOE	Receive FIFO Overflow Overwrite Enable. The ROOE bit enables in RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer, generated the overflow, is ignored or shifted in to the shift register. See Section 21.4.7.6, Receive FIFO overflow interrupt request, for more information. 0 Incoming data is ignored. 1 Incoming data is shifted in to the shift register.
PCSIS*x*	Peripheral Chip Select Inactive State. The PCSIS bit determines the inactive state of the PCS*x* signal. 0 The inactive state of PCS*x* is low. 1 The inactive state of PCS*x* is high.
MDIS	Module Disable. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See Section 21.4.8, Power- saving features, for more information. The reset value of the MDIS bit is parameterized, with a default reset value of 0. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
DIS_TXF	Disable Transmit FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 21.4.2.3, FIFO disable operation, for details. 0 TX FIFO is enabled. 1 TX FIFO is disabled.

Table 21-5. DSPI_MCR field descriptions (continued)

Field	Description
DIS_RXF	Disable Receive FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 21.4.2.3, FIFO disable operation, for details. 0 RX FIFO is enabled. 1 RX FIFO is disabled.
CLR_TXF	Clear TX FIFO. CLR_TXF flushes the TX FIFO. Writing a 1 to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO Counter. 1 Clear the TX FIFO Counter.
CLR_RXF	Clear RX FIFO. CLR_RXF flushes the RX FIFO. Writing a 1 to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO Counter. 1 Clear the RX FIFO Counter.
SMPL_PT	Sample Point. SMPL_PT field controls when the DSPI master samples SIN in Modified Transfer Format. Figure 21-20 shows where the master can sample the SIN pin. 00 DSPI samples SIN at driving SCK edge. 01 DSPI samples SIN one system clock after driving SCK edge. 10 DSPI samples SIN two system clocks after driving SCK edge. 11 Reserved.
PES	Parity Error Stop. PES bit controls SPI operation when a parity error detected in received SPI frame. 0 SPI frames transmission continue. 1 SPI frames transmission stop.
HALT	Halt. The HALT bit starts and stops DSPI transfers. See Section 21.4.1, Start and stop of DSPI transfers, for details on the operation of this bit. 0 Start transfers. 1 Stop transfers.

21.3.2.2 DSPI Hardware Configuration Register (DSPI_HCR)

The DSPI Hardware Configuration Register provides particular implementation details about the DSPI module, such as the number of Receive and Transmit FIFO entries and the number of CTAR registers. It is read-only register.

Address: Base + 00004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PISR	0	0	0	CTAR			TXFR				RXFR			
W																
Reset[1]	1	1	0	0	0	0	1	1	0	1	0	0	—[2]	1	—[2]	—[2]

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-4. DSPI Hardware Configuration Register (DSPI_HCR)

1 Reset values are as follows: DSPI_0: 0xC34F_0000, DSPI_1: 0xC34F_0000, DSPI_2: 0xC344_0000.

2 RXFR[0:3] reset values are as follows: DSPI_0: 0xF, DSPI_1: 0xF, DSPI_2: 0x4.

Table 21-6. DSPI_HCR field descriptions

Field	Description
PISR	PISR, PISR0-3 and parallel inputs frame positions selection logic are implemented for the module. 0 DSPI_PISR0-3 registers are not implemented. 1 DSPI_PISR0-3 registers are implemented.
STAR	CTAR, Maximum implemented DSPI_CTAR register number.
TXFR	TXFR, Maximum implemented DSPI_TXFR register number.
RXFR	RXFR, Maximum implemented DSPI_RXFR register number.

21.3.2.3 DSPI Transfer Count Register (DSPI_TCR)

The DSPI Transfer Count Register (DSPI_TCR) contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write to the DSPI_TCR register when the DSPI is in the RUNNING state.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SPI_TCNT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-5. DSPI Transfer Count Register (DSPI_TCR)

Table 21-7. DSPI_TCR field descriptions

Field	Description
SPI_TCNT	SPI Transfer Counter. The SPI_TCNT field counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The Transfer Counter “wraps around” (i.e., incrementing the counter past 65535 resets the counter to zero).

21.3.2.4 DSPI Clock and Transfer Attributes Register *n* (DSPI_CTAR*n*)

The DSPI Clock and Transfer Attributes Registers (DSPI_CTAR*n*) registers are used to define different transfer attributes. Do not write to the DSPI_CTAR*n* registers while the DSPI is in the RUNNING state.

In master mode, the DSPI_CTAR0–DSPI_CTAR3 registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays.

In slave mode, a subset of the bitfields in the DSPI_CTAR0 register is used to set the slave transfer attributes. The other DSPI_CTAR*n* registers (DSPI_CTAR1–DSPI_CTAR3) are not available in slave mode.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which DSPI_CTAR register is used. When the DSPI is configured as an SPI bus slave, the DSPI_CTAR0 register is used.

Address: Base + 0x000C (DSPI_CTAR0) Base + 0x0014 (DSPI_CTAR2)
Base + 0x0010 (DSPI_CTAR1) Base + 0x0018 (DSPI_CTAR3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-6. DSPI Clock and Transfer Attributes Register *n* (DSPI_CTAR*n*) in master mode

Address: Base + 0x000C (DSPI_CTAR0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FMSZ				CPOL	CPHA	PE	PP	0	0	0	0	0	0	0	
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-7. DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) in slave mode

Table 21-8. DSPI_CTAR*n* field descriptions in master mode

Field	Descriptions
DBR	Double Baud Rate. The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in master mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in Table 21-9. See the BR field description for details on how to calculate the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set. 0 The baud rate is computed normally with a 50/50 duty cycle 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler
FMSZ	Frame Size. The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.

Table 21-8. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
CPOL	Clock Polarity. The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low. 1 The inactive state value of SCK is high.
CPHA	Clock Phase. The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. In Continuous SCK mode the bit value is ignored and the transfers are done as CPHA bit is set to 1. 0 Data is captured on the leading edge of SCK and changed on the following edge. 1 Data is changed on the leading edge of SCK and captured on the following edge.
LSBFE	LSB First. The LSBFE bit selects if the LSB or MSB of the frame is transferred first. 0 Data is transferred MSB first. 1 Data is transferred LSB first.
PCSSCK	PCS to SCK Delay Prescaler. The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. See the CSSCK field description how to compute the PCS to SCK Delay. 00 PCS to SCK Prescaler value is 1. 01 PCS to SCK Prescaler value is 3. 10 PCS to SCK Prescaler value is 5. 11 PCS to SCK Prescaler value is 7.
PASC	After SCK Delay Prescaler. The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. See the ASC field description how to compute the After SCK Delay. 00 After SCK Delay Prescaler value is 1. 01 After SCK Delay Prescaler value is 3. 10 After SCK Delay Prescaler value is 5. 11 After SCK Delay Prescaler value is 7.
PDT	Delay after Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in master mode. See the DT field description for details on how to compute the Delay after Transfer. 00 Delay after Transfer Prescaler value is 1. 01 Delay after Transfer Prescaler value is 3. 10 Delay after Transfer Prescaler value is 5. 11 Delay after Transfer Prescaler value is 7.
PBR	Baud Rate Prescaler. The PBR field selects the prescaler value for the baud rate. This field is only used in master mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. See the BR field description for details on how to compute the baud rate. 00 Baud Rate Prescaler value is 2. 01 Baud Rate Prescaler value is 3. 10 Baud Rate Prescaler value is 5. 11 Baud Rate Prescaler value is 7.

Table 21-8. DSPI_CTAR*n* field descriptions in master mode (continued)

Field	Descriptions
CSSCK	PCS to SCK Delay Scaler. The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK Delay (t_{CSC}) is the delay between the assertion of PCS and the first edge of the SCK. Table 21-10 list the scaler values.The PCS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation: $$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$ ***Eqn. 21-1*** See Section 21.4.3.2, PCS to SCK delay (tCSC), for more details.
ASC	After SCK Delay Scaler. The ASC field selects the scaler value for the After SCK Delay. This field is only used in master mode. The After SCK Delay (t_{ASC}) is the delay between the last edge of SCK and the negation of PCS. Table 21-10 list the scaler values.The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation: $$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$ ***Eqn. 21-2*** See Section 21.4.3.3, After SCK delay (tASC), for more details.
DT	Delay after Transfer Scaler. The DT field selects the Delay after Transfer Scaler. This field is only used in master mode. The Delay after Transfer (t_{DT}) is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. Table 21-10 lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one SCK clock period, The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation: $$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$ ***Eqn. 21-3*** See Section 21.4.3.4, Delay after transfer (tDT), for more details.
BR	Baud Rate Scaler. The BR field selects the scaler value for the baud rate. This field is only used in master mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. Table 21-11 lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: $$SCK\ baud\ rate = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$$ ***Eqn. 21-4*** See Section 21.4.3.1, Baud rate generator, for more details.

Table 21-9. DSPI SCK duty cycle

DBR	CPHA	PBR	SCK Duty Cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33

Table 21-9. DSPI SCK duty cycle (continued)

DBR	CPHA	PBR	SCK Duty Cycle
1	1	10	60/40
1	1	11	57/43

Table 21-10. Delay scaler encoding

Field value	Scaler Value	Field value	Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 21-11. DSPI baud rate scaler

BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

Table 21-12. DSPI_CTAR*0*, DSPI_CTAR*1* field descriptions in slave mode

Field	Descriptions
FMSZ[0:4]	Frame Size. The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.
CPOL	Clock Polarity. The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). 0 The inactive state value of SCK is low. 1 The inactive state value of SCK is high.

Table 21-12. DSPI_CTAR*0*, DSPI_CTAR*1* field descriptions in slave mode (continued)

Field	Descriptions
CPHA	Clock Phase. The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. 0 Data is captured on the leading edge of SCK and changed on the following edge. 1 Data is changed on the leading edge of SCK and captured on the following edge.
PE	Parity Enable. PE bit enables parity bit transmission and reception for the frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
29–31	Not used, always write to 0 to keep software compatible with future updates.

21.3.2.5 DSPI Status Register (DSPI_SR)

The DSPI Status Register (DSPI_SR) contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software clears flag bits in the DSPI_SR register by writing a ‘1’ to it. Writing a ‘0’ to a flag bit has no effect. This register may not be writable in module disable mode due to the use of power saving mechanisms.

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	SPEF	0	RFOF	0	RFDF	0
W	w1c	w1c		w1c	w1c		w1c				w1c		w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-8. DSPI Status Register (DSPI_SR)

Table 21-13. DSPI_SR field descriptions

Field	Description
TCF	Transfer Complete Flag. The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit remains set until cleared by writing 1 to it. 0 Transfer not complete. 1 Transfer complete.

Table 21-13. DSPI_SR field descriptions (continued)

Field	Description
TXRXS	TX and RX Status. The TXRXS bit reflects the run status of the DSPI. See Section 21.4.1, Start and stop of DSPI transfers, what causes this bit to be set or cleared. 0 TX and RX operations are disabled (DSPI is in STOPPED state). 1 TX and RX operations are enabled (DSPI is in RUNNING state).
EOQF	End of Queue Flag. The EOQF bit indicates that the last entry in a queue has been transmitted when the DSPI in the master mode. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword (DSPI Push Tx FIFO) and the end of the transfer is reached. The EOQF bit remains set until cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executed command. 1 EOQ bit is set in the executed SPI command.
TFUF	Transmit FIFO Underflow Flag. The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by writing 1 to it. 0 TX FIFO underflow has not occurred. 1 TX FIFO underflow has occurred.
TFFF	Transmit FIFO Fill Flag. The TFFF bit provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller to the TX FIFO full request. 0 TX FIFO is full. 1 TX FIFO is not full.
SPEF	SPI Parity Error Flag. The SPEF flag indicates that an SPI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred. 1 Parity Error has occurred.
RFOF	Receive FIFO Overflow Flag. The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by writing 1 to it. 0 RX FIFO overflow has not occurred. 1 RX FIFO overflow has occurred.
RFDF	Receive FIFO Drain Flag. The RFDF bit provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller when the RX FIFO is empty. 0 RX FIFO is empty. 1 RX FIFO is not empty.
TXCTR	TX FIFO Counter. The TXCTR field indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
TXNXTPTR	Transmit Next Pointer. The TXNXTPTR field indicates which TX FIFO Entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 21.4.7.4, Transmit FIFO underflow interrupt request, for more details.
RXCTR	RX FIFO Counter. The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.

Table 21-13. DSPI_SR field descriptions (continued)

Field	Description
POPNXTPTR	Pop Next Pointer. The POPNXTPTR field contains a pointer to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 21.4.2.5, Receive First In First Out (RX FIFO) buffering mechanism, for more details.

21.3.2.6 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER) controls DMA and interrupt requests. Do not write to the DSPI_RSER while the DSPI is in the RUNNING state.

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	SPEF_RE	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-9. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 21-14. DSPI_RSER field descriptions

Field	Description
TCF_RE	Transmission Complete Request Enable. The TCF_RE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled. 1 TCF interrupt requests are enabled.
EOQF_RE	DSPI Finished Request Enable. The EOQF_RE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled. 1 EOQF interrupt requests are enabled.
TFUF_RE	Transmit FIFO Underflow Request Enable. The TFUF_RE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled. 1 TFUF interrupt requests are enabled.
TFFF_RE	Transmit FIFO Fill Request Enable. The TFFF_RE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled. 1 TFFF interrupt requests or DMA requests are enabled.

Table 21-14. DSPI_RSER field descriptions (continued)

Field	Description
TFFF_DIRS	Transmit FIFO Fill DMA or Interrupt Request Select. The TFFF_DIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFF_RE bit in the DSPI_RSER register is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is generated. 1 DMA request is generated.
SPEF_RE	SPI Parity Error Request Enable. The SPEF_RE bits enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled. 1 PEF interrupt requests are enabled.
RFOF_RE	Receive FIFO Overflow Request Enable. The RFOF_RE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled. 1 RFOF interrupt requests are enabled.
RFDF_RE	Receive FIFO Drain Request Enable. The RFDF_RE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled. 1 RFDF interrupt requests or DMA requests are enabled.
RFDF_DIRS	Receive FIFO Drain DMA or Interrupt Request Select. The RFDF_DIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDF_RE bit in the DSPI_RSER register is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is generated. 1 DMA request is generated.

21.3.2.7 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI PUSH TX FIFO Register (DSPI_PUSHR) allows writing to the TX FIFO. Data written to this register is transferred to the TX FIFO. See Section 21.4.2.4, Transmit First In First Out (TX FIFO) buffering mechanism, for more information. Eight or sixteen bit write accesses to the DSPI_PUSHR transfers all 32 register bits to the TX FIFO.The register structure is different in master and slave modes. In master mode the register provides 16-bit command and 16-bit data to the TX FIFO. In slave mode. all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CTCNT	PE	PP	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-10. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode

Table 21-15. DSPI_PUSHR field descriptions in master mode

Field	Descriptions
CONT	Continuous Peripheral Chip Select Enable. The CONT bit selects a Continuous Selection Format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 21.4.4.5, Continuous selection format, for more information. 0 Return Peripheral Chip Select signals to their inactive state between transfers. 1 Keep Peripheral Chip Select signals asserted between transfers.
CTAS	Clock and Transfer Attributes Select. The CTAS field selects number of the DSPI_CTAR register be used to set the transfer attributes for the associated SPI frame. The field is only used in SPI master mode. In SPI slave mode DSPI_CTAR0 is used. The number of DSPI_CTAR registers is implementation specific and the CTAS should be set to select only implemented one.
EOQ	End Of Queue. The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set. 0 The SPI data is not the last data to transfer. 1 The SPI data is the last data to transfer.
CTCNT	Clear Transfer Counter. The CTCNT bit clears the SPI_TCNT field in the DSPI_TCR register. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear SPI_TCNT field in the DSPI_TCR. 1 Clear SPI_TCNT field in the DSPI_TCR.
PE	Parity Enable. PE bit enables parity bit transmission and parity reception check for the SPI frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
PP	Parity Polarity. PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
PCS*x*	Peripheral Chip Select 0–7. The PCS bits select which PCS signals are asserted for the transfer. 0 Negate the PCS*x* signal. 1 Assert the PCS*x* signal.
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TXDATA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-11. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Table 21-16. DSPI_PUSHR field descriptions in slave mode

Field	Descriptions
TXDATA	Transmit Data. The TXDATA field holds SPI data to be transferred.

21.3.2.8 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI POP RX FIFO Register (DSPI_POPR) provides means to read the RX FIFO. See Section 21.4.2.5, Receive First In First Out (RX FIFO) buffering mechanism, for a description of the RX FIFO operations. 8- or 16-bit read accesses to the DSPI_POPR have the same effect on the RX FIFO as 32-bit read access.

Address: Base + 0x0038 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXDATA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-12. DSPI POP RX FIFO Register (DSPI_POPR)

Table 21-17. DSPI_POPR field descriptions

Field	Description
RXDATA	Received Data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

21.3.2.9 DSPI Transmit FIFO Registers (DSPI_TXFR*n*)

The DSPI Transmit FIFO Registers (DSPI_TXFR*n*) provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFR*n* registers does not alter the state of the TX FIFO.

Address: Base + 0x003C (DSPI_TXFR0) Base + 0x0048 (DSPI_TXFR3)
Base + 0x0040 (DSPI_TXFR1) Base + 0x004C (DSPI_TXFR4)
Base + 0x0044 (DSPI_TXFR2)

Access: User read/-only

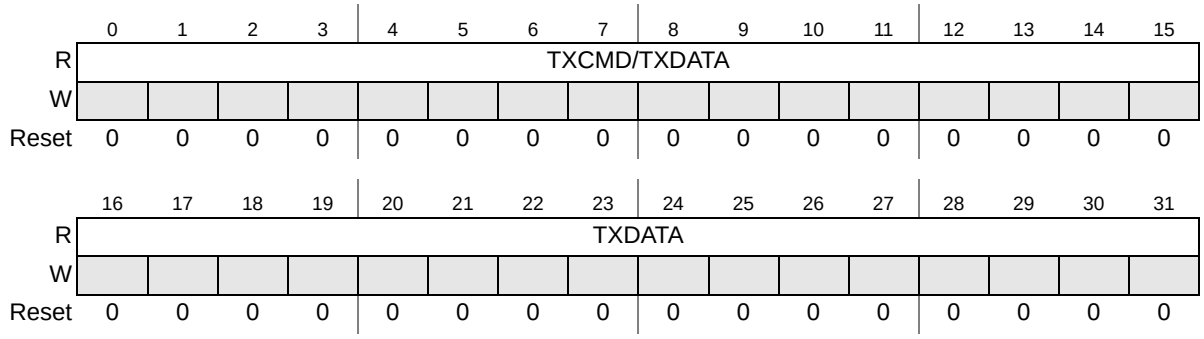


Figure 21-13. DSPI Transmit FIFO Register *n* (DSPI_TXFR*n*)

Table 21-18. DSPI_TXFR*n* field descriptions

Field	Description
TXCMD[0:15]/ TXDATA[0:15]	Transmit Command or Transmit Data. In master mode the TXCMD field contains the command that sets the transfer attributes for the SPI data. See Section 21.3.2.7, DSPI PUSH TX FIFO Register (DSPI_PUSHR), for details on the command field. In slave mode the TXDATA contains 16 MSB bits of the SPI data to be shifted out.
TXDATA[16:31]	Transmit Data. The TXDATA field contains the SPI data to be shifted out.

21.3.2.10 DSPI Receive FIFO Registers (DSPI_RXFR*n*)

The DSPI Receive FIFO Registers (DSPI_RXFR*n*) provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR*n* registers are read-only. Reading the DSPI_RXFR*n* registers does not alter the state of the RX FIFO.

Address: See Table 21-20

Access: User read-only

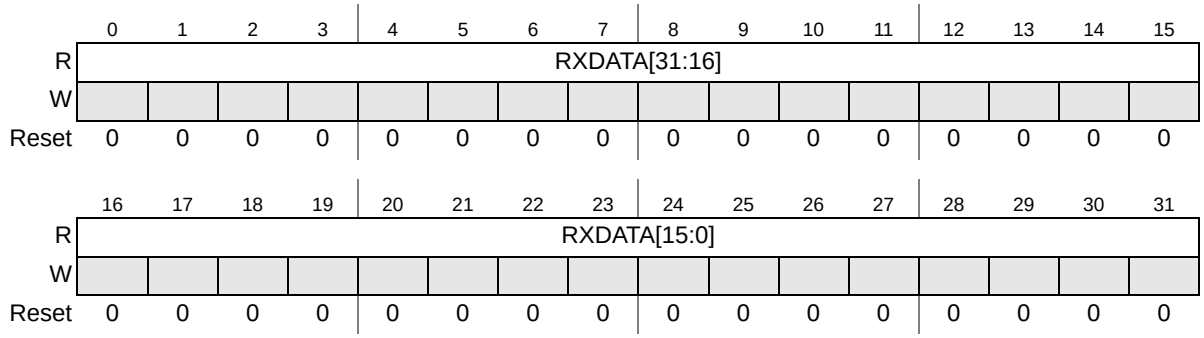


Figure 21-14. DSPI Receive FIFO Register *n* (DSPI_RXFR*n*)

Table 21-19. DSPI_RXFR*n* field descriptions

Field	Description
RXDATA	Receive Data. The RXDATA field contains the received SPI data.

Table 21-20. DSPI_RXFR*n* addresses

Address Offset	Register	DSPI_0, DSPI_1	DSPI_2
0x007C	DSPI_RXFR0	,	,
0x0080	DSPI_RXFR1	,	,
0x0084	DSPI_RXFR2	,	,
0x0088	DSPI_RXFR3	,	,
0x008C	DSPI_RXFR4	,	,
0x0090	DSPI_RXFR5	,	not implemented
0x0094	DSPI_RXFR6	,	not implemented
0x0098	DSPI_RXFR7	,	not implemented
0x009C	DSPI_RXFR8	,	not implemented
0x00A0	DSPI_RXFR9	,	not implemented
0x00A4	DSPI_RXFR10	,	not implemented
0x00A8	DSPI_RXFR11	,	not implemented
0x00AC	DSPI_RXFR12	,	not implemented
0x00B0	DSPI_RXFR13	,	not implemented
0x00B4	DSPI_RXFR14	,	not implemented
0x00B8	DSPI_RXFR15	,	not implemented

21.4 Functional description

The DSPI block supports full-duplex, synchronous serial communications between MCUs and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has one configuration:

- SPI Configuration, in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPI Module Configuration Register (DSPI_MCR) determines the DSPI Configuration. See Table 21-5 for the DSPI configuration values.

The DSPI_CTAR0–DSPI_CTAR3 registers hold clock and transfer attributes. The SPI configuration allows selecting which CTAR to use on a frame-by frame basis by setting a field in the SPI command. See Section 21.3.2.4, DSPI Clock and Transfer Attributes Register n (DSPI_CTARn), for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in the Figure 21-15. When a data transfer operation is performed, data is serially shifted a predetermined number of bit positions. Because the modules are linked, data is exchanged between the master and the slave. The data that was in the master shift register

is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate a completed transfer.

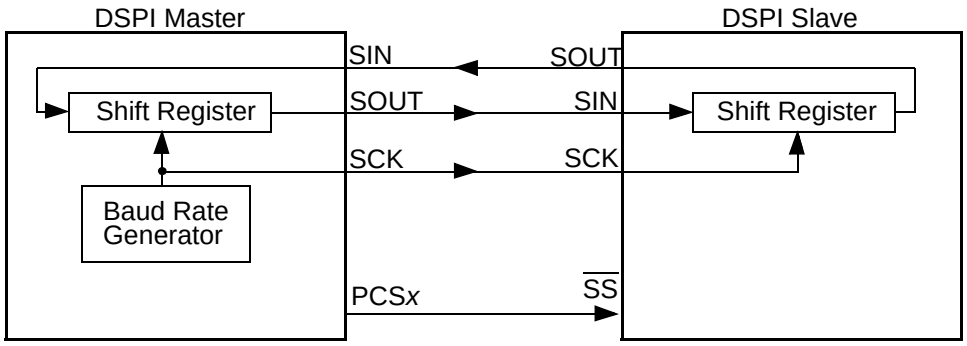


Figure 21-15. SPI serial protocol overview

Generally more than one slave device can be connected to the DSPI master. Eight Peripheral Chip Select (PCS) signals of the DSPI masters can be used to select which of the slaves with which to communicate.

The three DSPI configurations share transfer protocol and timing properties that are described independently of the configuration in Section 21.4.4, Transfer formats. The transfer rate and delay settings are described in Section 21.4.3, DSPI baud rate and clock delay generation.

21.4.1 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. Serial transfers take place in the RUNNING state.

The TXRXS bit in the DSPI_SR indicates in what state the DSPI is. If the module in RUNNING state, this bit is set.

The DSPI is started (DSPI transitions to RUNNING) when all of the following conditions are true:

- DSPI_SR[EOQF] bit is clear
- The device is not in the debug mode is or the DSPI_MCR[FRZ] bit is clear
- DSPI_MCR[HALT] bit is clear

The DSPI stops (transitions from RUNNING to STOPPED) after the current frame when any one of the following conditions exist:

- DSPI_SR[EOQF] bit is set
- The device is in the debug mode and the DSPI_MCR[FRZ] bit is set
- DSPI_MCR[HALT] bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or immediately if no transfers are in progress.

21.4.2 Serial Peripheral Interface (SPI) configuration

The SPI Configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI Configuration when the DCONF field in the DSPI_MCR is 0b00. The SPI frames can be from four to sixteen bits long. Host CPU or a DMA controller transfer the SPI data from the external to DSPI RAM queues to a transmit First-In First-Out (TX FIFO) buffer. The received data is stored in entries in the Receive FIFO (RX FIFO) buffer. Host CPU or the DMA controller transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffers operation is described in Section 21.4.2.4, Transmit First In First Out (TX FIFO) buffering mechanism, and Section 21.4.2.5, Receive First In First Out (RX FIFO) buffering mechanism. The interrupt and DMA request conditions are described in Section 21.4.7, Interrupts/DMA requests.

The SPI Configuration supports two block-specific modes—master mode and slave mode. The FIFO operations are similar for both modes. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field space is used for the 16 most significant bits of the transmit data.

21.4.2.1 SPI master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (PCS) signals. The SPI command field in the executing TX FIFO entry determines which CTAR registers are used to set the transfer attributes and which PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 21.3.2.7, DSPI PUSH TX FIFO Register (DSPI_PUSHR), for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

21.4.2.2 SPI slave mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0.

21.4.2.3 FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO Disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR.When the TX FIFO is disabled, the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. Likewise, when the RX

FIFO is disabled, the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO, but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

21.4.2.4 Transmit First In First Out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds 1–5 words, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI PUSH TX FIFO Register (DSPI_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO Counter field (TXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

The TXNXTPTR field indicates which TX FIFO entry is to be transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register. The maximum value of the field is equal to DSPI_HCR[TXFR] and it rolls over after reaching the maximum.

21.4.2.4.1 Filling the TX FIFO

Host software or other intelligent blocks can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO Fill Flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the DMA controller indicates that a write to DSPI_PUSHR is complete. Writing a ‘1’ to the TFFF bit also clears it. The TFFF can generate a DMA request or an interrupt request. See Section 21.4.7.2, Transmit FIFO fill interrupt or DMA request, for details.

The DSPI ignores attempts to push data to a full TX FIFO, the state of the TX FIFO does not change and no error condition is indicated.

21.4.2.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO Counter decrements by one. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a ‘1’ to the CLR_TXF bit in DSPI_MCR.

If an external bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the Transmit FIFO Underflow Flag (TFUF) in the slave’s DSPI_SR is set. See Section 21.4.7.4, Transmit FIFO underflow interrupt request, for details.

21.4.2.5 Receive First In First Out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds from one to sixteen received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data are removed (popped) from

the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO.

The RX FIFO Counter field (RXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI_SR points to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that is returned when DSPI_POPR is read. The POPNXTPTR field is incremented every time the DSPI_POPR is read. The maximum value of the field is equal to DSPI_HCR[RXFR] and it rolls over after reaching the maximum.

21.4.2.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO Counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

21.4.2.5.2 Draining the RX FIFO

Host CPU or a DMA can remove (pop) entries from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). A read of the DSPI_POPR decrements the RX FIFO Counter by one. Attempts to pop data from an empty RX FIFO are ignored and the RX FIFO Counter remains unchanged. The data, read from the empty RX FIFO, is undetermined.

When the RX FIFO is not empty, the RX FIFO Drain Flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by writing a ‘1’ to it.

21.4.3 DSPI baud rate and clock delay generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. Figure 21-16 shows conceptually how the SCK signal is generated.

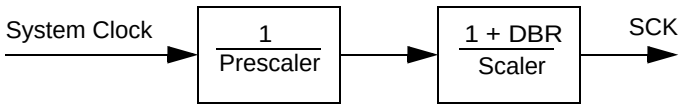


Figure 21-16. Communications clock prescalers and scalers

21.4.3.1 Baud rate generator

The baud rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibility of halving the scaler division. The DBR, PBR, and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR field description. Table 21-21 shows an example of how to compute the baud rate.

Table 21-21. Baud rate computation example

f_{sys}	PBR	Prescaler	BR	Scaler	DBR	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mb/s
20 MHz	0b00	2	0b0000	2	1	10 Mb/s

21.4.3.2 PCS to SCK delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See Figure 21-18 for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*n* registers select the PCS to SCK delay by the formula in the CSSCK field description (see Section 21.3.2.4, DSPI Clock and Transfer Attributes Register n (DSPI_CTARn)). Table 21-22 shows an example of how to compute the PCS to SCK delay.

Table 21-22. PCS to SCK delay computation example

f_{sys}	PCSSCK	Prescaler	CSSCK	Scaler	PCS to SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

21.4.3.3 After SCK delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See Figure 21-18 and Figure 21-19 for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*n* registers select the After SCK Delay by the formula in the ASC field description (see Section 21.3.2.4, DSPI Clock and Transfer Attributes Register n (DSPI_CTARn)). Table 21-23 shows an example of how to compute the After SCK delay.

Table 21-23. After SCK delay computation example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

21.4.3.4 Delay after transfer (t_{DT})

The Delay after Transfer is the minimum time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See Figure 21-18 for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*n* registers select the Delay after Transfer by the formula in the DT field description (see Section 21.3.2.4, DSPI Clock and Transfer Attributes Register n (DSPI_CTARn)). Table 21-24 shows an example of how to compute the Delay after Transfer.

Table 21-24. Delay after transfer computation example

f_{sys}	PDT	Prescaler	DT	Scaler	Delay after transfer
100 MHz	0b01	3	0b1110	32768	0.98 ms

When in non-continuous clock mode the t_{DT} delay is configured according Equation 21-3. When in continuous clock mode the delay is fixed at 1 SCK period.

21.4.3.5 Peripheral Chip Select Strobe Enable ($\overline{\text{PCSS}}$)

The $\overline{\text{PCSS}}$ signal provides a delay to allow the PCS signals to settle after a transition occurs thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI_MCR, $\overline{\text{PCSS}}$ provides a signal for an external demultiplexer to decode the PCS0–PCS4 and PCS6–PCS7 signals into as many as 128 glitch-free PCS signals. Figure 21-17 shows the timing of the $\overline{\text{PCSS}}$ signal relative to PCS signals.

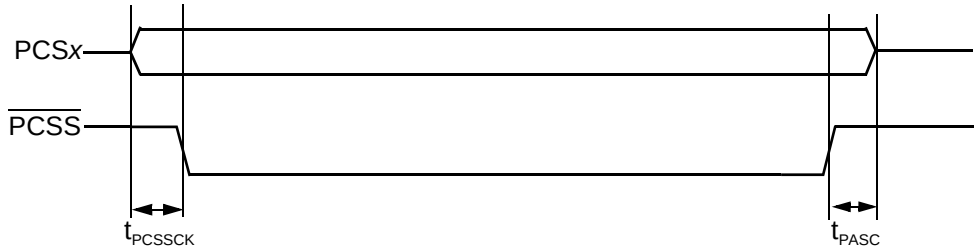


Figure 21-17. $\overline{\text{PCSS}}$ timing

The delay between the assertion of the PCS signals and the assertion of $\overline{\text{PCSS}}$ is selected by the PCSSCK field in the DSPI_CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK \qquad \textit{Eqn. 21-5}$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS negation is selected by the PASC field in the DSPI_CTAR based on the following formula:

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC \qquad \textit{Eqn. 21-6}$$

Table 21-25 shows an example of how to compute the t_{pcssck} delay.

Table 21-25. $\overline{\text{PCSS}}$ assert computation example

f_{sys}	PCSSCK	Prescaler	Delay before transfer
100 MHz	0b11	7	70.0 ns

Table 21-26 shows an example of how to compute the t_{pasc} delay.

Table 21-26. $\overline{\text{PCSS}}$ negate computation example

f_{sys}	PASC	Prescaler	Delay after transfer
100 MHz	0b11	7	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK is enabled.

21.4.4 Transfer formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the PCS signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI Clock and Transfer Attributes Registers (DSPI_CTAR*n*) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_CTAR0 select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master and the slave devices to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA=0
- Classic SPI with CPHA=1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format.

The DSPI provides the option of keeping the PCS signals asserted between frames. See Section 21.4.4.5, Continuous selection format, for details.

21.4.4.1 Classic SPI transfer format (CPHA = 0)

The transfer format shown in Figure 21-18 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.

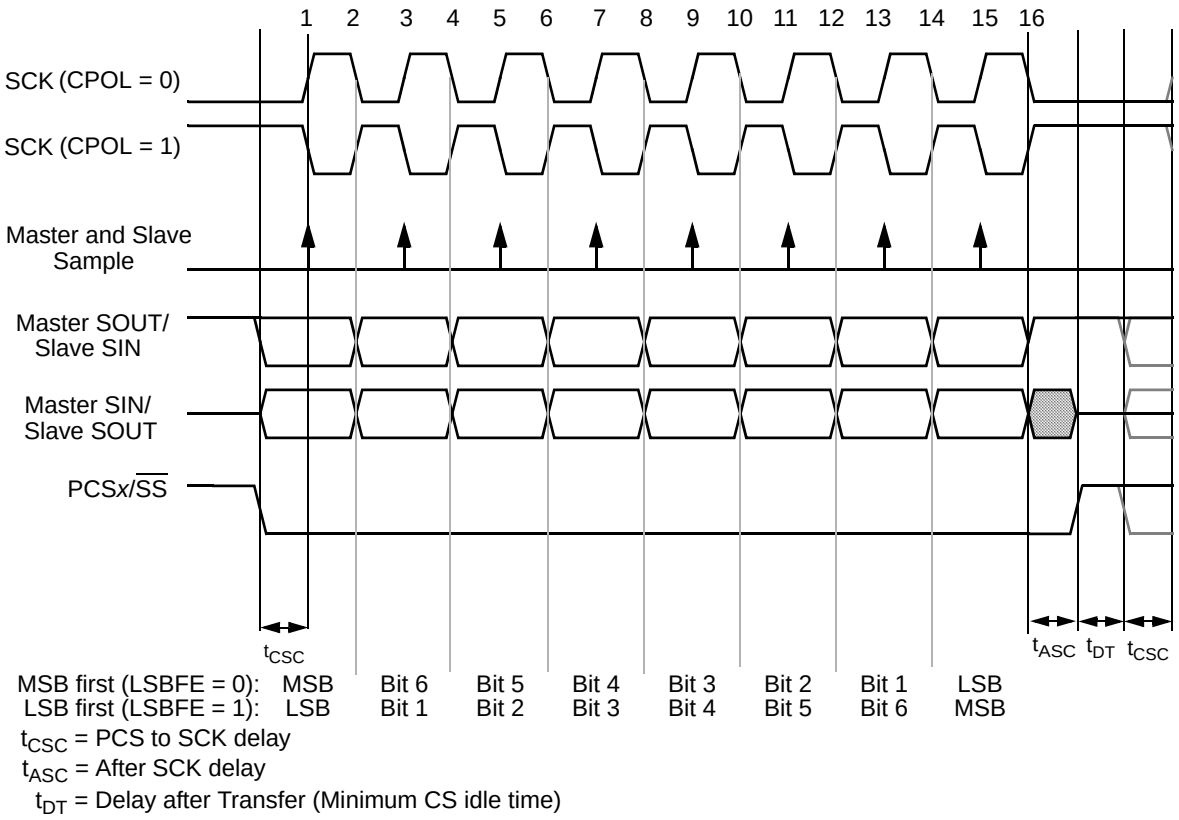


Figure 21-18. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the SCK the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN pins on the odd-numbered clock edges and changes the data on their SOUT pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

21.4.4.2 Classic SPI transfer format (CPHA = 1)

This transfer format shown in Figure 21-19 is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges

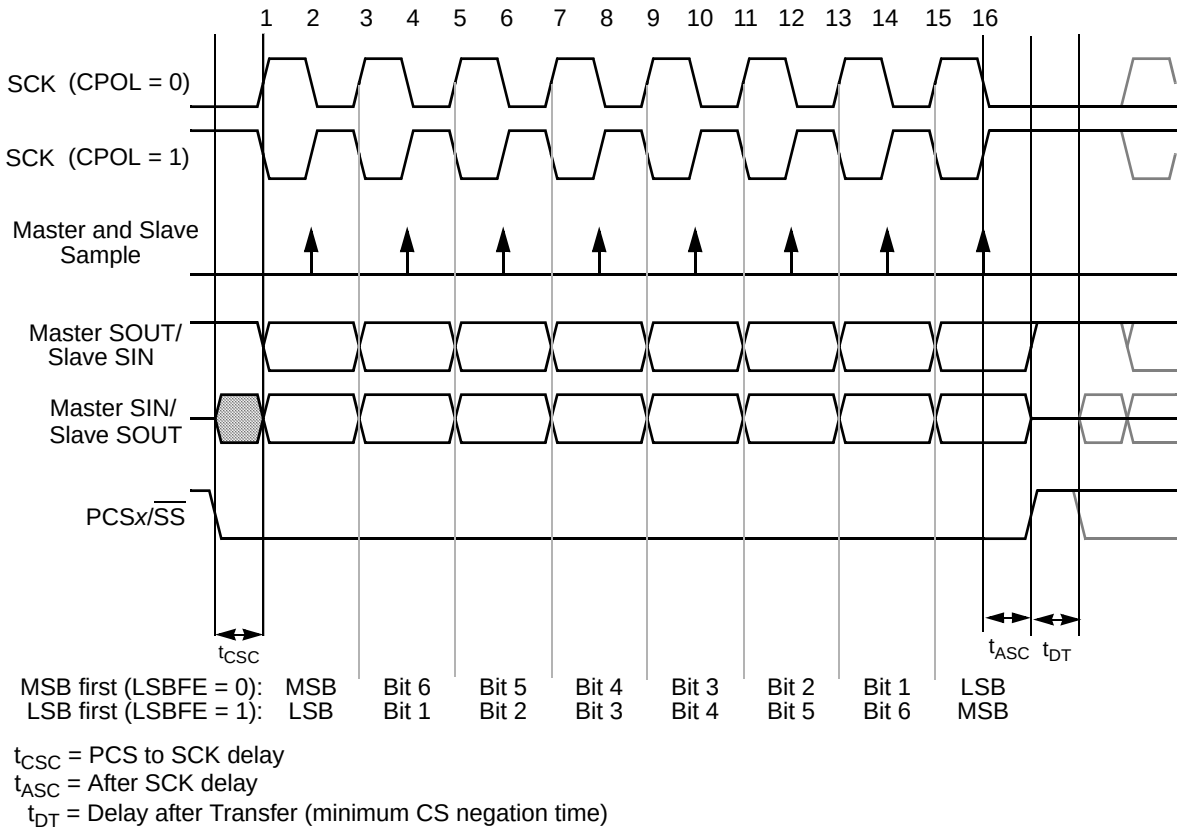


Figure 21-19. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

21.4.4.3 Modified SPI transfer format (MTFE = 1, CPHA = 0)

In this modified transfer format, both the master and the slave sample later in the SCK period than in classic SPI mode to allow tolerate more delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

The master and the slave place data on the SOUT pins at the assertion of the PCS signal. After the PCS to SCK delay has elapsed the first SCK edge is generated. The slave samples the master SOUT signal on every odd-numbered SCK edge. The DSPI in the slave mode when the MTFE bit is set also places new data on the slave SOUT on every odd numbered clock edge. Regular external slave, configured with CPHA = 0 format drives its SOUT output at every even numbered SCK clock edge.

The DSPI master places its second data bit on the SOUT line one system clock after odd-numbered SCK edge if the system frequency to SCK frequency ratio is higher than three. If this ratio is below four the master changes SOUT at odd numbered SCK edge. The point where the master samples the SIN is selected by the DSPI_MCR[SMPL_PT] field. The Table 21-5 lists the number of system clock cycles between the active edge of SCK and the master Sample point. The master sample point can be delayed by one or two system clock cycles. The SMPL_PT field should be set to 0 if the system to SCK frequency ratio is less than 4.

The following timing diagrams illustrate the DSPI operation with MTFE = 1. Timing delays shown are:

- T_{csc}—PCS to SCK assertion delay
- T_{acs}—After SCK PCS negation delay
- T_{su_ms}—Master SIN setup time
- T_{hd_ms}—Master SIN hold time
- T_{vd_sl}—Slave data output valid time, time between slave data output SCK driving edge and data becomes valid
- T_{su_sl}—Data setup time on slave data input
- T_{hd_sl}—Data hold time on slave data input
- T_{sys}—System clock period

Figure 21-20 shows the modified transfer format for CPHA = 0 and Fsys/Fsck = 4. Only the condition where CPOL = 0 is illustrated. Solid triangles show the data sampling clock edges. The two possible slave behavior are shown.

- The signal marked “SOUT of Ext Slave” represents the regular SPI slave serial output.
- The signal marked “SOUT of DSPI Slave” represents the DSPI in the slave mode with the MTFE bit set.

Other MTFE = 1 diagrams show DSPI SIN input as being driven by a regular external SPI slave, configured according DSPI master CPHA programming.

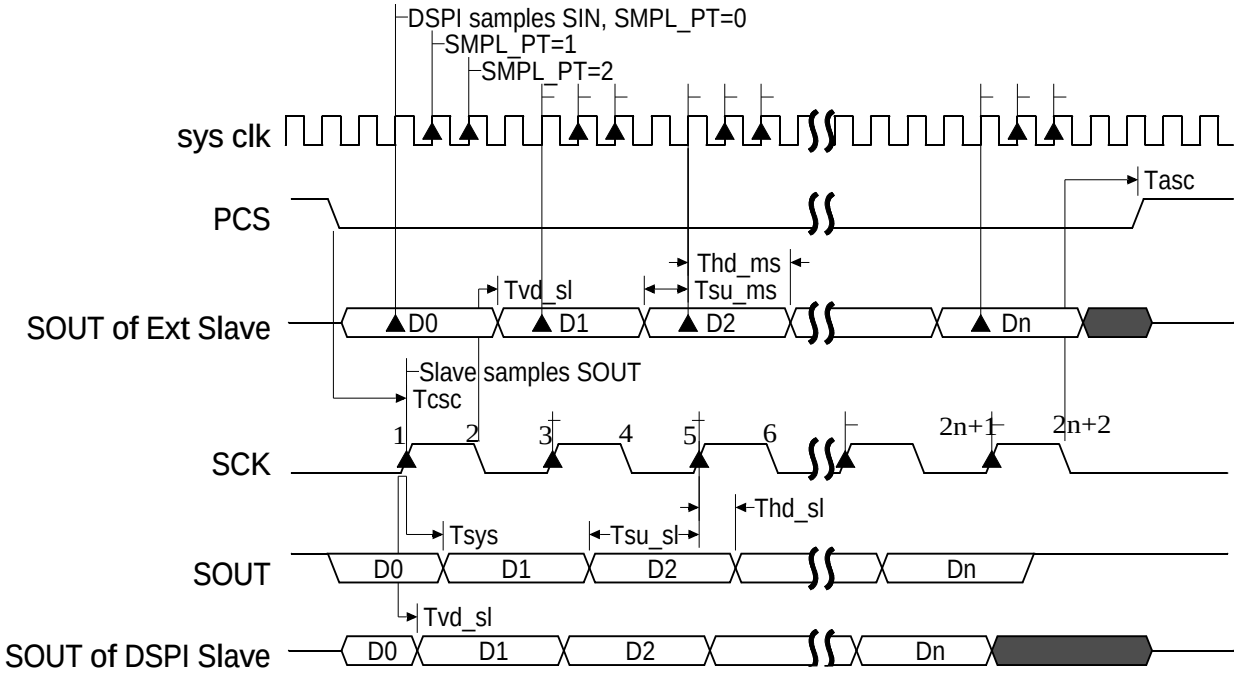


Figure 21-20. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/4$)

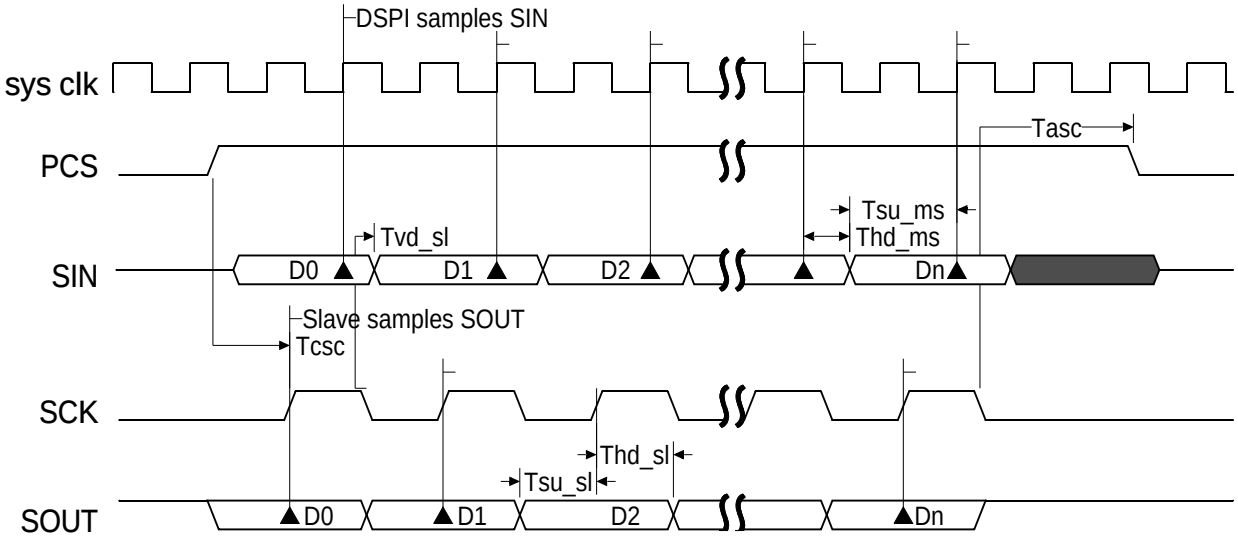


Figure 21-21. dspi Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/2$)

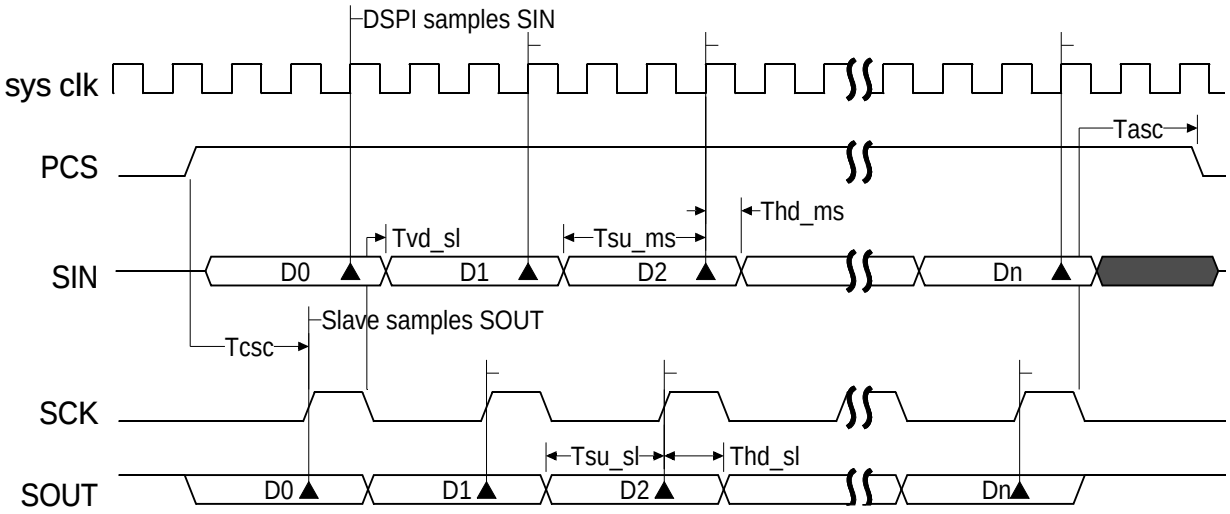


Figure 21-22. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/3$)

21.4.4.4 Modified SPI transfer format (MTFE = 1, CPHA = 1)

Figure 21-23 through Figure 21-25 show the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is shown. At the start of a transfer, the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed, the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even-numbered edges of SCK. The master samples the slave SOUT signal on the odd-numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one-half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit.

NOTE

The SCK to PCS delay must be greater than or equal to half of the SCK period.

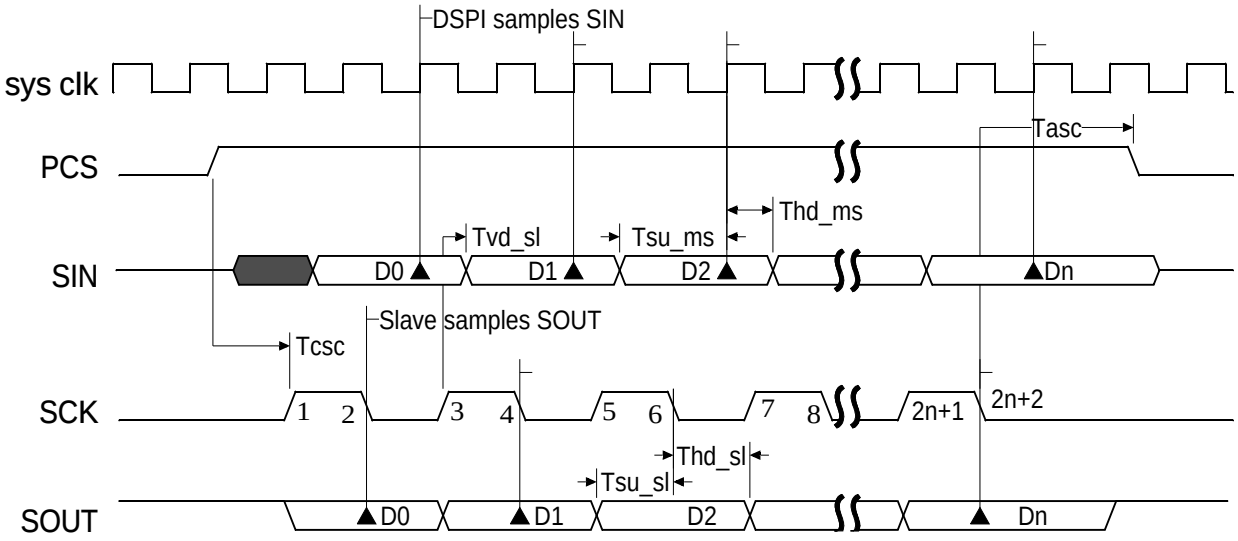


Figure 21-23. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/2$)

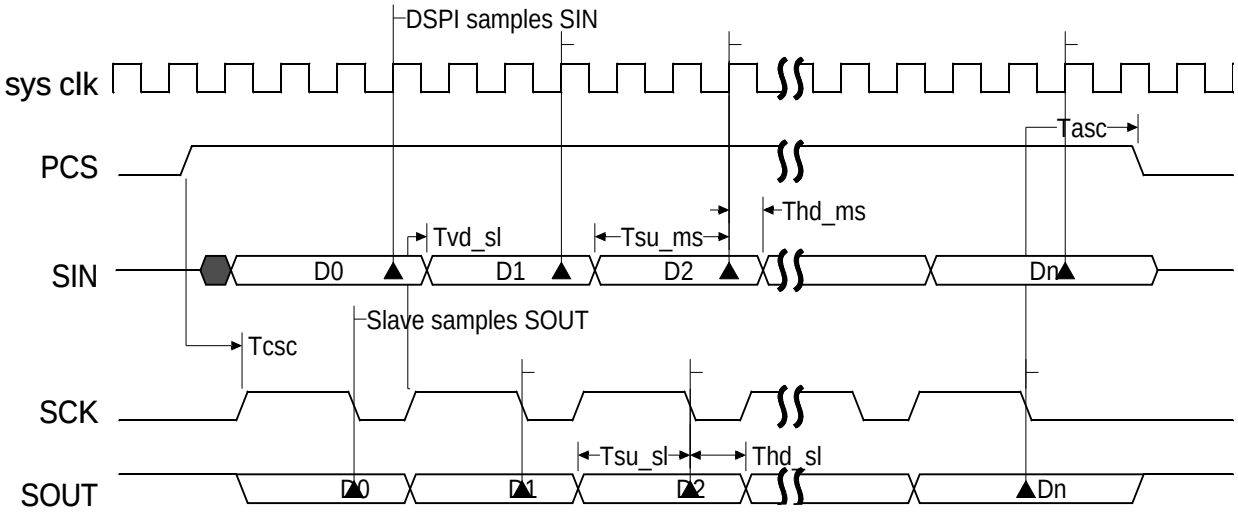


Figure 21-24. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/3$)

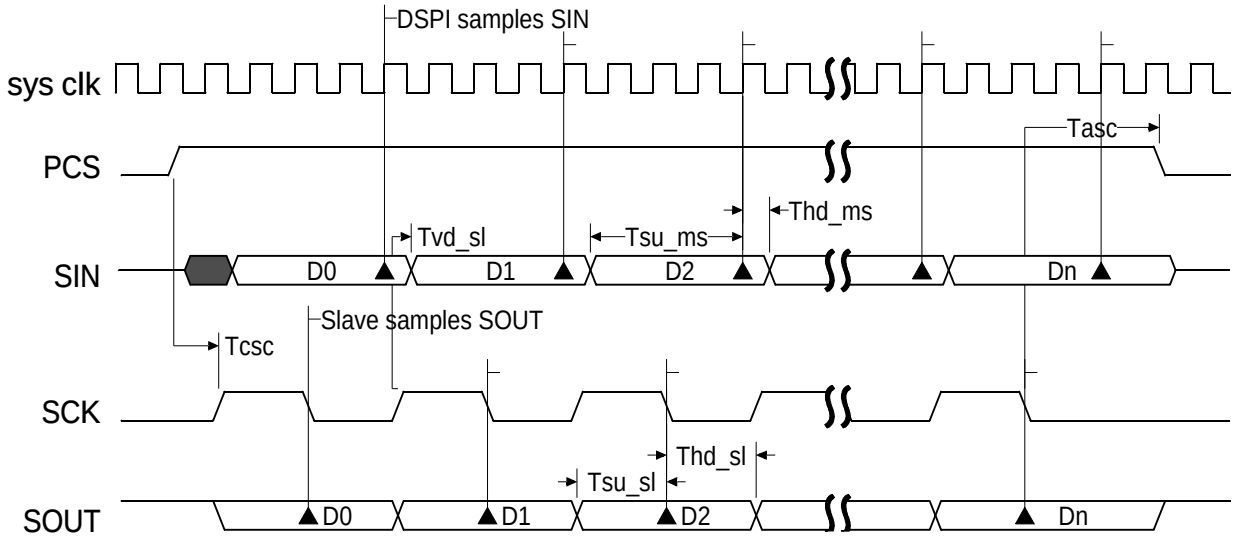
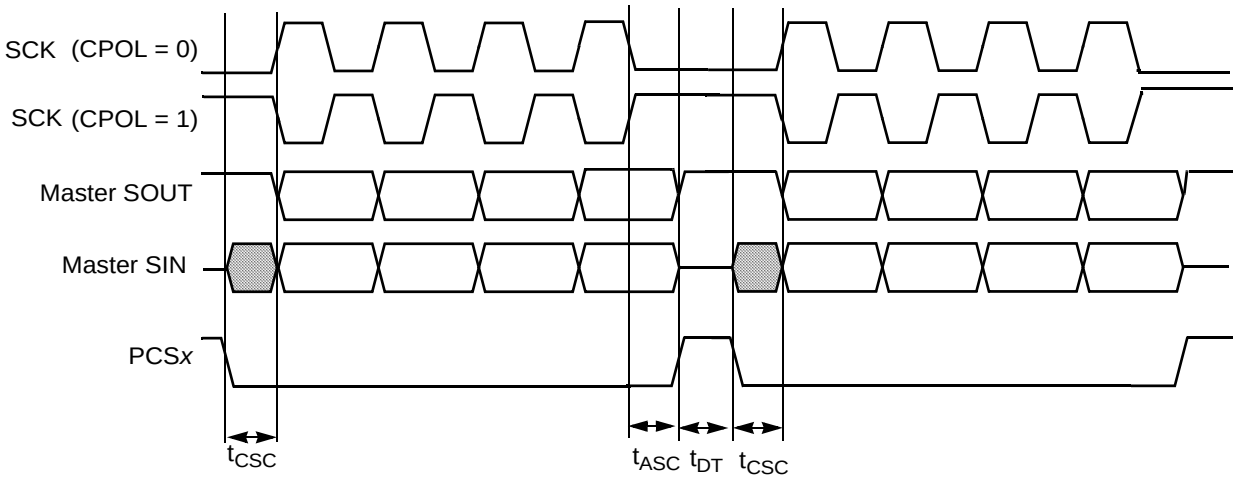


Figure 21-25. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/4$)

21.4.4.5 Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS*n* bits in the DSPI_MCR. Figure 21-26 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.



t_{CSC} = PCS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 21-26. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. Figure 21-27 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

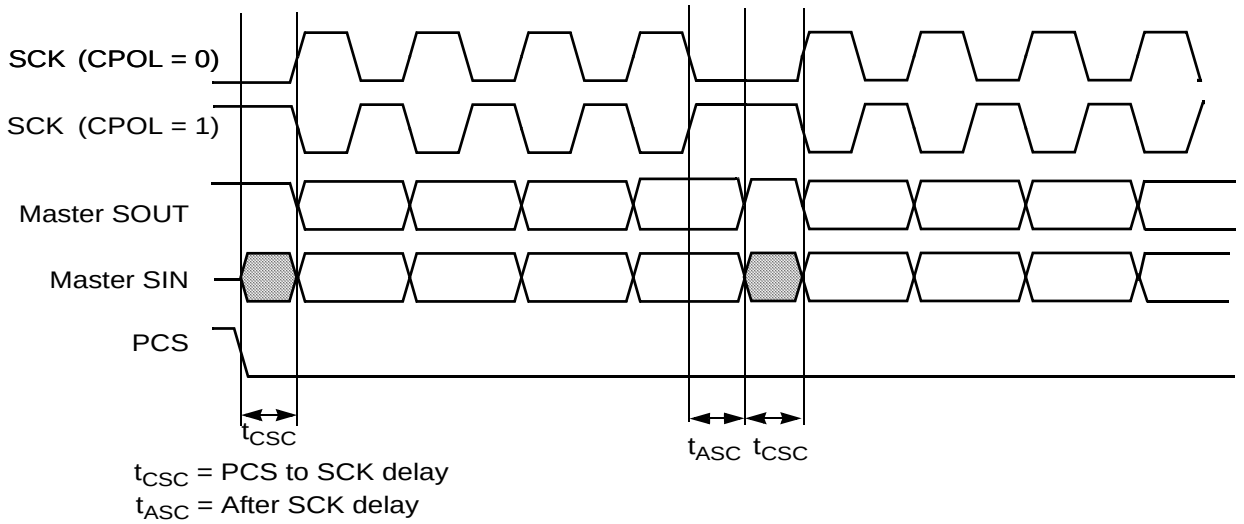


Figure 21-27. Example of continuous transfer (CPHA = 1, CONT = 1)

When using DSPI with continuous selection follow these rules:

- All transmit commands must have the same PCS*n* bits programming.
- The DSPI_CTAR*n* registers selected by transmit commands must be programmed with the same transfer attributes. Only the FMSZ bitfield can be programmed differently in these DSPI_CTAR*n* registers.

NOTE

User must fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. While transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).

When operating in slave mode, ensure that when the last-entry in the TXFIFO is completely transmitted (that is the corresponding TCF flag is asserted and TXFIFO is empty), the slave is deselected for any further serial communication; otherwise an underflow error occurs.

21.4.5 Continuous serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Clearing CPHA is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for Modified Transfer Format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- The currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

It is recommended to keep the baud rate the same while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into the external Stop mode or module disable mode.

Enabling Continuous SCK disables the PCS to SCK delay and the delay after transfer (t_{DT}) is fixed to one $_{SCK}$ cycle. Figure 21-28 shows timing diagram for continuous SCK format with continuous selection disabled.

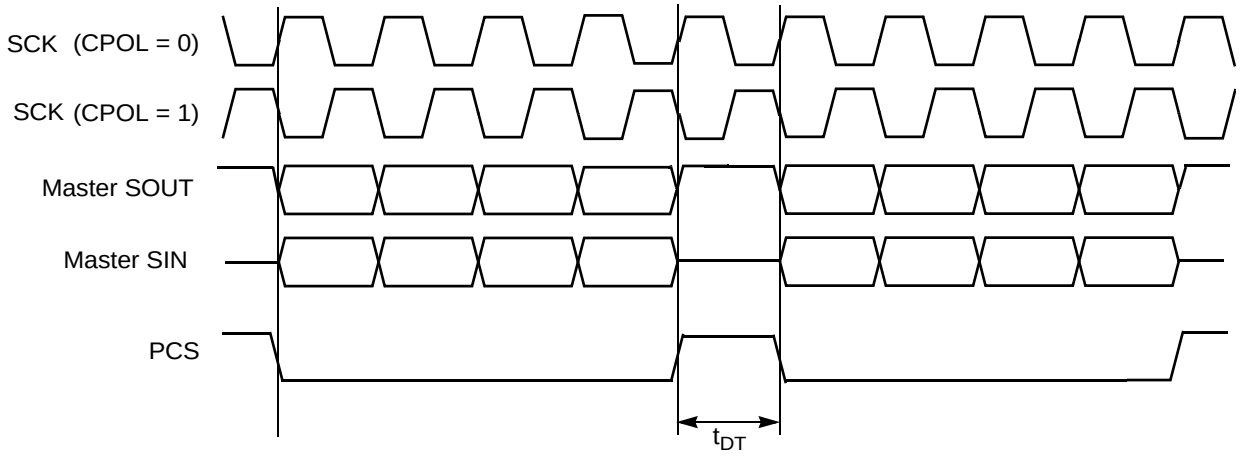


Figure 21-28. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with PCS asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to Section 21.4.1, Start and stop of DSPI transfers).
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 21-29 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

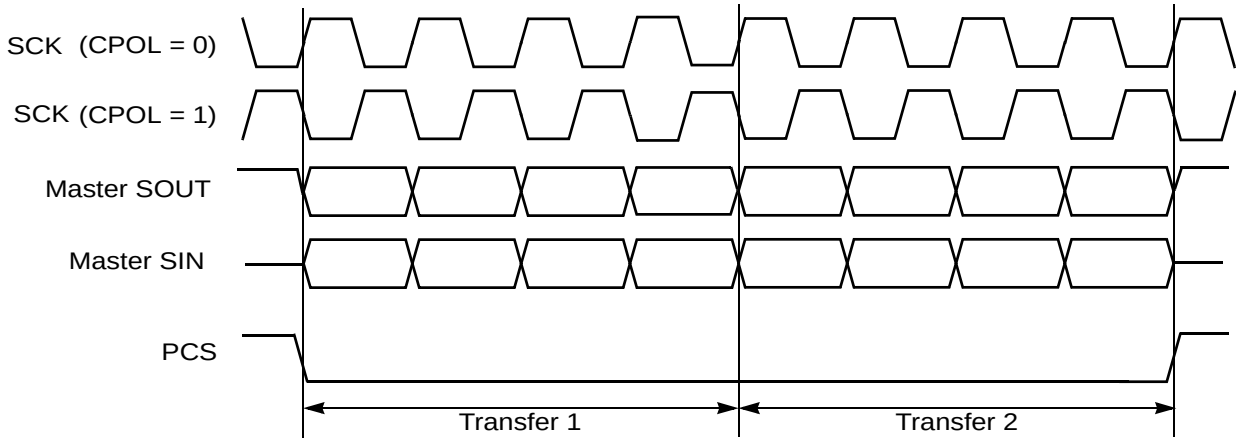


Figure 21-29. Continuous SCK timing diagram (CONT = 1)

NOTE

When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the MCR[CLR_TXF] field before initiating transfer.

21.4.6 Parity generation and check

The DSPI module can generate and check parity in the serial frame. The parity bit replaces the last transmitted bit in the frame. The parity is calculated for all transmitted data bits in frame, not including the last, would be transmitted, data bit. The parity generation/control is done on frame basis. The registers fields, setting frame size defines the total number of bits in the frame, including the parity bit. Thus, to transmit/receive the same number of data bits with parity check, increase the frame size by one versus the same data size frame without the parity check.

Parity can be selected as odd or even. Parity Errors in the received frame set Parity Error flags in the Status register. The Parity Error Interrupt Requests are generated if enabled. The DSPI module can be programmed to stop SPI frame transmission in case of a frame reception with parity error.

21.4.6.1 Parity for SPI frames

When the DSPI is in the master mode the parity generation is controlled by PE and PP bits of the TX FIFO entries (DSPI_PUSHR). Setting the PE bit enables parity generation for transmitted SPI frames and parity check for received frames. PP bit defines polarity of the parity bit.

When continuous PCS selection is used to transmit SPI data, two parity generation scenarios are available:

- Generate/check parity for the whole frame
- Generate/check parity for each sub-frame separately

To generate/check parity for the whole frame, set the PE bit only in the last command/TX FIFO entry forming this frame (with the DSPI_PUSHR register).

To generate/check parity for each sub-frame, set the PE bit in each command/TX FIFO entry forming this frame.

If the parity error occurs for received SPI frame, the DSPI_SR[SPEF] bit is set. If the DSPI_MCR[PES] bit is set, the DSPI stops SPI frame transmission. To resume SPI operation, clear the DSPI_SR[SPEF] bit or the DSPI_MCR[PES] bit.

In slave mode, parity is controlled by the PE and PP bits of the DSPI_CTAR0 register, similar to the master mode parity generation without continuous PCS selection.

21.4.7 Interrupts/DMA requests

The DSPI has several conditions that can only generate interrupt requests; and two conditions that can generate interrupt or DMA request. Table 21-27 lists these conditions.

Table 21-27. Interrupt and DMA request conditions

Condition	Flag	Interrupt	DMA
End of Queue (EOQ)	EOQF	X	—
TX FIFO Fill	TFFF	X	X
Transfer Complete	TCF	X	—
TX FIFO Underflow	TFUF	X	—
RX FIFO Drain	RFDF	X	X
RX FIFO Overflow	RFOF	X	—
SPI Parity Error	SPEF	X	—

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and a Request Enable bit in the DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER). The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests, depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI_RSER register.

The DSPI module also provides a global interrupt request line, which is asserted when any of the individual interrupt request lines is asserted.

21.4.7.1 End of queue interrupt request

The End of Queue Request indicates that the end of a transmit queue is reached. The End of Queue Request is generated when the EOQ bit in the executing SPI command is set and the EOQF_RE bit in the DSPI_RSER is set.

21.4.7.2 Transmit FIFO fill interrupt or DMA request

The Transmit FIFO Fill Request indicates that the TX FIFO is not full. The Transmit FIFO Fill Request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the DSPI_RSER[TFFF_RE] bit is set. The DSPI_RSER[TFFF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

21.4.7.3 Transfer complete interrupt request

The Transfer Complete Request indicates the end of the transfer of a serial frame. The Transfer Complete Request is generated at the end of each frame transfer when the DSPI_RSER[TCF_RE] bit is set.

21.4.7.4 Transmit FIFO underflow interrupt request

The Transmit FIFO Underflow Request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only when the DSPI is operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI_RSER is set, an interrupt request is generated.

21.4.7.5 Receive FIFO drain interrupt or DMA request

The Receive FIFO Drain Request indicates that the RX FIFO is not empty. The Receive FIFO Drain Request is generated when the number of entries in the RX FIFO is not zero, and the DSPI_RSER[RFDF_RE] bit is set. The DSPI_RSER[RFDF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

21.4.7.6 Receive FIFO overflow interrupt request

The Receive FIFO Overflow Request indicates that an overflow condition in the RX FIFO has occurred. A Receive FIFO Overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The DSPI_RSER[RFOF_RE] bit must be set for the interrupt request to be generated.

Depending on the state of the DSPI_MCR[ROOE] bit, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

21.4.7.7 SPI frame parity error interrupt request

The SPI frame parity error flag indicates that an SPI frame with parity error had been received. The DSPI_RSER[SPEF_RE] bit must be set for the interrupt request to be generated.

21.4.8 Power- saving features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode—Clock gating of non-memory mapped logic

21.4.8.1 .Stop mode (external stop mode)

The DSPI supports the stop mode protocol. When a request is made to enter external stop mode, the DSPI block acknowledges the request. If a serial transfer is in progress, the DSPI waits until it reaches the frame boundary before it is ready to have its clocks shut off. While the clocks are shut off, the DSPI

memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in External Stop mode.

21.4.8.2 Module disable mode

Module disable mode is a block-specific mode that the DSPI can enter to save power. Host CPU can initiate the module disable mode by setting the MDIS bit in the DSPI_MCR. The module disable mode can also be initiated by hardware.

When the MDIS bit is set, the DSPI negates Clock Enable signal at the next frame boundary. If implemented, the Clock Enable signal can stop the clock to the non-memory mapped logic. When Clock Enable is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different effect when the DSPI is in the module disable mode. Reading the RX FIFO Pop Register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO Push Register does not change the state of the TX FIFO. Clearing either of the FIFOs has no effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR have no effect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect. Writing to the DSPI_TCR during module disable mode has no effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

21.5 Initialization and application information

21.5.1 Managing queues

The queues are not part of the DSPI, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI Configuration.

1. When DSPI executes last command word from a queue, the EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the DSPI_SR[EOQF] flag bit is set.
3. The setting of the EOQF flag disables serial transmission and reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is cleared to indicate the STOPPED state.
4. The DMA can continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the DMA Controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI_SR or by checking RFDF in the DSPI_SR after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues
8. Flush TX the FIFO by writing a ‘1’ to the CLR_TXF bit in the DSPI_MCR. Flush RX FIFO by writing a ‘1’ to the CLR_RXF bit in the DSPI_MCR.

9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPI_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

21.5.2 Switching master and slave mode

When changing modes in the DSPI, follow the steps below.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

21.5.3 Baud rate settings

Table 21-28 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI*n*_CTAR registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 21-28. Baud rate values (bps)

		Baud rate divider prescaler values (DSPI*n*_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud Rate scaler values (DSPI*n*_CTAR[BR])	**2**	25.0 MHz	16.7 MHz	10.0 MHz	7.14 MHz
	4	12.5 MHz	8.33 MHz	5.00 MHz	3.57 MHz
	6	8.33 MHz	5.56 MHz	3.33 MHz	2.38 MHz
	8	6.25 MHz	4.17 MHz	2.50 MHz	1.79 MHz
	16	3.12 MHz	2.08 MHz	1.25 MHz	893 kHz
	32	1.56 MHz	1.04 MHz	625 kHz	446 kHz
	64	781 kHz	521 kHz	312 kHz	223 kHz
	128	391 kHz	260 kHz	156 kHz	112 kHz
	256	195 kHz	130 kHz	78.1 kHz	55.8 kHz
	512	97.7 kHz	65.1 kHz	39.1 kHz	27.9 kHz
	1024	48.8 kHz	32.6 kHz	19.5 kHz	14.0 kHz
	2048	24.4 kHz	16.3 kHz	9.77 kHz	6.98 kHz
	4096	12.2 kHz	8.14 kHz	4.88 kHz	3.49 kHz
	8192	6.10 kHz	4.07 kHz	2.44 kHz	1.74 kHz
	16384	3.05 kHz	2.04 kHz	1.22 kHz	872 Hz
	32768	1.53 kHz	1.02 kHz	610 Hz	436 Hz

21.5.4 Delay settings

Table 21-29 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (T_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

Table 21-29. Delay values

		Delay prescaler values (DSPI*n*_CTAR[PBR])			
		1	**3**	**5**	**7**
Delay scaler values (DSPI*n*_CTAR[DT])	**2**	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

21.5.5 Calculation of FIFO pointer addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

See Section 21.4.2.4, Transmit First In First Out (TX FIFO) buffering mechanism, and Section 21.4.2.5, Receive First In First Out (RX FIFO) buffering mechanism, for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 21-30 illustrates the concept of first-in and last-in FIFO entries along with the FIFO Counter.

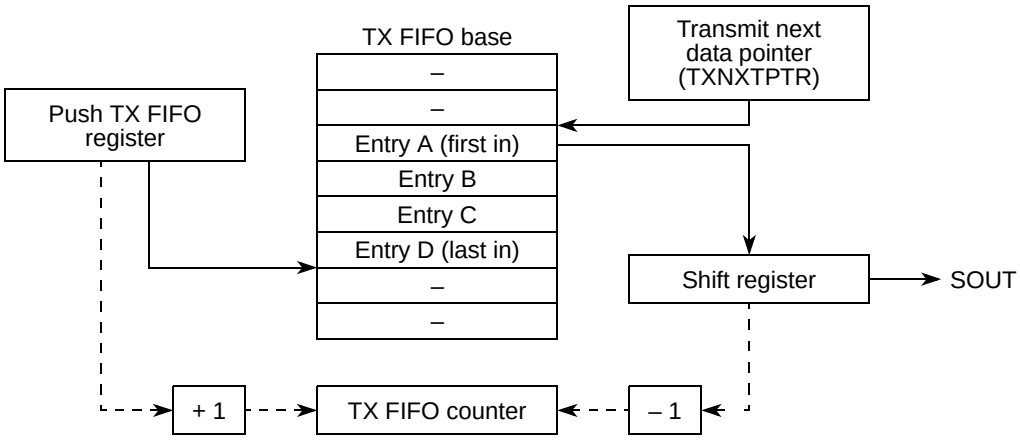


Figure 21-30. TX FIFO pointers and counter

21.5.5.1 Address calculation for the first-in entry and last-in entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR}) \quad \textit{Eqn. 21-7}$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1) \bmod (\text{TXFIFOdepth}) \quad \textit{Eqn. 21-8}$$

TX FIFO Base — Base address of TX FIFO
TXCTR — TX FIFO Counter
TXNXTPTR — Transmit Next Pointer
TX FIFO Depth — Transmit FIFO depth (5 entries)

21.5.5.2 Address calculation for the first-in entry and last-in entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR}) \quad \textit{Eqn. 21-9}$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1) \bmod (\text{RXFIFOdepth}) \quad \textit{Eqn. 21-10}$$

RX FIFO Base — Base address of RX FIFO
RXCTR — RX FIFO counter
POPNXTPTR — Pop Next Pointer
RX FIFO Depth — Receive FIFO depth (16 entries on DSPI0 and DSPI1, 5 entries on DSPI2))

Chapter 22
External Bus Interface (EBI)

22.1 Introduction

22.1.1 Overview

The External Bus Interface (EBI) handles the transfer of information between the internal busses and the memories or peripherals in the external address space. The EBI includes a memory controller that generates interface signals to support a variety of external memories. This includes Single Data Rate (SDR) burst mode flash, SRAM, and asynchronous memories. It supports up to four regions (via chip selects), each with its own programmed attributes (plus four calibration chip-select regions).

22.1.2 Features

- 64-bit AMBA AHB (no arbitration support)
- 32-bit Address bus with transfer size indication (24 address signals available on pins)
- 32-bit Data bus (16-bit Data Bus Mode also supported)
- Multiplexed Address on Data pins (single master)
- Memory controller with support for various memory types:
  - synchronous burst SDR flash and SRAM
  - asynchronous/legacy flash and SRAM
- Burst support (wrapped only)
- Bus monitor
- Port size configuration per chip select (16 or 32 bits)
- Configurable wait states
- Configurable internal or external transfer acknowledge ($\overline{TA}$) per chip select
- Four Chip-Select ($\overline{CS}$[0:3]) signals
- Four Write/Byte Enable ($\overline{WE}$[0:3]/$\overline{BE}$[0:3]) signals
- Slower-speed clock modes
- Stop and Module Disable Modes for power savings
- Optional automatic CLKOUT gating to save power and reduce EMI
- Misaligned access support (for chip-select accesses only)
- Compatible with MPC5xx external bus (with some limitations)

22.1.3 Modes of operation

The mode of the EBI is determined by the MDIS, EXTM, and AD_MUX bits in the EBI_MCR. See Section 22.3.1.1, EBI Module Configuration Register (EBI_MCR), for details. Slower-speed modes,

Debug Mode, Stop Mode, and Factory Test Mode are modes that the MCU may enter, in parallel to the EBI being configured in one of its block-specific modes.

22.1.3.1 Single master mode

In single master mode, the EBI responds to internal requests matching one of its regions, but ignores all externally initiated bus requests. The MCU is the only master allowed to initiate transactions on the external bus in this mode; therefore, it acts as a parked master and does not have to arbitrate for the bus before starting each cycle. The $\overline{BR}$, $\overline{BG}$, and $\overline{BB}$ signals are not used by the EBI in this mode, and are available for use in an alternate function by another block of the MCU. Single master mode is entered when EXTM = 0 and MDIS = 0 in the EBI_MCR.

External master mode is not supported on this chip.

22.1.3.2 Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the EBI can be stopped while in module disable mode. Logic on the MCU external to the EBI is needed to fully implement the module disable mode (to shut off the clock). Internal master requests made to the external bus in module disable mode are terminated with transfer error (internally, no external $\overline{TEA}$ assertion). Module disable mode is entered when MDIS = 1 in the EBI_MCR.

22.1.3.3 Stop mode

The EBI supports the stop mode mechanism used for MCU power management. When a request is made to enter stop mode (controlled in device logic outside EBI), the EBI block completes any pending bus transactions and acknowledges the stop request. After the acknowledgement, the system clock input may be shut off by the clock driver on the MCU. While the clocks are shut off, the EBI is not accessible. While in stop mode, accesses to the EBI from the internal master are terminated with a transfer error (internally, no external $\overline{TEA}$ assertion).

22.1.3.4 Slower-speed modes

In slower-speed modes, the external CLKOUT frequency is divided (by 2, 3, etc.) compared with that of the internal system bus. The EBI behavior remains dictated by the mode of the EBI, except that it drives and samples signals at the CLKOUT frequency rather than the internal system frequency. This mode is selected by writing a clock control register in a block outside of the EBI. Refer to the device-specific documentation to see which slower-speed modes are available for a particular MCU (1/2, 1/3, etc.).

22.1.3.5 16-bit data bus mode

The EBI on the MPC5675K supports a 16-bit data bus mode. In this mode, only 16 data signals are used by the EBI. The user can select which 16 data signals are used (DATA[0:15] or DATA[16:31]) by writing the D16_31 bit in the EBI_MCR.

For EBI-mastered accesses, the operation in 16-bit Data Bus Mode (DBM = 1, PS = *x*) is similar to a chip-select access to a 16-bit port in 32-bit Data Bus Mode (DBM = 0, PS = 1), except for the case of a non-chip-select access of exactly 32-bit size.

EBI-mastered non-chip-select accesses of exactly 32-bit size are supported via a two (16-bit) beat burst for both reads and writes. See Section 22.4.2.9, Non-chip-select burst in 16-bit data bus mode. Non-chip-select transfers of non-32-bit size are supported in standard non-burst fashion.

16-bit Data Bus Mode is entered when DBM = 1 in the EBI_MCR. Some MCUs may have DBM = 1 by default out of reset. See the device-specific documentation for the DBM and D16_31 reset values.

22.1.3.6 Multiplexed address on data bus mode

This mode covers several cases aimed at reducing pin count on MCU and external components. In this mode, the DATA pins drive (for internal master cycles) the address value on the first clock of the cycle (while $\overline{\text{TS}}$ is asserted).

The memory controller supports per-chip-select selection of multiplexing address/data through the BR*x*[AD_MUX] bit.

Address on Data bus multiplexing also supports the 16-bit data bus mode (MCR[DBM] = 1) and 16-bit memories (OR*x*[PS] = 1). The user can select which 16 data signals are used (DATA[0:15] or DATA[16:31]) by writing the D16_31 bit in the EBI_MCR. For either setting of D16_31, the 16 LSBs of external address (ADDR[16:31]) are driven onto the selected 16 DATA pins. If additional address lines are required to interface to the memory, then non-muxed address pins are sometimes (see note below) required to complete the address space (for example, ADDR[8:15] are commonly present as non-muxed address pins).

NOTE

The EBI also drives the unused 16 DATA signals with the MSBs of the external address, zero-padded in front (for example, when D16_31 bit is set for a device with 24 ADDR pins, the EBI drives (0b00000000,ADDR[8:15]) on DATA[0:15]. This allows the device to optionally use DATA[8:15] for the upper 8 external address lines instead of requiring separate non-muxed ADDR[8:15] pins. This is relevant primarily for devices that support both 32-bit and 16-bit A/D muxed operation, so therefore have DATA[16:31] pins present on the device, and in that case are not required to have separate ADDR pins.

For more details (for example, timing diagrams), see Section 22.4.2.11, Address/data multiplexing.

22.1.3.7 Debug mode

When the MCU is in debug mode, the EBI behavior is unaffected and remains dictated by the mode of the EBI.

22.2 External signal description

22.2.1 Overview

Table 22-1 lists the external pins used by the EBI. Not all signals listed here are externally available.

Table 22-1. Signal properties

Name	I/O Type	Function	Pull[1]
ADDR[8:31]	I/O	Address bus	—
$\overline{\text{BDIP}}$	Output	Burst Data in Progress	Up
CLKOUT[2]	Output	Clock out	—
DATA[0:31]	I/O	Data bus[3]	—
$\overline{\text{OE}}$	Output	Output Enable	Up
RD_$\overline{\text{WR}}$	I/O	Read_Write	Up
$\overline{\text{TA}}$	I/O	Transfer Acknowledge	Up
$\overline{\text{TEA}}$	I/O	Transfer Error Acknowledge	Up
$\overline{\text{TS}}$	I/O	Transfer Start	Up
TSIZ[0:1]	I/O	Transfer Size	—
$\overline{\text{WE}}$[0:3]/$\overline{\text{BE}}$[0:3]	Output	Write/Byte Enables	Up
ALE	Output	Address Latch Enable	Up

1 This column shows which signals require a weak pullup or pulldown.

2 The CLKOUT signal is driven by the System Clock Block outside the EBI.

3 In Address/Data multiplexing modes, data also shows the address during the address phase.

22.2.2 Detailed signal descriptions

22.2.2.1 ADDR [8:31] — Address Lines 8–31

The ADDR[8:31] signals specify the physical address of the bus transaction.

The 24 address lines correspond to bits 8-31 of the EBI's 32-bit internal address bus.

22.2.2.2 $\overline{\text{BDIP}}$ — Burst Data in Progress

$\overline{\text{BDIP}}$ is asserted to indicate that the master is requesting another data beat following the current one.

This signal is driven by the EBI on all EBI-mastered external burst cycles, but is only sampled by burst mode memories that have a corresponding pin. See Section 22.4.2.5, Burst transfer.

22.2.2.3 CLKOUT — Clock Output

CLKOUT is a general-purpose clock output signal to connect to the clock input of SDR external memories and in some cases to the input clock of another MCU in multi-master configurations.

22.2.2.4 External calibration bus clock divider

EBI Clock is derived from sys_clk by means of a divider. The divider is in the Clock Generation Module at address 0xC3FE017F (byte) and uses the 6 least significant bits.

Divide ratio is n+1, which means a 0 in the register is div by 1, a 1 in the register is div by 2, and so on. The highest divide value is 0x3F which results in div by 64.

Word (16 bit) and Long (32 bit) accesses are also possible but need to be aligned (0xC3FE017E for Word and 0xC3FE017C for Long).

22.2.2.5 $\overline{\text{CS}}$ [0:3] — Chip Selects 0-3

$\overline{\text{CS}}$x is asserted by the master to indicate that this transaction is targeted for a particular memory bank on the Primary external bus.

$\overline{\text{CS}}$ is driven in the same clock as the assertion of $\overline{\text{TS}}$ and valid address, and is kept valid until the cycle is terminated. See Section 22.4.1.4, Memory controller with support for various memory types, for details on chip-select operation.

22.2.2.6 DATA [0:31] — Data Lines 0-31

The DATA[0:31] signals contain the data to be transferred for the current transaction. DATA[0:31] is driven by the EBI when it owns the external bus and it initiates a write transaction to an external device. DATA[0:31] is driven by an external device during a read transaction from the EBI. For 8-bit and 16-bit transactions, the byte lanes not selected for the transfer do not supply valid data. DATA[0:31] is driven by the EBI in the address phase with the ADDR value if the Address on Data multiplexing mode is enabled. See Section 22.1.3.6, Multiplexed address on data bus mode, for details.

In 16-bit data bus mode, (or for chip-select accesses to a 16-bit port), only DATA[0:15] or DATA[16:31] are used by the EBI, depending on the setting of the D16_31 bit in the EBI_MCR. See Section 22.1.3.5, 16-bit data bus mode.

22.2.2.7 $\overline{\text{OE}}$ — Output Enable

$\overline{\text{OE}}$ indicates when an external memory is permitted to drive back read data. External memories must have their data output buffers off when $\overline{\text{OE}}$ is negated. $\overline{\text{OE}}$ is only asserted for chip-select accesses.

For read cycles, $\overline{\text{OE}}$ is asserted one clock after $\overline{\text{TS}}$ assertion (by default with EBI_BR[EOE] = 00) and held until the termination of the transfer. For write cycles, $\overline{\text{OE}}$ is negated throughout the cycle.

22.2.2.8 RD_$\overline{\text{WR}}$ — Read / Write

RD_$\overline{\text{WR}}$ indicates whether the current transaction is a read access or a write access.

RD_$\overline{\text{WR}}$ is driven in the same clock as the assertion of $\overline{\text{TS}}$ and valid address, and is kept valid until the cycle is terminated.

22.2.2.9 $\overline{\text{TA}}$ — Transfer Acknowledge

$\overline{\text{TA}}$ is asserted to indicate that the slave has received the data (and completed the access) for a write cycle, or returned data for a read cycle. If the transaction is a burst read, $\overline{\text{TA}}$ is asserted for each one of the transaction beats. For write transactions, $\overline{\text{TA}}$ is only asserted once at access completion, even if more than one write data beat is transferred.

$\overline{\text{TA}}$ is driven by the EBI when the access is controlled by the chip selects (and SETA = 0). Otherwise, $\overline{\text{TA}}$ is driven by the slave device to which the current transaction was addressed.

See Section 22.4.2.8, Termination signals protocol, for more details.

22.2.2.10 $\overline{\text{TEA}}$ — Transfer Error Acknowledge

$\overline{\text{TEA}}$ is asserted by either the EBI or an external device to indicate that an error condition has occurred during the bus cycle.

$\overline{\text{TEA}}$ is asserted by the EBI when the internal bus monitor detected a timeout error.

See Section 22.4.2.8, Termination signals protocol, for more details.

22.2.2.11 $\overline{\text{TS}}$ — Transfer Start

$\overline{\text{TS}}$ is asserted by the current bus owner to indicate the start of a transaction on the external bus.

$\overline{\text{TS}}$ is only asserted for the first clock cycle of the transaction, and is negated in the successive clock cycles until the end of the transaction.

22.2.2.12 TSIZ [0:1] — Transfer Size 0–1

TSIZ[0:1] indicates the size of the requested data transfer, for aligned internal master transfers only. For the special case of misaligned chip-select transfers, TSIZ[0:1] values are not specified, and must not be used by external devices for those transfers.

The TSIZ[0:1] signals may be used with ADDR[30:31] to determine which byte lanes of the data bus are involved in the transfer. For non-burst transfers, the TSIZ[0:1] signals specify the number of bytes starting from the byte location addressed by ADDR[30:31]. In burst transfers, the value of TSIZ[0:1] is always 00.

Table 22-2. TSIZ[0:1] encoding

Burst Cycle	TSIZ[0:1]	Transfer Size
N	0b01	Byte
N	0b10	16-bit
N	0b11	Reserved
N	0b00	32-bit
Y	0b00	Burst

The SIZEN bit has no effect on the EBI when it is mastering a transaction on the external bus. TSIZ[0:1] is still driven appropriately by the EBI. See Section 22.3.1.1, EBI Module Configuration Register (EBI_MCR).

22.2.2.13 $\overline{WE}$ [0:3] / $\overline{BE}$ [0:3] — Write/Byte Enables 0–3

Write enables are used to enable program operations to a particular memory. These signals can also be used as byte enables for read and write operation by setting the WEBS bit in the appropriate Base Register. $\overline{WE}$[0:3]/BE[0:3] are only asserted for chip-select accesses.

For chip-select accesses to a 16-bit port, only $\overline{WE}$[0:1]/$\overline{BE}$[0:1] are used by the EBI, regardless of which half of the DATA bus is selected via the D16_31 bit in the EBI_MCR.

See Section 22.4.1.10, Four Write/Byte Enable (WE/BE) signals, for more details on $\overline{WE}$[0:3]/$\overline{BE}$[0:3] functionality.

22.2.2.14 ALE - Output

For more information, please refer to Section 22.4.1.12, Address Latch Enable (ALE).

22.2.3 Signal output buffer enable logic by mode

Table 22-3 describes how the EBI drives its output buffer enable (OBE) signals. These are internal signals from the EBI to device logic outside the EBI, that determine when the EBI strongly drives values on pins. When the OBE for an EBI signal is asserted (1), the EBI strongly drives the value on that pin. When the OBE is negated (0), the EBI does not drive the signal, and the value is determined by internal or external pullups/pulldowns, and/or device logic outside EBI block. The logic in Table 22-3 can be overwritten by device logic, so see the device-specific documentation for any exceptions to the logic below.

Table 22-3. Signal output buffer enable logic by mode[1]

Signal	OBE value by mode (1 = strongly driven, 0 = not driven by EBI)		
	Module disable mode[2] (EXTM = X, MDIS = 1)	Single master mode (EXTM = 0, MDIS = 0)	External master mode (EXTM = 1, MDIS = 0)
ADDR[8:31]	0	1	1 during int. master access
BDIP	0	1	1 during int. master access
$\overline{CAL\_CS}$[0:3]	0	1	1 during int. master access
DATA[16:31]	0	Only 1 during write access or on Address phase when Addr/Data muxing is enabled.	
$\overline{OE}$	0	1	1 during int. master access
RD_$\overline{WR}$	0	1	1 during int. master access
$\overline{TA}$	0	Only 1 during chip-select (or cal-chip-select) SETA = 0 access	
$\overline{TEA}$	0	Only 1 for 2 cycles when timeout occurs	
$\overline{TS}$	0	1	1 during int. master access
TSIZ[0:1]	0	1	1 during int. master access
$\overline{WE}$[0:3]/$\overline{BE}$[0:3]	0	1	1 during int. master access

1 The values in this table only indicate when signals are strongly driven, not the logic value on the pin itself.

[2] This assumes that the clock to the EBI is shut off when MDIS = 1. This is an optional device feature. If the clocks are left running to EBI even when MDIS = 1, then the EBI OBE behavior is as if in Single Master Mode (though EBI accesses are not supported in this scenario).

22.3 Memory map and register description

Table 22-4 shows the EBI registers.

Table 22-4. EBI memory map

Offset from EBI_BASE (0xC3F8_4000)	Register	Access [1]	Reset Value [2, 3]	Location
0x0000	EBI Module Configuration Register (EBI_MCR)	R/W	0x0000_0805	on page 611
0x0004	Reserved			
0x0008	EBI Transfer Error Status Register (EBI_TESR)	R	0x0000_0000	on page 613
0x00C	EBI Bus Monitor Control Register (EBI_BMCR)	R/W	0x0000_FF80	on page 614
0x0010	EBI Base Register Bank 0 (EBI_BR0)	R/W	0x2000_0002	on page 614
0x0014	EBI Option Register Bank 0 (EBI_OR0)	R/W	0xE000_0000	on page 617
0x0018	EBI Base Register Bank 1 (EBI_BR1)	R/W	0x2000_0002	on page 614
0x001C	EBI Option Register Bank 1 (EBI_OR1)	R/W	0xE000_0000	on page 617
0x0020	EBI Base Register Bank 2 (EBI_BR2)	R/W	0x2000_0002	on page 614
0x0024	EBI Option Register Bank 2 (EBI_OR2)	R/W	0xE000_0000	on page 617
0x0028	EBI Base Register Bank 3 (EBI_BR3)	R/W	0x2000_0002	on page 614
0x002C	EBI Option Register Bank 3 (EBI_OR3)	R/W	0xE000_0000	on page 617
0x0030–0xFFFF	Reserved			

[1] In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

[2] In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

[3] In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

22.3.1 Register descriptions

NOTE

Other than the exceptions noted below, EBI registers must not be written while a transaction to the EBI (from internal master) is in progress (or within 2 CLKOUT cycles after a transaction has just completed, to allow internal state machines to go IDLE). In those cases, the behavior is undefined.

Exceptions that can be written while an EBI transaction is in progress:

— All bits in EBI_TESR

— SIZE, SIZEN fields in EBI_MCR

See Section 22.5.1, Booting from external memory, for related application information.

22.3.1.1 EBI Module Configuration Register (EBI_MCR)

The EBI Module Configuration Register (EBI_MCR) contains bits that configure various attributes associated with EBI operation.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZEN	SIZE		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACGE	EXTM	EARB	EARP		0	0	0	0	MDIS	0	0	0	D16_31	AD_MUX	DBM
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1

Figure 22-1. EBI Module Configuration Register (EBI_MCR)

Table 22-5. EBI_MCR field descriptions

Field	Description
SIZEN	SIZE enable. The SIZEN bit enables the control of transfer size by the SIZE field (as opposed to external TSIZ pins) for external master transactions to internal address space. 0 Transfer size controlled by TSIZ[0:1] pins. 1 Transfer size controlled by SIZE field.
SIZE	Transfer size' The SIZE field determines the transfer size of external master transactions to internal address space when SIZEN = 1. This field is ignored when SIZEN = 0. This field must be written before doing an external master transfer of a different size than the current setting. The DBM bit affects how the SIZE field should be set for the special case of 64-bit external master transfers. Table 22-6 shows how to set SIZE appropriately for all the cases.
ACGE	Automatic CLKOUT Gating Enable. The ACGE bit enables the EBI feature of turning off CLKOUT (holding it high) during idle periods in-between external bus accesses. 0 Automatic CLKOUT Gating is disabled. 1 Automatic CLKOUT Gating is enabled.
EXTM	External Master Mode 0 External Master Mode is inactive (Single Master Mode) 1 The EXTM bit enables the External Master Mode of operation when MDIS = 0. When MDIS = 1, the EXTM bit is a don't care, and is treated as 0. In External Master Mode, an external master on the external bus can access any internal memory-mapped space while the internal e200z core is fully operational. When EXTM = 0, only internal masters can access the internal memory space. This bit also determines the functionality of the $\overline{BR}$, $\overline{BG}$ and $\overline{BB}$ signals.External Master Mode is active. **Note:** External master mode is not supported on this chip.

Table 22-5. EBI_MCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>EARB</td><td>External Arbitration
0 Internal arbitration is used.
When EXTM = 0, the EARB bit is a don’t care, and is treated as 0.External arbitration is used.</td></tr>
<tr><td>EARP</td><td>External Arbitration Request Priority. This field defines the priority of an external master’s arbitration request (0-2), with 2 being the highest priority level (EARP = 11 is reserved). This field is valid only when EARB = 00 (internal arbitration). The internal masters of the MCU have a fixed priority of 1. By default, internal and external masters have equal priority.
<table>
<tr><th>EARP</th><th>Priority</th></tr>
<tr><td>0b00</td><td>MCU has priority</td></tr>
<tr><td>0b01</td><td>Equal priority, round-robin used</td></tr>
<tr><td>0b10</td><td>External master has priority</td></tr>
<tr><td>0b11</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>MDIS</td><td>Module Disable Mode. The MDIS bit controls an internal EBI “enable clk” signal which can be used (if MCU logic supports) to control the clocks to the EBI. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the EBI, effectively putting the EBI in a software controlled power-saving state. See Section 22.1.3.2, Module disable mode, for more information. No external bus accesses can be performed when the EBI is in Module Disable Mode (MDIS = 1).
0 Module Disable Mode is inactive (assert “enable clk” signal).
1 Module Disable Mode is active (negate “enable clk” signal).</td></tr>
<tr><td>D16_31</td><td>Data Bus 16_31 Select. The D16_31 bit controls whether the EBI uses the DATA[0:15] or DATA[16:31] signals, when in 16-bit Data Bus Mode (DBM = 1) or for chip-select accesses to a 16-bit port (PS = 1).
0 DATA[0:15] signals are used for 16-bit port accesses.
1 DATA[16:31] signals are used for 16-bit port accesses.
Note: One usage case exists where D16_31 = 0 cannot be used. For systems that are using both a 32-bit non-chip-select device AND a 16-bit device (chip-select or non-chip-select), the user must set D16_31 = 1. Otherwise (if D16_31 = 0 is used), when an access to the 16-bit device is performed, the 32-bit non-chip-select device may decode the wrong upper address bits and may respond to that access, causing contention on the shared address/data lines going to both devices. This is due to the 32-bit device expecting address 8:15 on DATA[8:15] pins, whereas D16_31 = 0 has address 24:31 on those pins. This contention case is avoided when D16_31 = 1, because the shared address/data pins have the same functions for both the 16-bit and 32-bit memory accesses (address 16:31 on DATA[16:31] pins). This contention case is also avoided when only chip-select devices are used, since the devices ignore accesses when their corresponding chip-select is not asserted.</td></tr>
<tr><td>AD_MUX</td><td>Address on Data Bus Multiplexing Mode. The AD_MUX bit controls whether non-chip-select accesses have the address driven on the data bus in the address phase of a cycle.
0 Only Data on data pins for non-CS accesses.
1 Address on Data Multiplexing Mode is used for non-CS accesses.</td></tr>
<tr><td>DBM</td><td>Data Bus Mode. The DBM bit controls whether the EBI is in 32-bit or 16-bit Data Bus Mode.
0 32-bit Data Bus Mode is used.
1 16-bit Data Bus Mode is used.</td></tr>
</table>

Table 22-6. SIZE Encoding[1]

DBM	Size of Ext. Master Transfer[2]	SIZE
X[3]	Byte	0b01
X	16-bit	0b10
X	32-bit[4]	0b00
0	64-bit	0b00
1	64-bit	0b10
X	Reserved	0b11

1 This table is not affected by width of internal AMBA bus (32 or 64 bits), only by the size of the transfer.

2 This refers to size of external master transfer on external master MCU as determined by the source code. For example, if the external master code does a 64-bit load (or store) to an EBI with a 32-bit bus (DBM = 0), this is broken into two 32-bit transfers, so SIZE field must be set to 32 bits (0b00). For a 16-bit bus, the 64-bit access is broken into four 16-bit transfers, so SIZE must be set to 16 bits (0b01).

3 "Don't care" value for this case.

4 32-bit transfers use SIZE 0b00 regardless of DBM because special logic in the EBI handles 32-bit coherent non-chip-select accesses even when DBM = 1. Similar logic is not present for 64-bit transfers, so these are broken into separate bus-size accesses.

22.3.1.2 EBI Transfer Error Status Register (EBI_TESR)

The EBI Transfer Error Status Register contains a bit for each type of transfer error on the external bus. A bit set to logic 1 indicates what type of transfer error occurred since the last time the bits were cleared. Each bit can be cleared by reset or by writing a 1 to it. Writing a 0 has no effect.

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TEAF	BMTF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-2. EBI Transfer Error Status Register (EBI_TESR)

Table 22-7. EBI_TESR field descriptions

Field	Description
TEAF	Transfer Error Acknowledge Flag This bit is set if the cycle was terminated by an externally generated $\overline{\text{TEA}}$ signal. 0 No error. 1 External $\overline{\text{TEA}}$ occurred.
BMTF	Bus Monitor Timeout Flag This bit is set if the cycle was terminated by a bus monitor timeout. 0 No error. 1 Bus monitor timeout occurred.

22.3.1.3 EBI Bus Monitor Control Register (EBI_BMCR)

The EBI Bus Monitor Control Register controls the timeout period of the bus monitor and whether it is enabled or disabled.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BMT								BME	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Figure 22-3. EBI Bus Monitor Control Register (EBI_BMCR)

Table 22-8. EBI_BMCR field descriptions

Field	Description
BMT	Bus Monitor Timing. This field defines the timeout period, in 8 external bus clock resolution, for the Bus Monitor. See Section 22.4.1.6, Bus monitor, for more details on bus monitor operation. Timeout Period = (2 + (8 × BMT)) / external bus clock frequency.
BME	Bus Monitor Enable. This bit controls whether the bus monitor is enabled for internal to external bus cycles. The BME bit is ignored (treated as 0) for chip-select accesses with internal $\overline{\text{TA}}$ (SETA = 0). 0 Disable bus monitor. 1 Enable bus monitor (for external $\overline{\text{TA}}$ accesses only).

22.3.1.4 EBI Base Registers (EBI_BR*n*)

The EBI Base Registers (EBI_BR*n*) are used to define the base address and other attributes for the corresponding chip select.

Address: Base + 0x0010 (EBI_BR0) Base + 0x0020 (EBI_BR2)
Base + 0x0018 (EBI_BR1) Base + 0x0028 (EBI_BR3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BA															
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BA	0	0	LWRN	PS	EOE		SBL	AD_MUX	BL	WEBS	TBDIP	GCSN	SETA	BI	V
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 22-4. EBI Base Registers (EBI_BR*n*)

Table 22-9. EBI_BR*n* field descriptions

Field	Description
BA	Base Address. These bits are compared to the corresponding unmasked address signals among ADDR[0:16] of the internal address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master.
LWRN	Late RD_$\overline{WR}$ Negation. The LWRN bit determines the timing of RD_$\overline{WR}$ signal negation for a write transfer (except for SETA = 1 transfers, where RD_$\overline{WR}$ negates with $\overline{CS}$ by default, so LWRN has no effect). 0 Negate RD_$\overline{WR}$ one cycle earlier than $\overline{CS}$ negation. See Figure 22-15. 1 Negate RD_$\overline{WR}$ the same cycle as $\overline{CS}$ negation. See Figure 22-19.
PS	Port Size. The PS bit determines the data bus width of transactions to this chip-select bank. 0 32-bit port. 1 16-bit port. **Note:** In the case where the DBM bit in EBI_MCR is set for 16-bit Data Bus Mode, the PS bit value is ignored and is always treated as a'1' (16-bit port).
EOE	Early $\overline{OE}$. The EOE field determines the timing of $\overline{OE}$ signal assertion for a read transfer. When EBI_BR[ADMUX] = 1, the EOE value is ignored and treated as 0b00 (in order to avoid contention on shared address/data bus for muxed transfers).

EOE	Timing of $\overline{OE}$ Assertion
00[1]	Assert $\overline{OE}$ 1 CLKOUT cycle after $\overline{TS}$ assertion (see Figure 22-10)
01	Assert $\overline{OE}$ the same CLKOUT cycle as $\overline{TS}$ assertion (see Figure 22-13)
10	Assert $\overline{OE}$ one internal clock cycle after $\overline{TS}$ assertion (see Figure 22-13)[2]
11	Reserved

1 The 0b00 timing case also applies when EBI_BR[ADMUX]=1, regardless of EOE value in Base Register.

2 In divided bus speed modes (where the EBI runs at a slower frequency than the internal system bus), the EBI uses the internal higher-frequency clock to assert $\overline{OE}$ partway through the CLKOUT cycle where $\overline{TS}$ is asserted. In 1:1 bus speed mode (where CLKOUT and internal system bus run at same frequency), the behavior for EOE Value 0b10 is identical to that of value 0b00.

Table 22-9. EBI_BR*n* field descriptions (continued)

Field	Description
SBL	Short Burst Length. The SBL bit provides support for a 2-word external burst (see Figure 22-35 in Section 22.4.2.6.5, Small access example #5: 32-byte read to 32-bit port with SBL = 1). When SBL = 1, the number of beats in a burst is automatically determined by the EBI to be 2 or 4 according to the Port Size (PS bit), regardless of BL bit value. When SBL = 0, the number of beats in a burst is determined by the BL bit. See Table 22-10 under BL bit description for SBL and BL encodings.
AD_MUX	Address on Data Bus Multiplexing. The AD_MUX bit controls whether accesses for this chip select have the address driven on the data bus in the address phase of a cycle. 0 Address on Data Multiplexing Mode is disabled for this chip select. 1 Address on Data Multiplexing Mode is enabled for this chip select.
BL	Burst Length. The BL bit (along with SBL bit) determines the amount of data transferred in a burst for this chip select, measured in 32-bit words. When SBL = 0, the number of beats in a burst is automatically determined by the EBI to be 4, 8, or 16 according to the Port Size (PS bit) so that the burst fetches the number of words chosen by BL. For internal AMBA data bus width of 32-bits, the BL bit is ignored (treated as 1). When SBL = 1, the BL bit value is a don't care. See Table 22-10 for SBL and BL encodings.
WEBS	Write Enable / Byte Select. This bit controls the functionality of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals. 0 The $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals function as $\overline{WE}$[0:3]. 1 The $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals function as $\overline{BE}$[0:3].
TBDIP	Toggle Burst Data in Progress. This bit determines how long the $\overline{BDIP}$ signal is asserted for each data beat in a burst cycle. See Section 22.4.2.5.1, TBDIP effect on burst transfer, for details. 0 Assert $\overline{BDIP}$ throughout the burst cycle, regardless of wait state configuration. 1 Only assert $\overline{BDIP}$ (BSCY+1) external cycles before expecting subsequent burst data beats.
GCSN	Guarantee $\overline{CS}$ Negation. The GCSN bit allows support for guaranteeing that the EBI negates $\overline{CS}$ between all back-to-back transfer cases (even those that are part of a set of Small Accesses). See Figure 22-25. 0 Default operation ($\overline{CS}$ may or may not negate between transfers, depending on the particular back-to-back case). 1 Negate $\overline{CS}$ between all external transfers. This adds an extra dead cycle for some back-to-back cases.
SETA	Select External Transfer Acknowledge. The SETA bit controls whether accesses for this chip select terminate (end transfer without error) based on externally asserted $\overline{TA}$ or internally asserted $\overline{TA}$. SETA should only be set when the BI bit is 1 as well, since burst accesses with SETA = 1 are not supported. Setting SETA = 1 causes the BI bit to be ignored (treated as 1, burst inhibited). 0 Transfer Acknowledge ($\overline{TA}$) is an output from the EBI, data phase is terminated by the EBI. 1 Transfer Acknowledge ($\overline{TA}$) is an input to the EBI, data phase is terminated by an external device.
BI	Burst Inhibit. This bit determines whether or not burst read accesses are allowed for this chip-select bank. The BI bit is ignored (treated as 1) for chip-select accesses with external $\overline{TA}$ (SETA = 1). 0 Enable burst accesses for this bank. 1 Disable burst accesses for this bank. This is the default value out of reset (or when SETA = 1).
V	Valid bit. The user writes this bit to indicate that the contents of this Base Register and Option Register pair are valid. The appropriate $\overline{CS}$ signal does not assert unless the corresponding V-bit is set. 0 This bank is not valid. 1 This bank is valid.

Table 22-10. SBL, BL Values

Value		Burst Length[1]	PS	# Beats in Burst[2]
SBL	BL			
0	0[3]	8-word[4]	0 (32-bit)	8
			1 (16-bit)	16
0	1	4-word	0 (32-bit)	4
			1 (16-bit)	8
1	X	2 word[4]	0 (32-bit)	2
			1 (16-bit)	4

1 Total amount of data fetched in a burst transfer, measured in 32-bit words.

2 Number of external data beats used in external burst transfer. The size of each beat is determined by PS value.

3 An 8-word burst length is only supported for devices using 64-bit AMBA data bus width to EBI.

4 A word always refers to 32 bits of data, regardless of PS.

22.3.1.5 EBI Option Registers (EBI_OR*n*)

The EBI Option Registers (EBI_OR*n*) are used to define the address mask and other attributes for the corresponding chip select.

Address: Base + 0x0014 (EBI_OR0) Base + 0x0024 (EBI_OR2)
Base + 0x001C (EBI_OR1) Base + 0x002C (EBI_OR3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	AM															
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AM	0	0	0	0	0	0	0	SCY				0	BSCY		0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-5. EBI Option Registers (EBI_OR*n*)

Table 22-11. EBI_OR*n* field descriptions

Field	Description
AM	Address Mask. This field allows masking of any corresponding bits in the associated Base Register. Masking the address independently allows external devices of different size address ranges to be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in comparison with the address pins. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time.

Table 22-11. EBI_OR*n* field descriptions (continued)

Field	Description
SCY	Cycle length in clocks. This field represents the number of wait states (external cycles) inserted after the address phase in the single transfer case, or in the first beat of a burst, when the memory controller handles the external memory access. Values range from 0 to 15. This is the main parameter for determining the length of the cycle. These bits are ignored when SETA = 1. The total cycle length for the first beat (including the $\overline{TS}$ cycle) = (2 + SCY) external clock cycles. See Section 22.5.3.1, Example wait state calculation, for related application information.
BSCY	Burst beats length in clocks. This field determines the number of wait states (external cycles) inserted in all burst beats except the first, when the memory controller starts handling the external memory access and thus is using SCY[0:3] to determine the length of the first beat. These bits are ignored when SETA = 1. The total memory access length for each beat is (1 + BSCY) external clock cycles. The total cycle length (including the TS cycle) = (2 + SCY) + (#beats – 1) × (BSCY + 1) where #beats is the number of beats (4,8,16) determined by BL and PS bits in Base Register.

BSCY	Meaning
00	0-clock cycle wait states (1 clock per data beat)
01	1-clock cycle wait states (2 clocks per data beat)
10	2-clock cycle wait states (3 clocks per data beat)
11	3-clock cycle wait states (4 clocks per data beat)

22.4 Functional description

22.4.1 EBI features

22.4.1.1 32-bit address bus with transfer size indication (up to 24 available on pins)

The transfer size for an external transaction is indicated by the TSIZ[0:1] signals during the clock where address is valid. Valid transaction sizes are 8, 16, and 32 bits. Only 24 address lines are pinned out externally (device-specific option), but a full 32-bit decode is done internally to determine the target of the transaction and whether a chip select should be asserted.

22.4.1.2 16-bit data bus

There is a 16-bit data bus mode available via the DBM bit in EBI_MCR. See Section 22.1.3.5, 16-bit data bus mode.

22.4.1.3 Multiplexed address on data pins (single master)

When this mode is enabled, the address shows up on the data pins during the address phase of the cycle. This mode can be enabled separately for non-chip-select accesses and per-chip-select access. See Section 22.1.3.6, Multiplexed address on data bus mode.

22.4.1.4 Memory controller with support for various memory types

The EBI contains a memory controller that supports a variety of memory types, including synchronous burst mode flash and SRAM, and asynchronous/legacy flash and SRAM with a compatible interface.

Each $\overline{CS}$ bank is configured via its own pair of Base and Option Registers. Each time an internal to external bus cycle access is requested, the internal address is compared with the base address of each valid Base Register (with 17 bits having mask). See Figure 22-6. If a match is found, the attributes defined for this bank in its BR and OR are used to control the memory access. If a match is found in more than one bank, the lowest bank matched handles the memory access (for example, bank 0 is selected over bank 1).

A match on a valid calibration chip-select register overrides a match on any non-calibration chip-select register, with CAL_CS0 having the highest priority. Thus the full priority of the chip-selects is: CAL_CS0,...,CAL_CS3,CS0,...,CS3

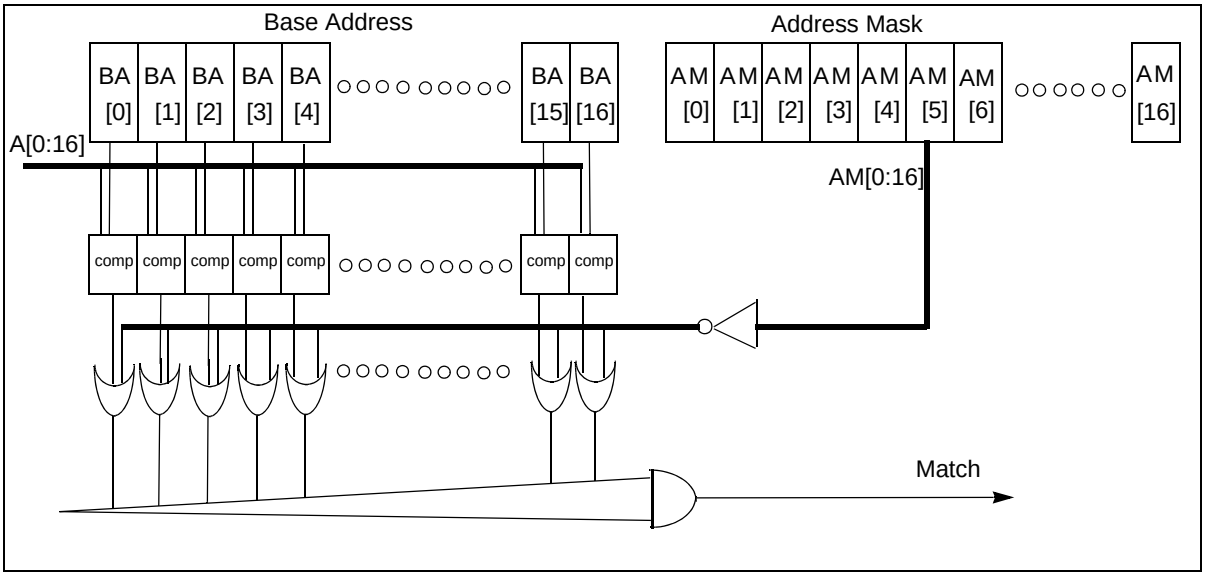


Figure 22-6. Bank base address and match structure

When a match is found on one of the chip-select banks, all its attributes (from the appropriate Base and Option Registers) are selected for the functional operation of the external memory access, such as:

- Number of wait states for a single memory access, and for any beat in a burst access
- Burst enable
- Port size for the external accessed device

See Section 22.3.1.4, EBI Base Registers (EBI_BRn), and Section 22.3.1.5, EBI Option Registers (EBI_ORn), for a full description of all chip-select attributes.

When no match is found on any of the chip-select banks, the default transfer attributes shown in Table 22-12 are used.

Table 22-12. Default attributes for non-chip-select transfers

CS Attribute	Default Value	Comment
PS	0	32-bit port size
BL	0	Burst length is don't care since burst is disabled
WEBS	0	Write enables
TBDIP	0	Don't care since burst is disabled
BI	1	Burst inhibited
SCY	0	Don't care since external $\overline{TA}$ is used
BSCY	0	Don't care since external $\overline{TA}$ is used
AD_MUX	0	Address on Data multiplexing
SETA	1	Select external TA to terminate access

22.4.1.5 Burst support (wrapped only)

The EBI supports burst read accesses of external burstable memory. To enable bursts to a particular memory region, clear the BI (Burst Inhibit) bit in the appropriate Base Register. External burst lengths of 2, 4, and 8 words are supported. Burst length is configured for each chip select by using the BL and SBL bits in the appropriate Base Register. See Section 22.4.2.5, Burst transfer, for more details on burst operation.

In 16-bit data bus mode (DBM = 1 in EBI_MCR), a special 2-beat burst case is supported for reads and writes for 32-bit non-chip-select accesses only. This is to allow 32-bit coherent accesses to another MCU. See Section 22.4.2.9, Non-chip-select burst in 16-bit data bus mode.

Bursting of accesses that are not controlled by the chip selects is not supported for any other case besides the special case of 32-bit accesses in 16-bit data bus mode.

Burst writes are not supported for any other case besides the special case of 32-bit non-chip-select writes in 16-bit data bus mode. Internal requests to write >32 bits (such as a cache line) externally are broken up into separate 32-bit or 16-bit external transactions according to the port size. See Section 22.4.2.6, Small accesses (small port size and short burst length), for more detail on these cases.

22.4.1.6 Bus monitor

When enabled (via the BME bit in the EBI_BMCR), the bus monitor detects when no $\overline{TA}$ assertion is received within a maximum timeout period for external $\overline{TA}$ accesses. The timeout for the bus monitor is specified by the BMT field in the EBI_BMCR. Each time a timeout error occurs, the BMTF bit is set in the EBI_TESR. The timeout period is measured in external bus (CLKOUT) cycles. Thus the effective real-time period is multiplied (by 2, 3, etc.) when a slower-speed mode is used, even though the BMT field itself is unchanged.

22.4.1.7 Port size configuration per chip select (16 or 32 bits)

The EBI supports memories with data widths of 16 or 32 bits. The port size for a particular chip select is configured by writing the PS bit in the corresponding Base Register.

22.4.1.8 Configurable wait states

From 0 to 15 wait states can be programmed for any cycle that the memory controller generates, via the SCY bits in the appropriate Option Register. From 0 to 3 wait states between burst beats can be programmed using the BSCY bits in the appropriate Option Register.

22.4.1.9 Configurable internal or external $\overline{TA}$ per chip select

Each chip select can be configured (via the SETA bit) to have $\overline{TA}$ driven internally (by the EBI), or externally (by an external device). See Section 22.3.1.4, EBI Base Registers (EBI_BRn), for more details on SETA bit usage.

22.4.1.10 Four Write/Byte Enable ($\overline{WE}$/$\overline{BE}$) signals

The functionality of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals depends on the value of the WEBS bit in the corresponding Base Register. Setting WEBS to 1 configures these pins as $\overline{BE}$[0:3], while resetting it to 0 configures them as $\overline{WE}$[0:3]. $\overline{WE}$[0:3] are asserted only during write accesses, while $\overline{BE}$[0:3] is asserted for both read and write accesses. The timing of the $\overline{WE}$[0:3]/$\overline{BE}$[0:3] signals remains the same in either case.

The upper Write/Byte Enable ($\overline{WE0}$/$\overline{BE0}$) indicates that the upper 8 bits of the data bus (DATA[0:7]) contain valid data during a write/read cycle. The upper middle Write/Byte Enable ($\overline{WE1}$/$\overline{BE1}$) indicates that the upper middle 8 bits of the data bus (DATA[8:15]) contain valid data during a write/read cycle. The lower middle Write/Byte Enable ($\overline{WE2}$/$\overline{BE2}$) indicates that the lower middle 8 bits of the data bus (DATA[16:23]) contain valid data during a write/read cycle. The lower Write/Byte Enable ($\overline{WE3}$/$\overline{BE3}$) indicates that the lower 8 bits of the data bus (DATA[24:31]) contain valid data during a write/read cycle.

NOTE

The exception to the preceding $\overline{WE}$/$\overline{BE}$ description is that for 16-bit port transfers (DBM = 1 or PS = 1), only the $\overline{WE}$[0:1]/$\overline{BE}$[0:1] signals are used, regardless of whether DATA[0:15] or DATA[16:31] are selected (via the D16_31 bit in the EBI_MCR). This means for the case where DATA[16:31] are selected, that $\overline{WE0}$ indicates that DATA[16:23] contains valid data, and $\overline{WE1}$ indicates that DATA[24:31] contains valid data.

The Write/Byte Enable lines affected in a transaction for a 32-bit port (PS = 0) and a 16-bit port (PS = 1) are shown in Table 22-13. Only big-endian byte ordering is supported by the EBI.

Table 22-13. Write/Byte Enable signals function [1]

Transfer Size	TSIZ[0:1]	Address		32-bit port size				16-bit port size[2]			
		A30	A31	WE0/BE0	WE1/BE1	WE2/BE2	WE3/BE3	WE0/BE0	WE1/BE1	WE2/BE2	WE3/BE3
Byte	01	0	0	X				X			
	01	0	1		X				X		
	01	1	0			X		X			
	01	1	1				X		X		
16-bit	10	0	0	X	X			X	X		
	10	1	0			X	X	X	X		
32-bit	00	0	0	X	X	X	X	X[3]	X[3]		
Burst	00	0	0	X	X	X	X	X	X		

1 This table applies to aligned internal master transfers only. In the case of a misaligned internal master transfer that is split into multiple aligned external transfers, not all of the write enables X'd in the table will necessarily assert. See Section 22.4.2.10, Misaligned access support.

2 Also applies when DBM = 1 for 16-bit data bus mode.

3 This case consists of two 16-bit external transactions, but for both transactions the $\overline{WE}$[0:1]/$\overline{BE}$[0:1] signals are the only $\overline{WE}$/$\overline{BE}$ signals affected.

22.4.1.11 Slower-speed clock modes

For memories that cannot run with a full-speed external bus, the EBI supports slower-speed clock modes. Refer to Section 22.1.3.4, Slower-speed modes, for more details on this feature. The timing diagrams for slower-speed modes are identical to those for full-speed mode, except that the frequency of CLKOUT is reduced.

22.4.1.12 Address Latch Enable (ALE)

The Address Latch Enable (ALE) indicates when the address is present during a multiplexed bus access. This can be used in conjunction with an external latch to hold the state of the address access if connecting the MCU to a non-multiplexed bus compatible memory. ALE signal timing is shown in Figure 22-7.

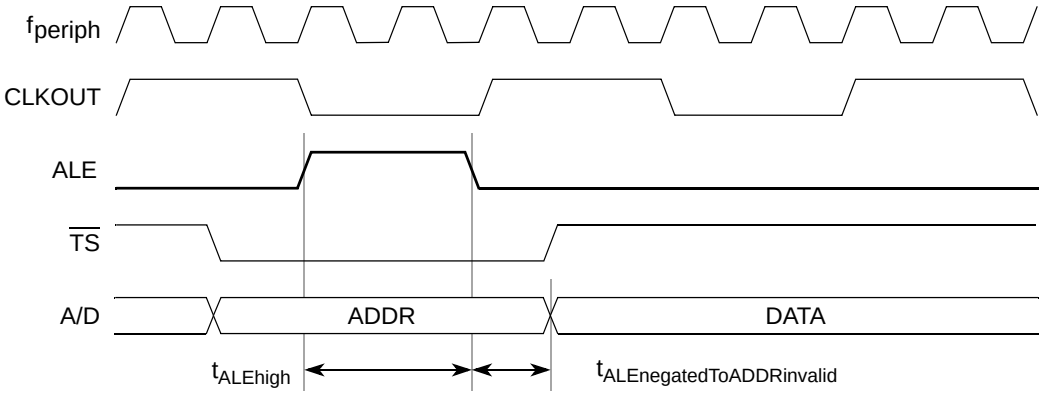


Figure 22-7. ALE signal timing

There are at least two timing requirements that relate to external components for multiplexed addr/data bus in systems using ALE. These are not EBI specifications. The two are

- ALE — minimum high time
- ALE negated to ADDR invalid

These values depend on maximum clkout frequency and the actual clock tree insertion on the ALE clock gate.

22.4.1.13 Stop and module disable modes for power savings

See Section 22.1.3, Modes of operation, for a description of the power saving modes.

22.4.1.14 Optional automatic CLKOUT gating

The EBI has the ability to hold the external CLKOUT pin high when the EBI's internal master state machine is idle and no requests are pending. The EBI outputs a signal to the pads logic in the MCU to disable CLKOUT. This feature is disabled out of reset, and can be enabled or disabled by the ACGE bit in the EBI_MCR.

NOTE

This feature must be disabled for multi-master systems. In those cases, one master is getting its clock source from the other master and needs it to stay valid continuously.

22.4.1.15 Misaligned access support

The EBI has limited misaligned access support. Misaligned non-burst chip-select transfers from internal masters are supported. The EBI aligns the accesses when it sends them out to the external bus (splitting them into multiple aligned accesses if necessary), so that external devices are not required to support misaligned accesses. Burst accesses (internal master) must match the internal bus size (64-bit aligned). See Section 22.4.2.10, Misaligned access support, for more details.

22.4.1.16 Compatible with MPC5xx external bus (with some limitations)

The EBI is compatible with the external bus of the MPC5xx parts, meaning that it supports most devices supported by the MPC5xx family of parts. However, there are some differences between this EBI and that of the MPC5xx parts that the user needs to be aware of before assuming that an MPC5xx-compatible device works with this EBI. See Section 22.5.6, Summary of differences from MPC5xx, for details.

NOTE

Due to testing and complexity concerns, multi-master (or master/slave) operation between an eSys MCU and MPC5xx is not guaranteed.

22.4.2 External bus operations

The following sections provide a functional description of the external bus, the bus cycles provided for data transfer operations, and error conditions.

22.4.2.1 External clocking

The CLKOUT signal sets the frequency of operation for the bus interface directly. Internally, the MPC5675K uses a phase-locked loop (PLL) circuit to generate a master clock for all of its circuitry (including the EBI) that is phase-locked to the CLKOUT signal. In general, all signals for the EBI are specified with respect to the rising edge of the CLKOUT signal, and they are configured to be sampled as inputs or changed as outputs with respect to that edge.

22.4.2.2 Reset

Upon detection of internal reset assertion, the EBI immediately ends all transactions (abruptly, not through normal termination protocol), and ignores any transaction requests that take place while reset is asserted.

22.4.2.3 Basic transfer protocol

The basic transfer protocol defines the sequence of actions that must occur on the external bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is shown in Figure 22-8.

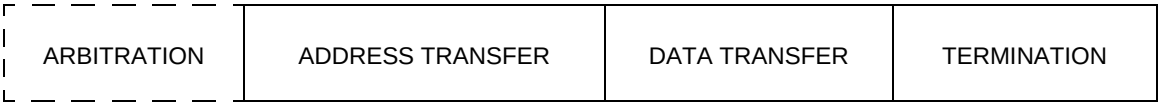


Figure 22-8. Basic transfer protocol

The arbitration phase is where bus ownership is requested and granted. This phase is not needed in Single Master Mode because the EBI is the permanent bus owner in this mode.

The address transfer phase specifies the address for the transaction and the transfer attributes that describe the transaction. The signals related to the address transfer phase are $\overline{TS}$, ADDR (or DATA if Address/Data multiplexing is used), $\overline{CS}$[0:3], RD_$\overline{WR}$, TSIZ[0:1], and $\overline{BDIP}$. The address and its related signals (with the exception of $\overline{TS}$, $\overline{BDIP}$) are driven on the bus with the assertion of the $\overline{TS}$ signal, and kept valid until

the bus master receives $\overline{TA}$ asserted (the EBI holds them one cycle beyond $\overline{TA}$ for writes and external $\overline{TA}$ accesses). Note that for writes with internal $\overline{TA}$, RD_$\overline{WR}$ is not held one cycle past $\overline{TA}$.

The data transfer phase performs the transfer of data, from master to slave (in write cycles) or from slave to master (on read cycles), if any is to be transferred. The data phase may transfer a single beat of data (1-4 bytes) for non-burst operations or a 2-beat, 4-beat, 8-beat, or 16-beat burst of data (2 or 4 bytes per beat depending on Port Size) when burst is enabled. On a write cycle, the master must not drive write data until after the address transfer phase is complete. This is to avoid electrical contentions when switching between drivers. The master must start driving write data one cycle after the address transfer cycle. The master can stop driving the data bus as soon as it samples the $\overline{TA}$ line asserted on the rising edge of CLKOUT. To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where $\overline{RD\_WR}$ and $\overline{WE}$ are negated (for chip-select accesses only). See Figure 22-15 for an example of write timing. On a read cycle, the master accepts the data bus contents as valid on the rising edge of the CLKOUT in which the $\overline{TA}$ signal is sampled asserted. See Figure 22-10 for an example of read timing.

The termination phase is where the cycle is terminated by the assertion of either $\overline{TA}$ (normal termination) or $\overline{TEA}$ (termination with error). Termination is discussed in detail in Section 22.4.2.8, Termination signals protocol.

NOTE

In the timing diagrams in this document, asynchronous relationships between signals that switch in the same CLKOUT cycle are not guaranteed. For example, in Figure 22-15, $\overline{WE}$ and write DATA change during the same CLKOUT cycle. There is no guarantee that DATA will be stable before $\overline{WE}$ assertion. External devices should not be latching write DATA on $\overline{WE}$ assertion, but instead must use a signal edge that takes place in a later CLKOUT cycle, such as $\overline{WE}$ negation.

22.4.2.4 Single beat transfer

The flow and timing diagrams in this section assume that the EBI is configured in Single Master Mode. Therefore, arbitration is not needed and is not shown in these diagrams.

22.4.2.4.1 Single beat read flow

The handshakes for a single beat read cycle are illustrated in the following flow and timing diagrams.

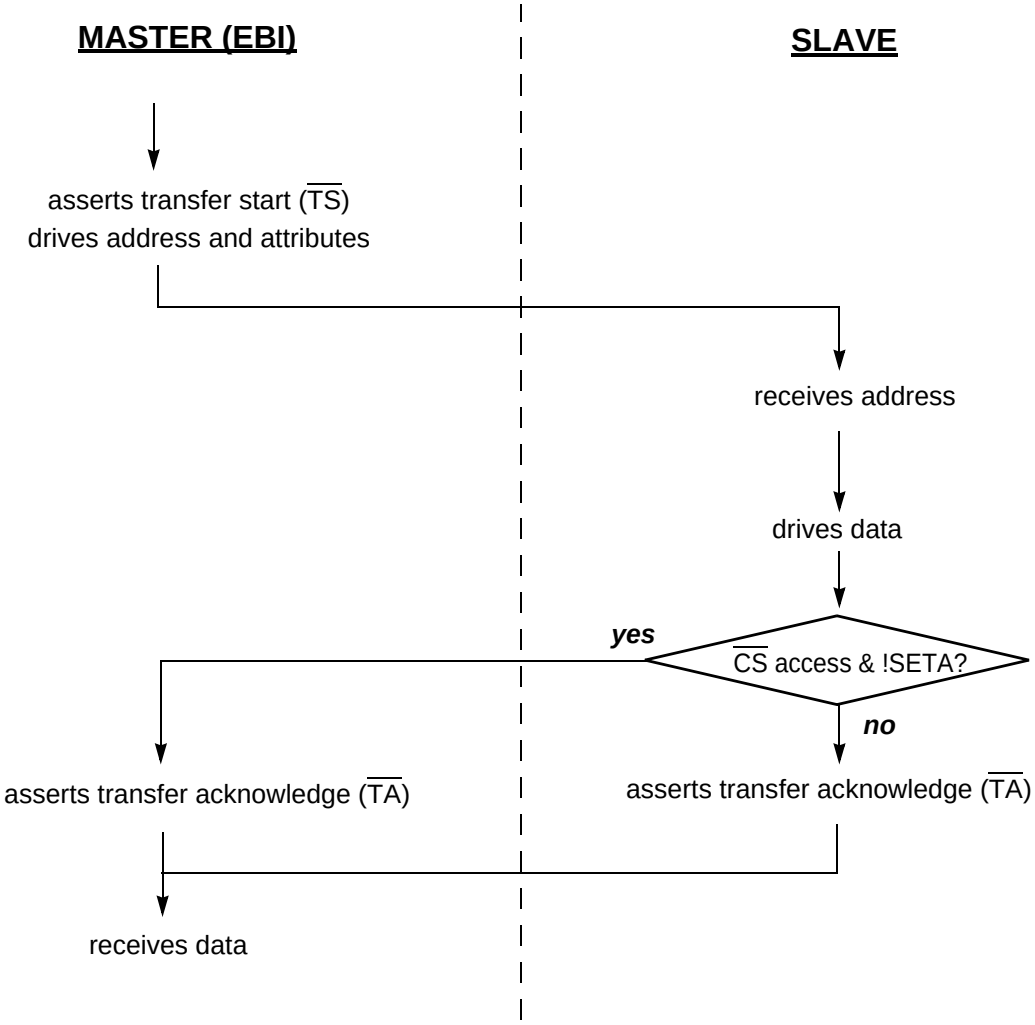


Figure 22-9. Basic flow diagram of a single beat read cycle

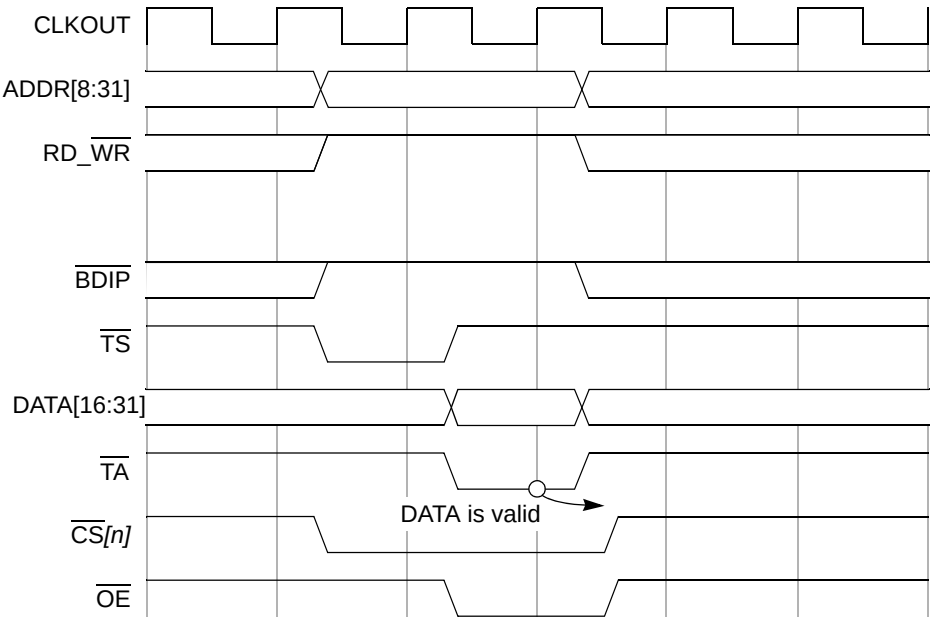


Figure 22-10. Single beat 32-bit read cycle, CS access, zero wait states

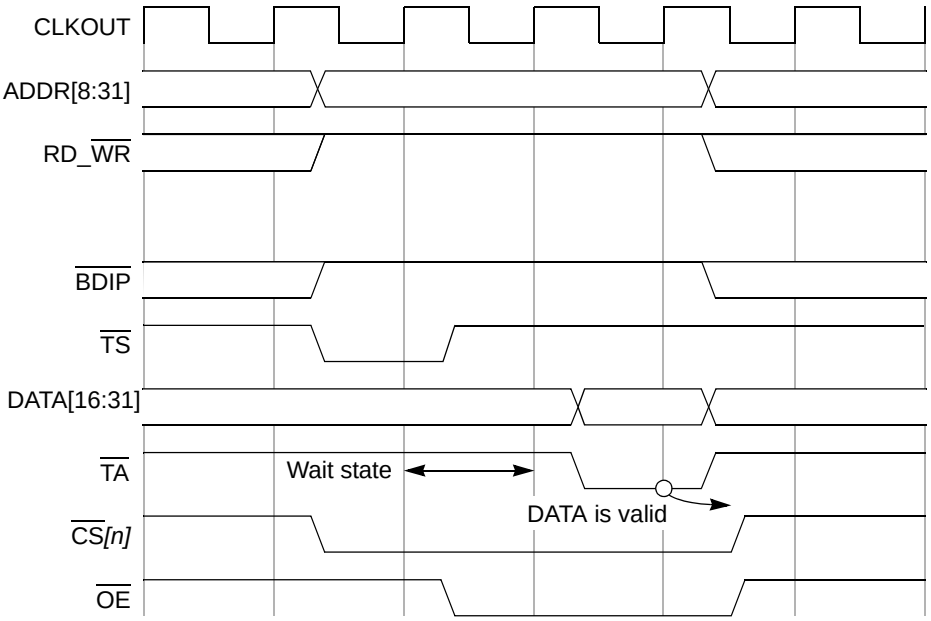
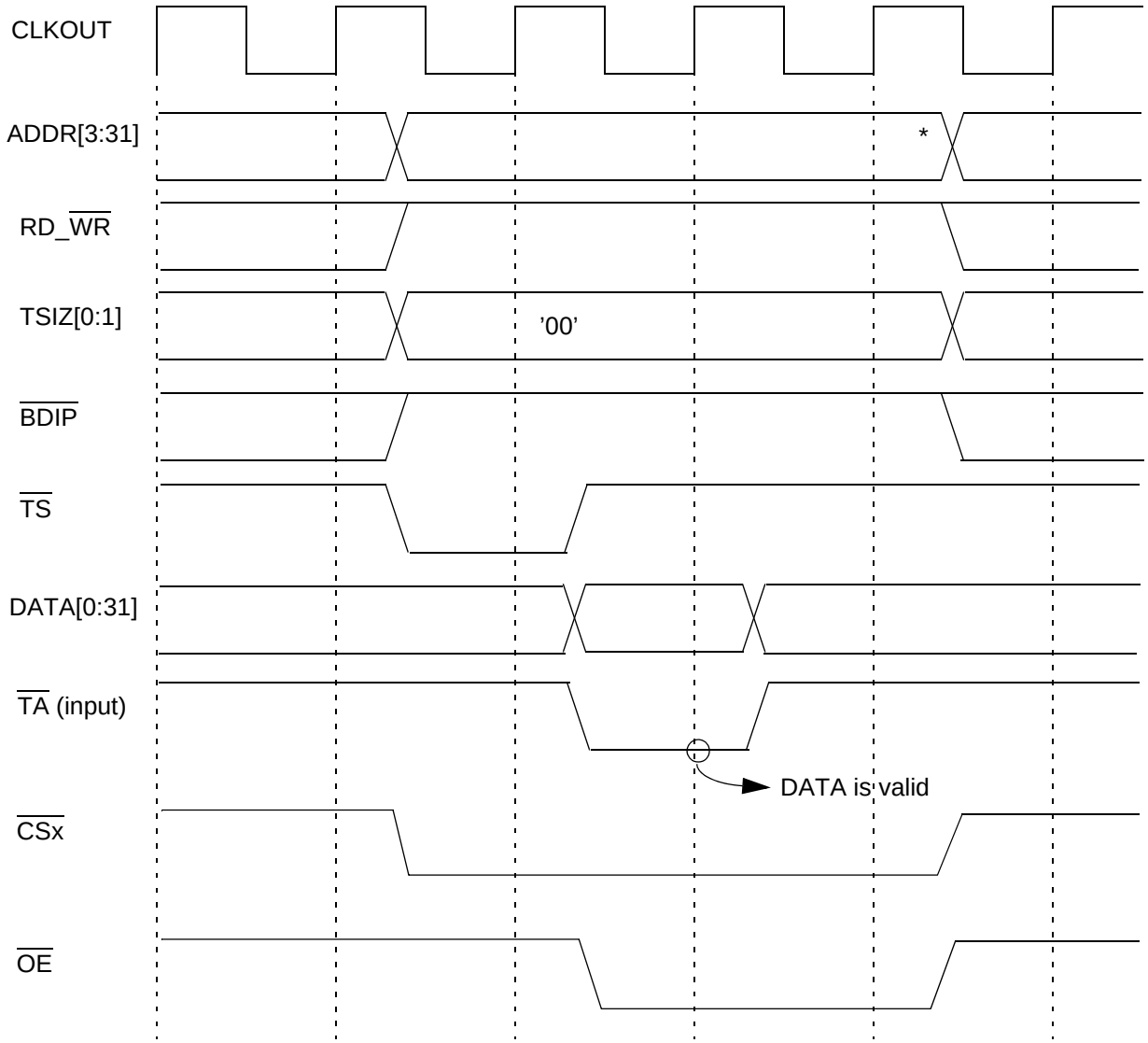


Figure 22-11. Single beat 32-bit read cycle, CS access, one wait state



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 22-12. Single beat 32-bit read cycle, CS access, SETA = 1, zero wait states

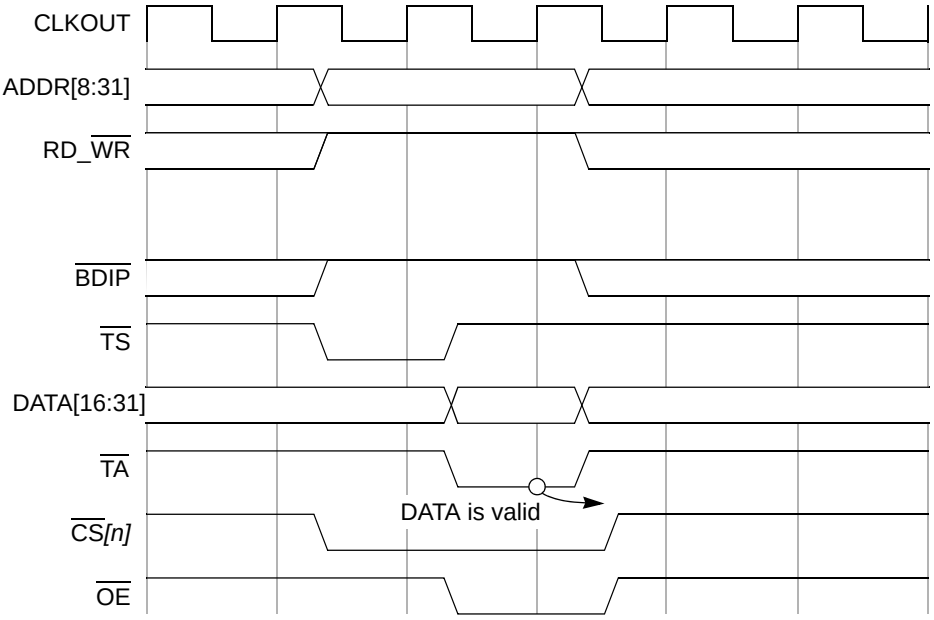


Figure 22-13. Single beat 32-bit read cycle, CS access, zero wait states (EOE = 0b01, 0b10)

22.4.2.4.2 Single beat write flow

The handshakes for a single beat write cycle are illustrated in the following flow and timing diagrams.

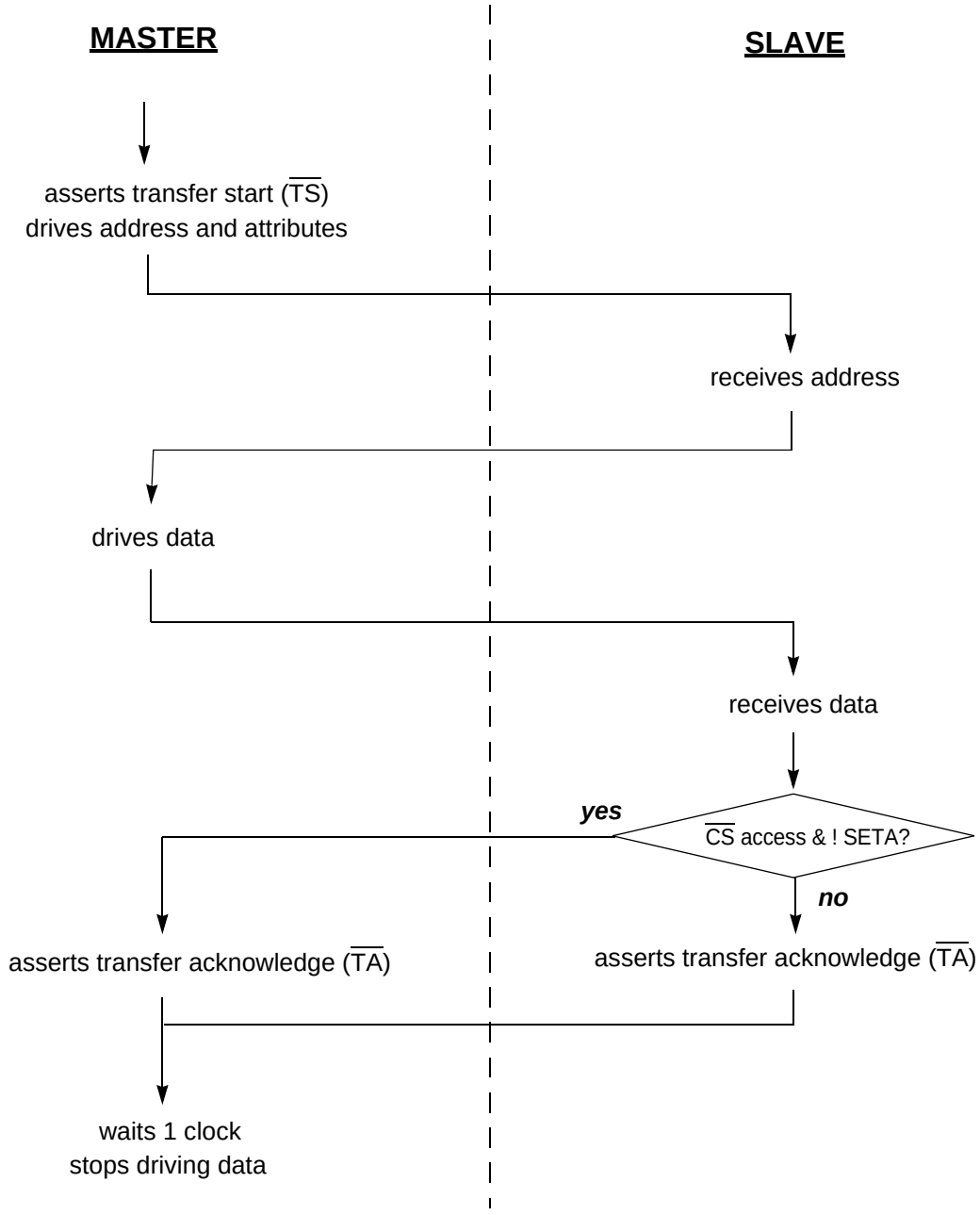


Figure 22-14. Basic flow diagram of a single beat write cycle

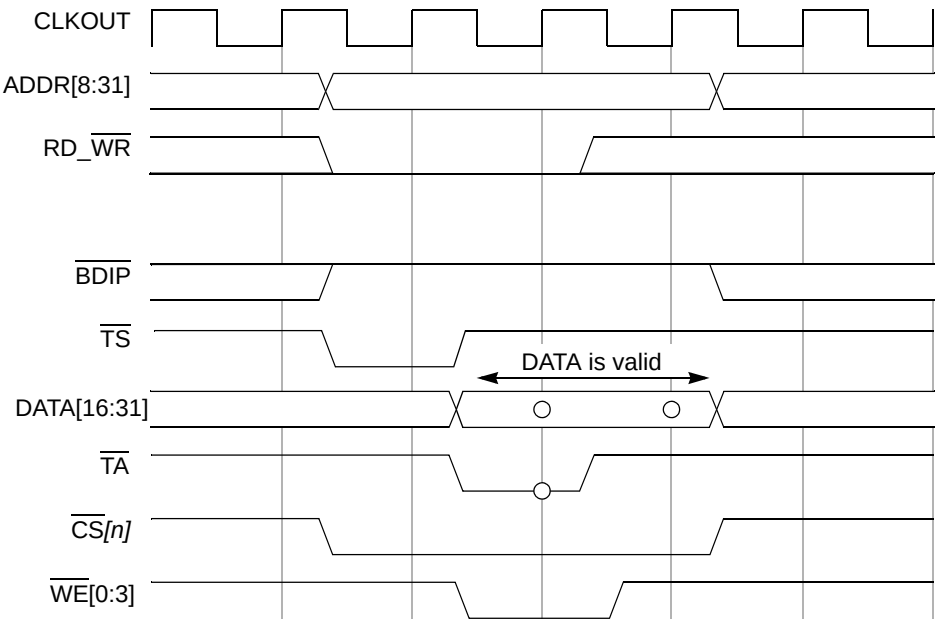


Figure 22-15. Single beat 32-bit write cycle, CS access, zero wait states

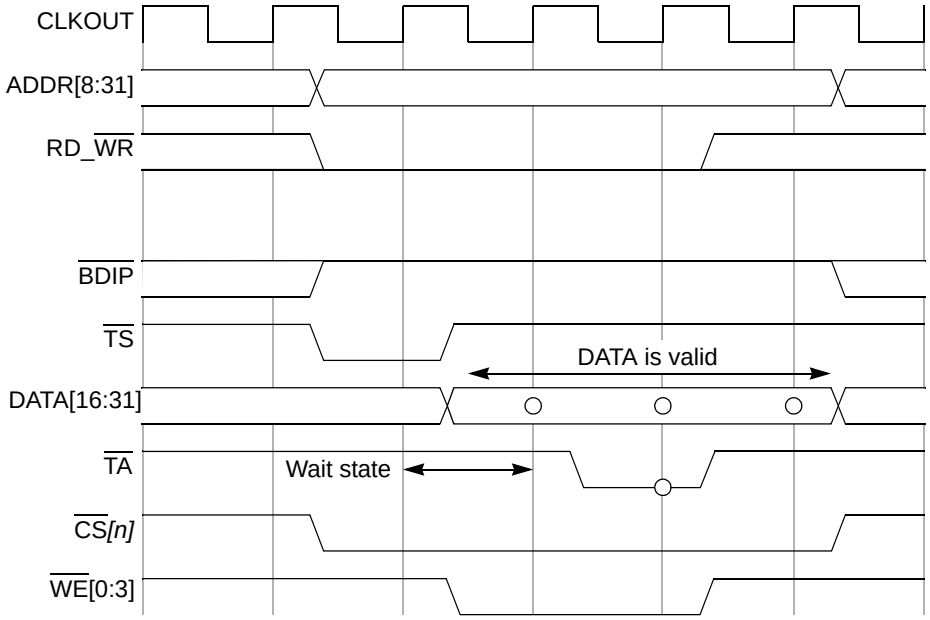
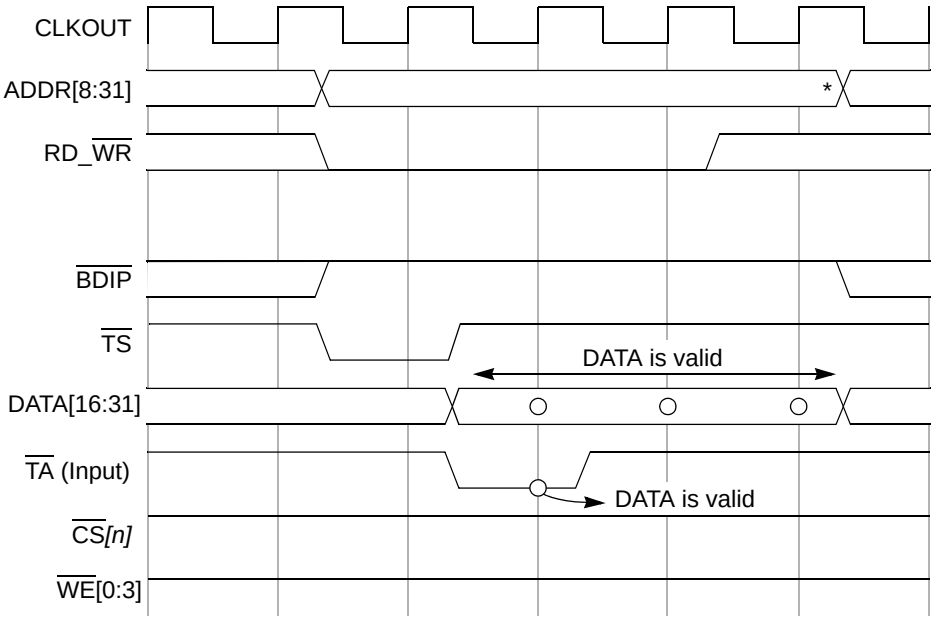
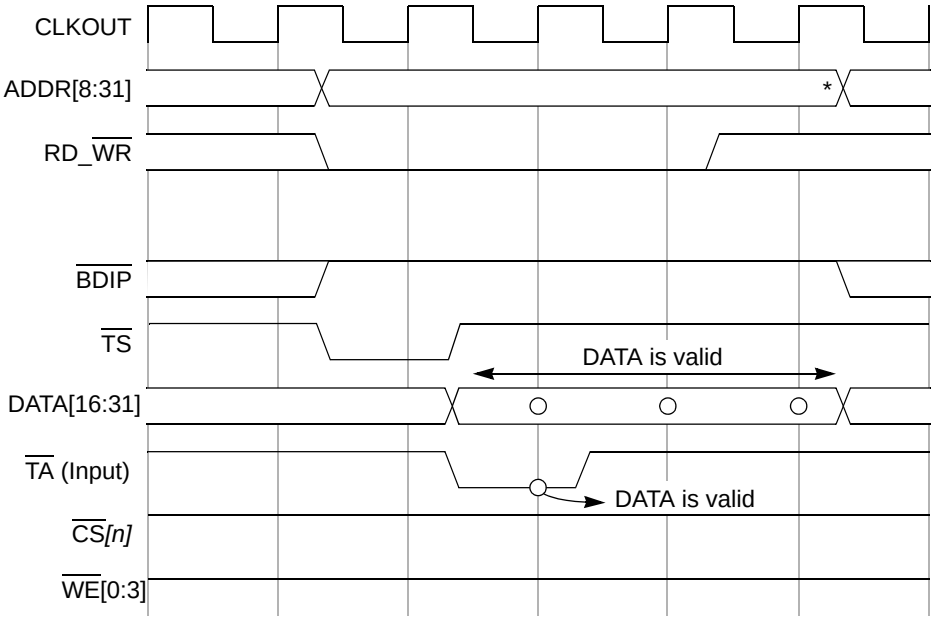


Figure 22-16. Single beat 32-bit write cycle, CS access, one wait state



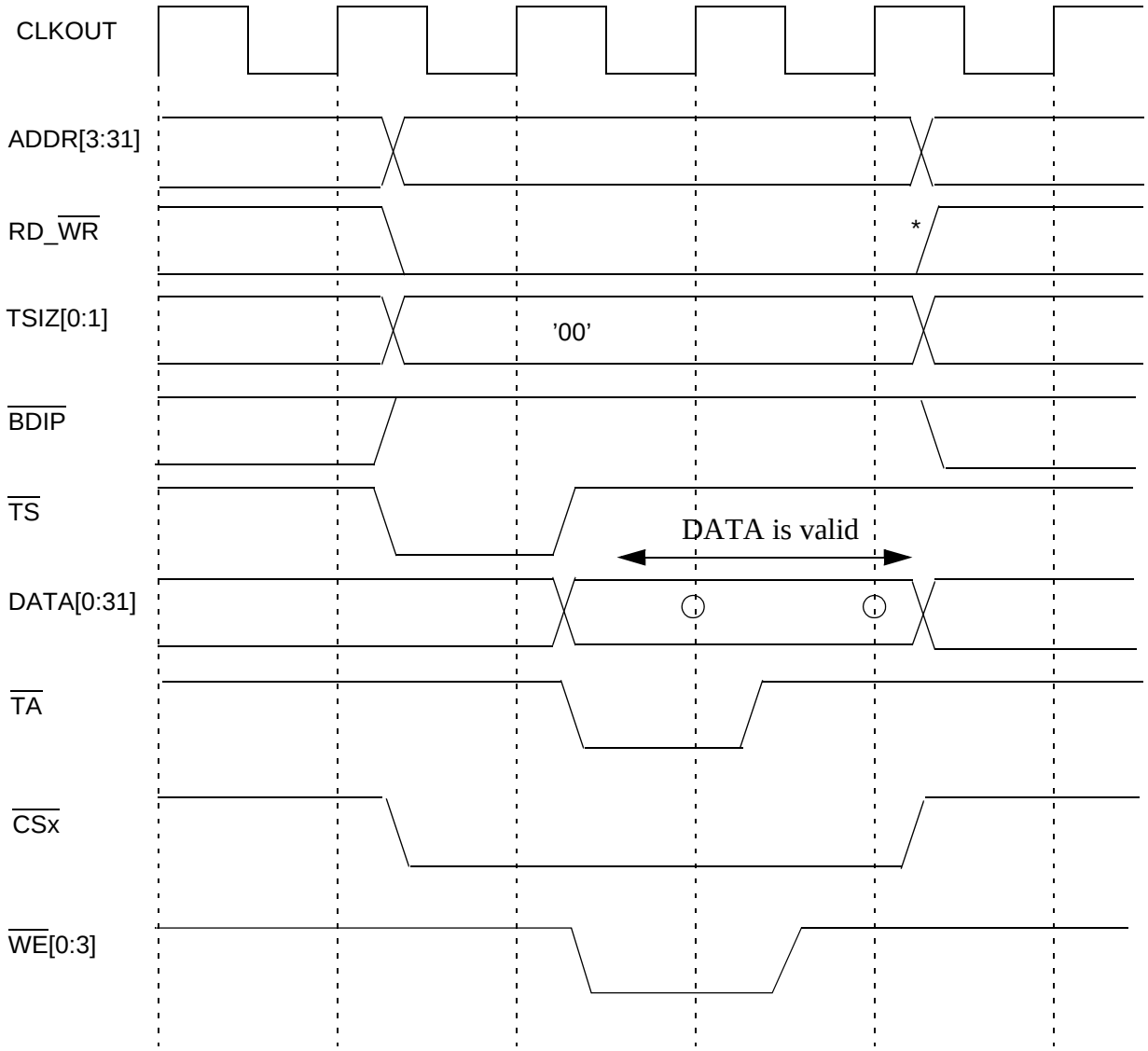
* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 22-17. Single beat 32-bit write cycle, CS access, SETA = 1, zero wait states



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 22-18. Single beat 32-bit write cycle, non-CS access, zero wait states



* Negation of RD_WR signal is delayed by 1 cycle (as shown) when EBI_BR[LWRN]=1.

Figure 22-19. Single beat 32-bit write cycle, CS access, zero wait states, LWRN = 1

22.4.2.4.3 Back-to-back accesses

Due to internal bus protocol, one dead cycle is necessary between back-to-back external bus accesses that are not part of a set of small accesses (see Section 22.4.2.6, Small accesses (small port size and short burst length), for small access timing). A dead cycle refers to a cycle between the TA of a previous transfer and the TS of the next transfer.

NOTE

In some cases, $\overline{CS}$ remains asserted during this dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select (with EBI_BR[GCSN] = 0). See Figure 22-23 and Figure 22-24. However, if EBI_BR[GCSN] = 1 (see Figure 22-25), then the EBI inserts an extra dead cycle between the accesses for these cases in order to negate $\overline{CS}$.

Besides this dead cycle, in most cases, back-to-back accesses on the external bus do not cause any change in the timing from that shown in the previous diagrams, and the two transactions are independent of each other. The only exceptions to this are listed below:

- Back-to-back accesses where the first access ends with an externally driven $\overline{TA}$ or $\overline{TEA}$. In these cases, an extra cycle is required between the end of the first access and the $\overline{TS}$ assertion of the second access. See Section 22.4.2.8, Termination signals protocol, for more details.

The following diagrams show a few examples of back-to-back accesses on the external bus.

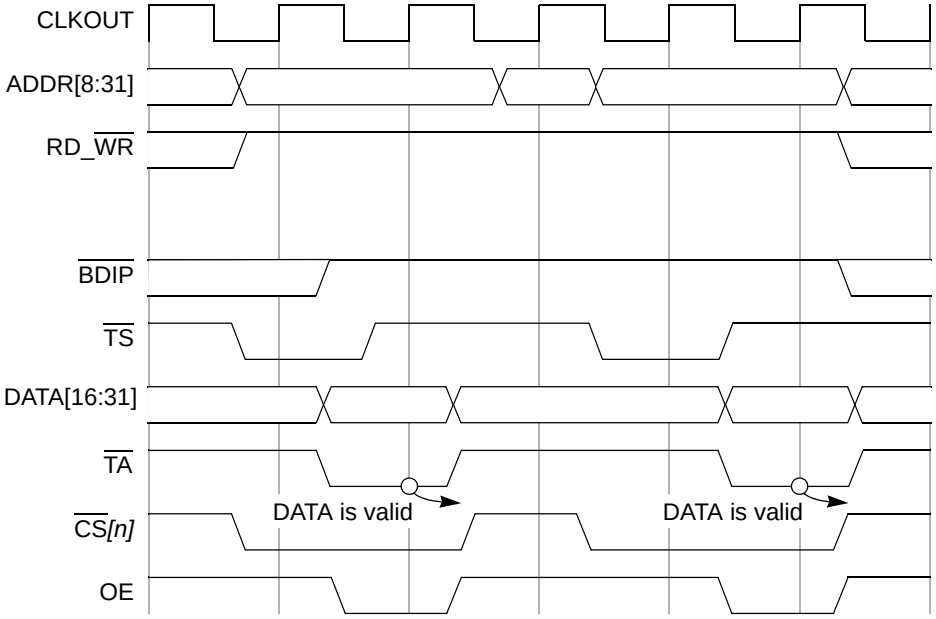


Figure 22-20. Back-to-back 32-bit reads to the same $\overline{CS}$ bank

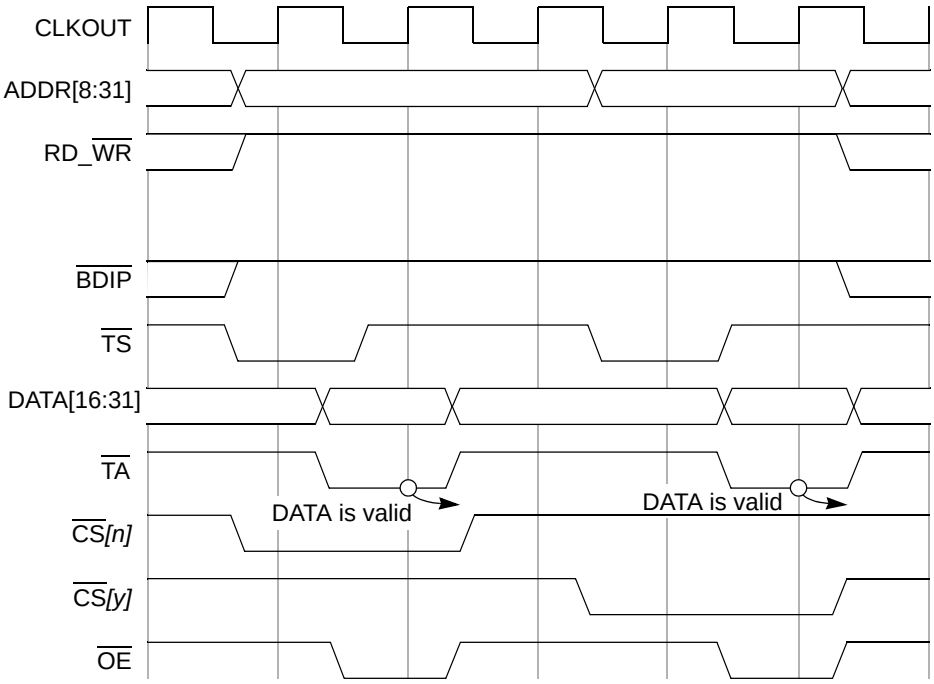


Figure 22-21. Back-to-back 32-bit reads to different CS banks

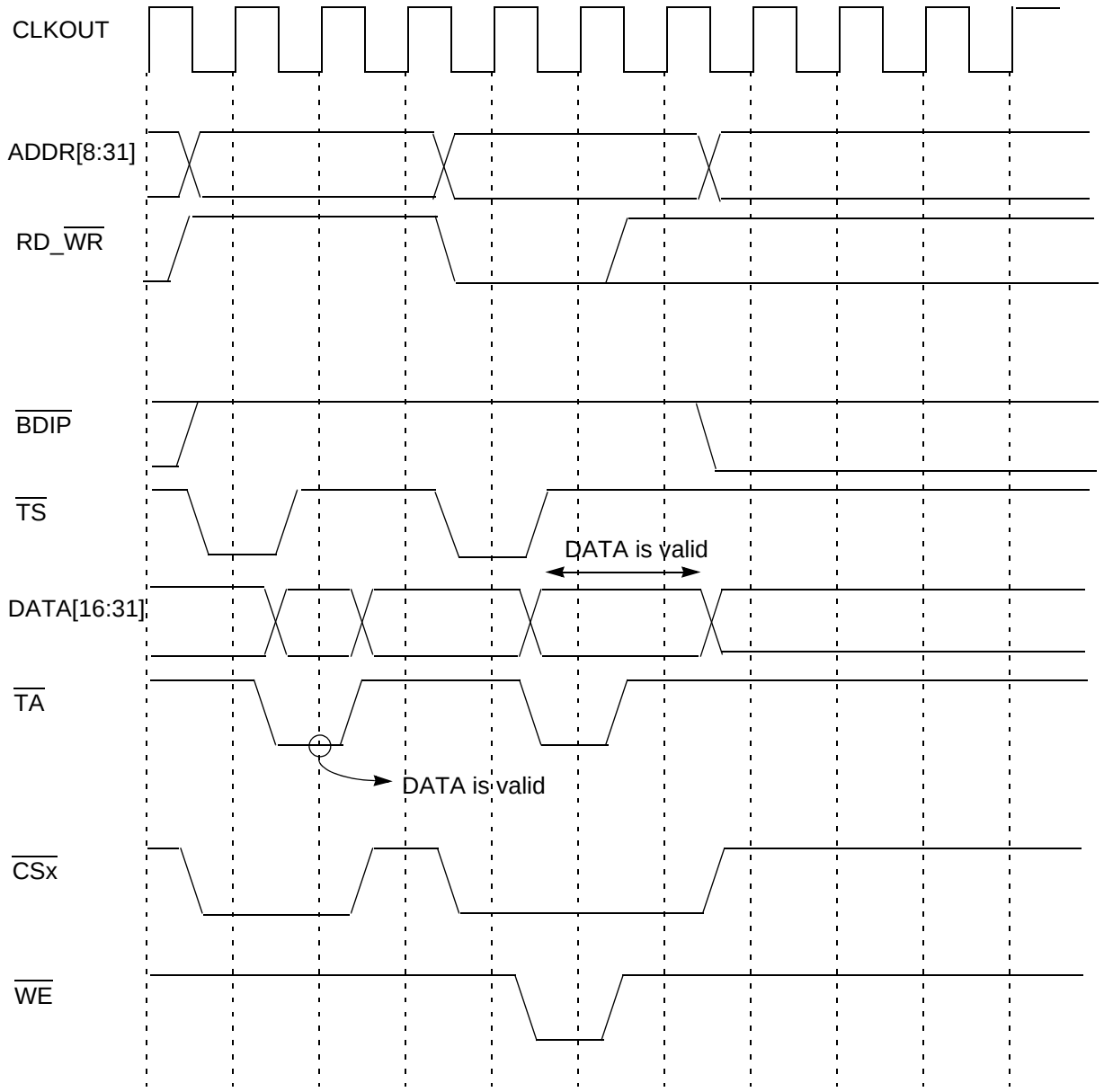


Figure 22-22. Write after read to the same CS bank

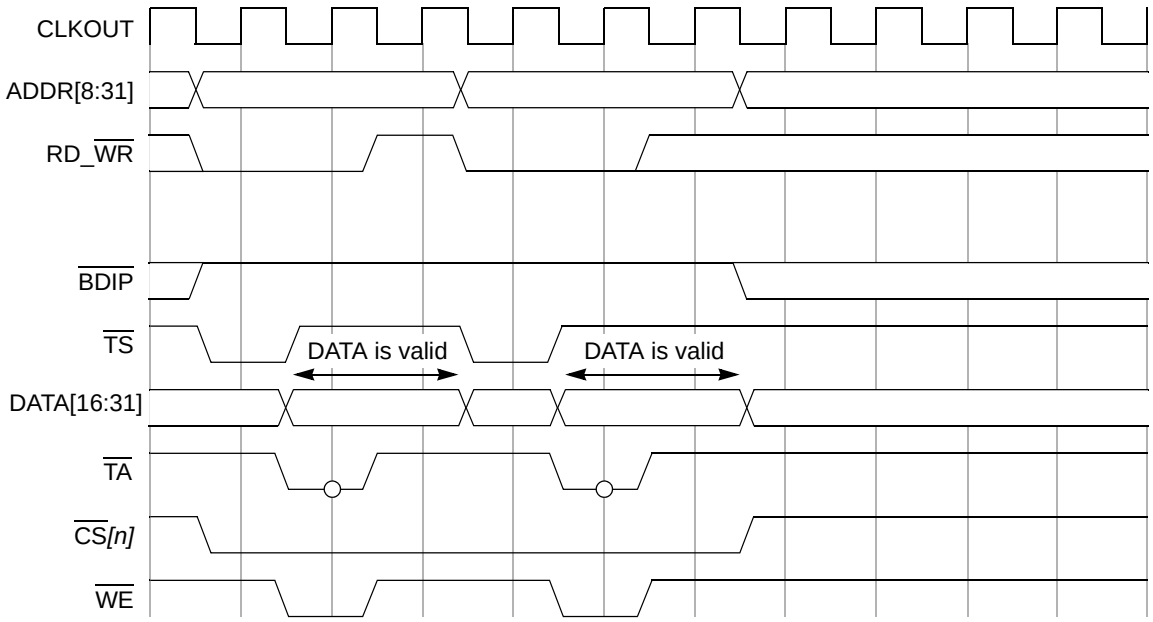


Figure 22-23. Back-to-back 32-bit writes to the same CS bank

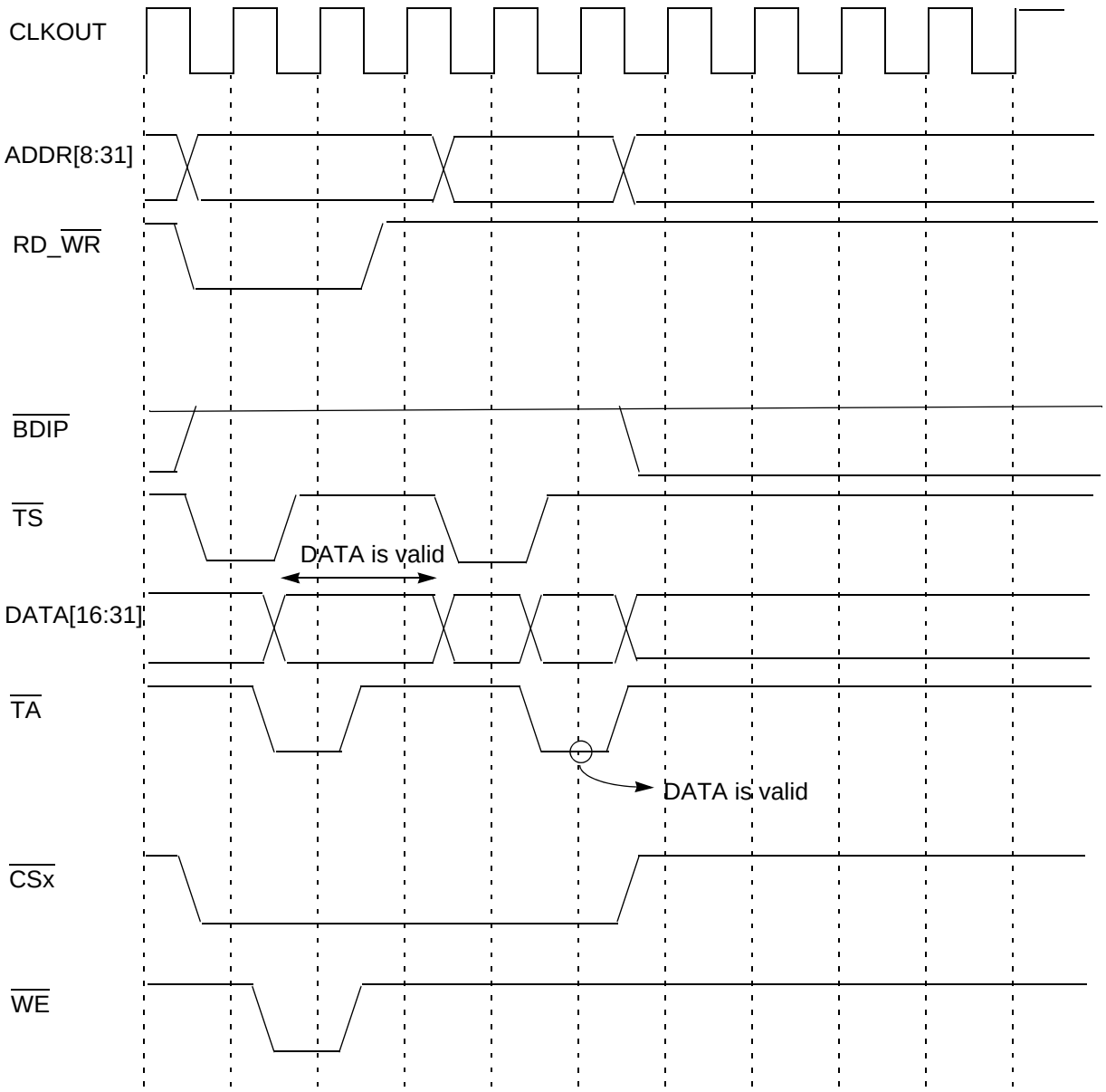
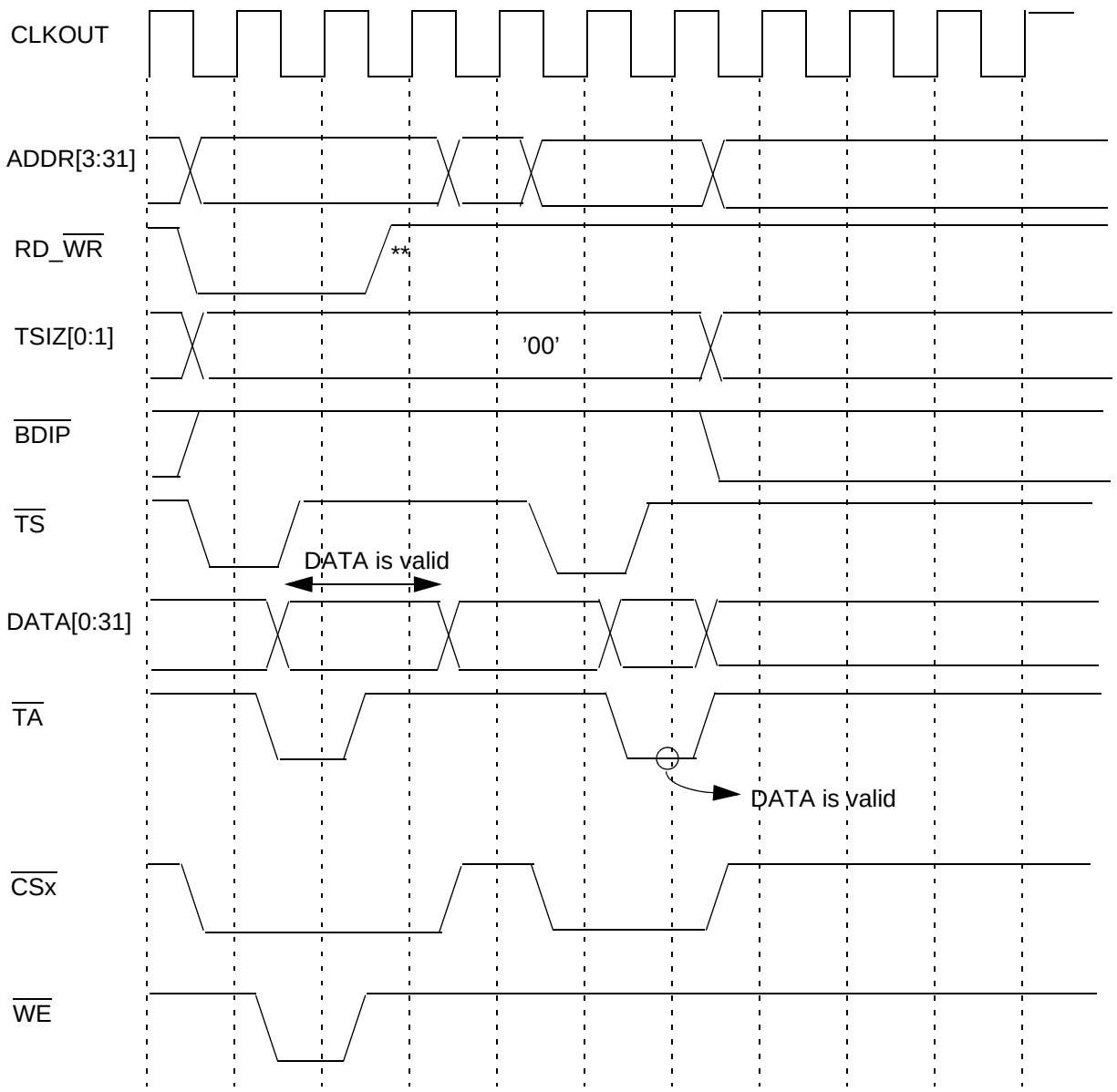


Figure 22-24. Read After Write to the Same CS Bank



* Timing shown applies when EBI_BR[GCSN]=1.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

Figure 22-25. Read after write to the same CS bank (EBI_BR[GCSN] = 1)

22.4.2.5 Burst transfer

The EBI supports wrapping 32-byte critical-doubleword-first burst transfers. Bursting is supported only for internally requested (for example, core, DMA, etc.) cache-line size (32-byte) read accesses to external devices that use the chip selects[1].

Accesses to devices operating without a chip select are always single beat. If an internal request to the EBI indicates a size of less than 32 bytes, the request is fulfilled by running one or more single-beat external transfers, not by an external burst transfer.

An 8-word wrapping burst reads eight 32-bit words by supplying a starting address that points to one of the words (doubleword aligned) and requiring the memory device to sequentially drive each word on the data bus. The selected slave device must internally increment ADDR[27:29] (also ADDR30 in the case of a 16-bit port size device) of the supplied address for each transfer, until the address reaches an 8-word boundary, and then wrap the address to the beginning of the 8-word boundary. The address and transfer attributes supplied by the EBI remain stable during the transfers. Termination of each beat transfer occurs by the EBI asserting $\overline{TA}$ (SETA = 1 is not supported for burst transfers). The EBI requires that addresses be aligned to a doubleword boundary on all burst cycles.

Table 22-14 shows the burst order of beats returned for an 8-word burst to a 32-bit port.

Table 22-14. Wrap bursts order

Burst starting address ADDR[27:28]	Burst order (assuming 32-bit port size)
00	word0 → word1 → word2 → word3 → word4 → word5 → word6 → word7
01	word2 → word3 → word4 → word5 → word6 → word7 → word0 → word1
10	word4 → word5 → word6 → word7 → word0 → word1 → word2 → word3
11	word6 → word7 → word0 → word1 → word2 → word3 → word4 → word5

The general case of burst transfers assumes that the external memory has 32-bit port size and 8-word burst length (SBL = 0, BL = 0). The EBI can also burst from 16-bit port size memories, taking twice as many external beats to fetch the data as compared to a 32-bit port with the same burst length. The EBI can also burst from 16-bit or 32-bit memories that have a 4-word burst length (SBL = 0, BL = 1 in the appropriate Base Register). In this case, two external 4-word burst transfers (wrapping on 4-word boundary) are performed to fulfill the internal 8-word request[2]. This operation is considered atomic by the EBI, so the EBI does not allow other unrelated master accesses or bus arbitration to intervene between the transfers. For more details and a timing diagram, see Section 22.4.2.6.3, Small access example #3: 32-byte read to 32-bit port with BL = 1.

The EBI can also burst from 16-bit or 32-bit memories that have a 2-word burst length (SBL = 1, BL = X in the appropriate Base Register). In this case, four[3] external 2-word burst transfers (wrapping on 2-word boundary) are performed to fulfill the internal 8-word request. This operation is considered atomic by the EBI, so the EBI does not allow other unrelated master accesses or bus arbitration to intervene between the

1. Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See Section 22.4.2.9, Non-chip-select burst in 16-bit data bus mode.
2. This case (of 2 external 4-word burst transfers being required) applies only to AMBA data bus width of 64 bits.
3. Applies to 64-bit AMBA data bus width. For 32-bit AMBA data bus width, only two transfers are performed.

transfers. For more details and a timing diagram, see Section 22.4.2.6.5, Small access example #5: 32-byte read to 32-bit port with SBL = 1.

During burst cycles, the $\overline{\text{BDIP}}$ (Burst Data In Progress) signal indicates the duration of the burst data. During the data phase of a burst read cycle, the EBI receives data from the addressed slave. If the EBI needs more than one data, it asserts the $\overline{\text{BDIP}}$ signal. Upon receiving the data prior to the last data, the EBI negates $\overline{\text{BDIP}}$. Thus, the slave stops driving new data after it receives the negation of $\overline{\text{BDIP}}$ on the rising edge of the clock. Some slave devices have their burst length and timing configurable internally and thus may not support connecting to a $\overline{\text{BDIP}}$ pin. In this case, $\overline{\text{BDIP}}$ is driven by the EBI normally, but the output is ignored by the memory and the burst data behavior is determined by the internal configuration of the EBI and slave device. When the TBDIP bit is set in the appropriate Base Register, the timing for $\overline{\text{BDIP}}$ is altered. See Section 22.4.2.5.1, TBDIP effect on burst transfer, for this timing.

Since burst writes are not supported by the EBI[1], the EBI negates $\overline{\text{BDIP}}$ during write cycles.

1. Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See Section 22.4.2.9, Non-chip-select burst in 16-bit data bus mode.

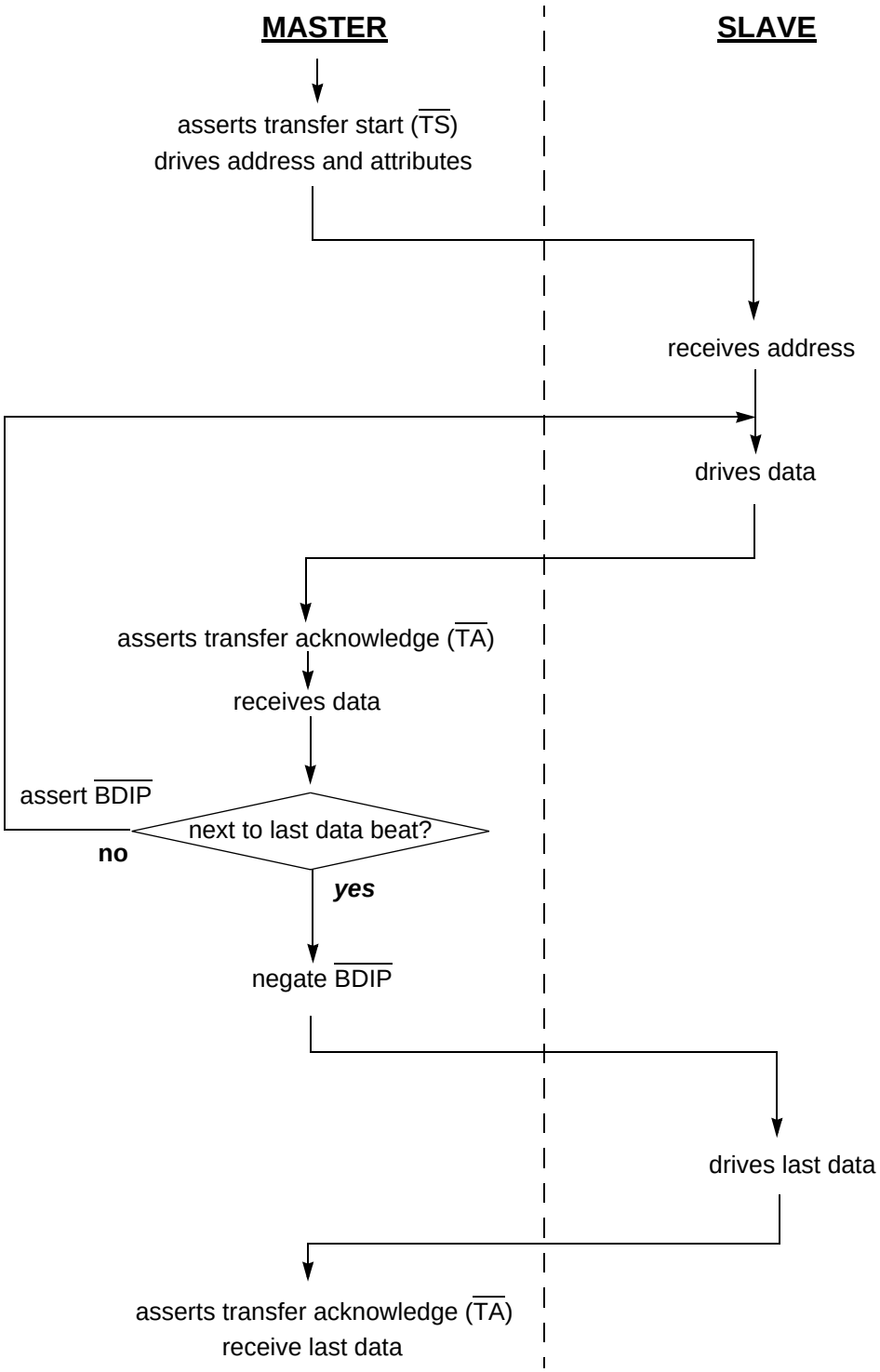


Figure 22-26. Basic flow diagram of a burst read cycle

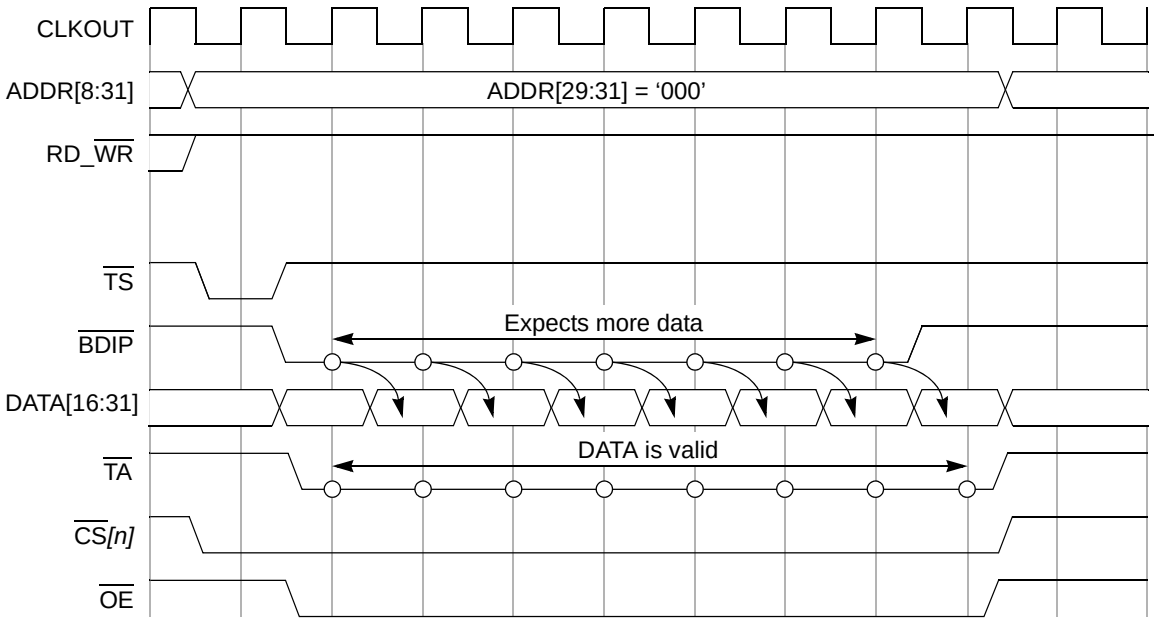


Figure 22-27. Burst 32-bit read cycle, zero wait states

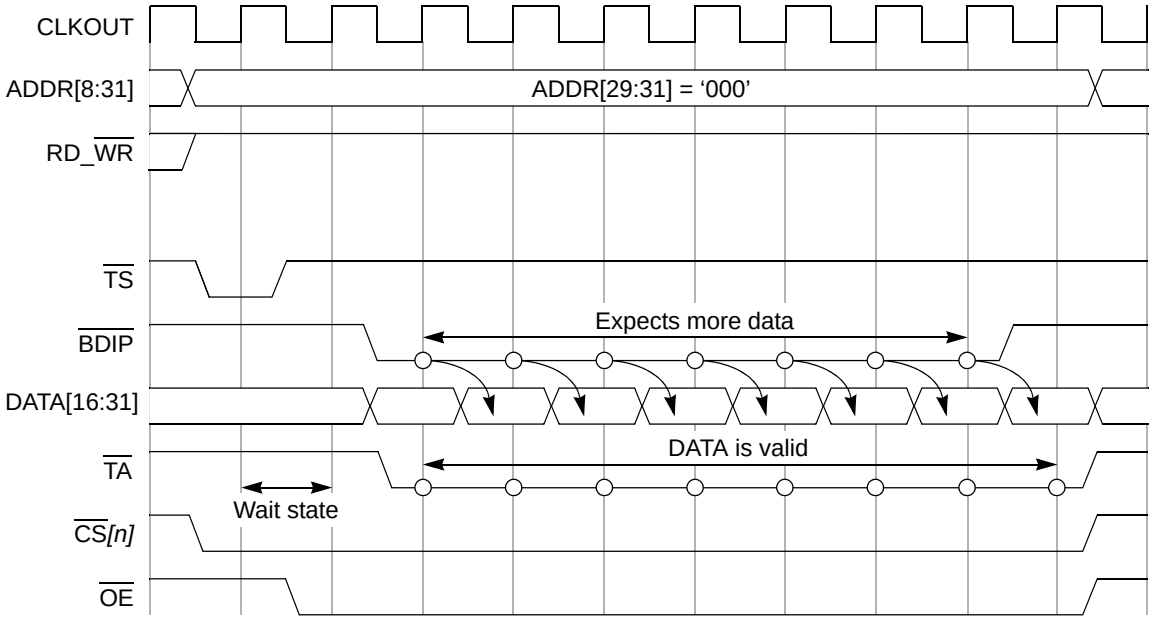


Figure 22-28. Burst 32-bit read cycle, one initial wait state

22.4.2.5.1 TBDIP effect on burst transfer

Some memories require different timing on the $\overline{\text{BDIP}}$ signal than the default to run burst cycles. Using the default value of TBDIP = 0 in the appropriate EBI Base Register results in $\overline{\text{BDIP}}$ being asserted (SCY + 1) cycles after the address transfer phase, and being held asserted throughout the cycle regardless of the wait

states between beats (BSCY). Figure 22-29 shows an example of the TBDIP = 0 timing for a 4-beat burst with BSCY = 1.

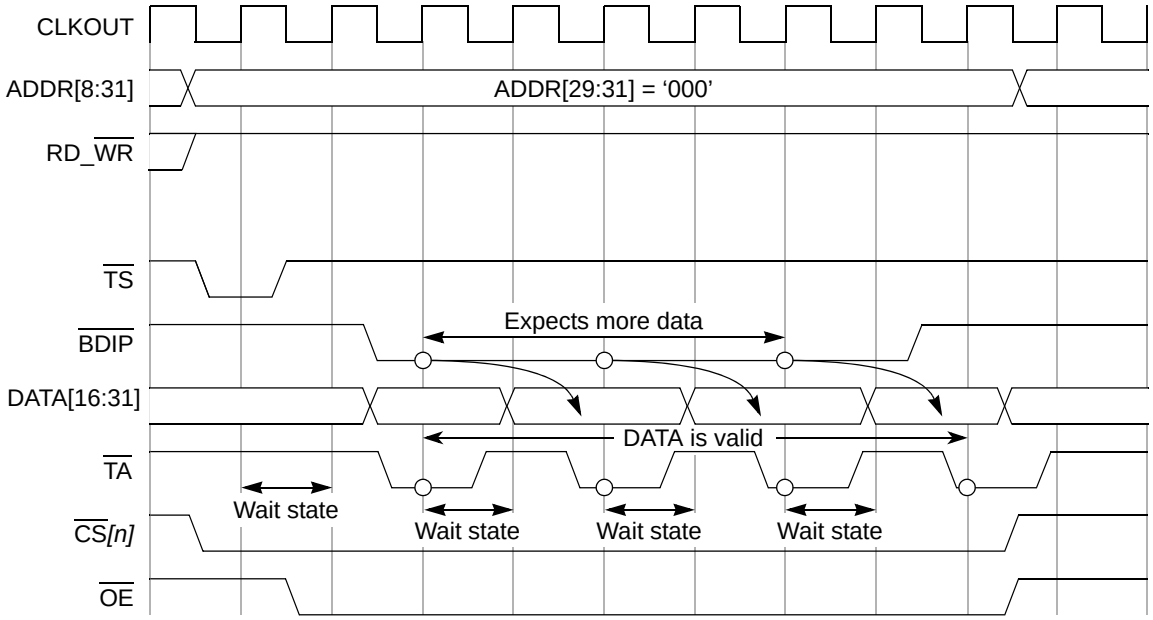


Figure 22-29. Burst 32-bit read cycle, one wait state between beats, TBDIP = 0

When using TBDIP = 1, the BDIP behavior changes to toggle between every beat when BSCY is a non-zero value. Figure 22-30 shows an example of the TBDIP = 1 timing for the same 4-beat burst shown in Figure 22-29.

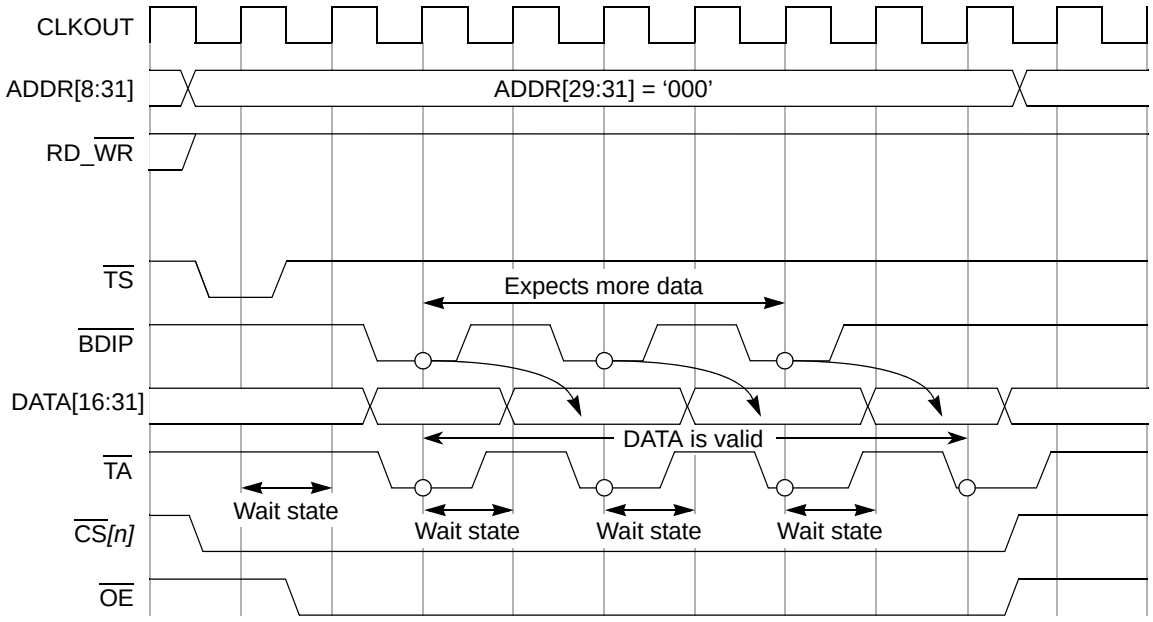


Figure 22-30. Burst 32-bit read cycle, one wait state between beats, TBDIP = 1

22.4.2.6 Small accesses (small port size and short burst length)

In this context, a *small access* refers to an access whose burst length and port size (SBL, BL, PS bits in Base Register for chip-select access or default burst disabled, 32-bit port for non-chip-select access) are such that the number of bytes requested by the internal master cannot all be fetched (or written) in one external transaction. If this is the case, the EBI initiates multiple transactions until all the requested data is transferred. It should be noted that all the transactions initiated to complete the data transfer are considered as an atomic transaction, so the EBI does not allow other unrelated master accesses to intervene between the transfers.

Table 22-15 shows all the combinations of burst length, port size, and requested byte count that cause the EBI to run multiple external transactions to fulfill the request.

Table 22-15. Small access cases

Byte count requested by internal master	Burst length	Port size	# external accesses to fulfill request
Non-burstable chip-select banks (BI = 1) or non-chip-select access			
4	1 beat	16-bit	2/1[1]
8	1 beat	32-bit	2
8	1 beat	16-bit	4
16	1 beat	32-bit	4
16	1 beat	16-bit	8
32[2]	1 beat	32-bit	8
32[2]	1 beat	16-bit	16
Burstable chip-select banks (BI = 0)			
32[2]	4 words	16-bit (8 beats), 32-bit (4 beats)	2
32[3]	2 words	16-bit (4 beats), 32-bit (2 beats)	4

1 In 32-bit data bus mode (DBM = 0 in EBI_MCR), two accesses are performed. In 16-bit data bus mode (DBM = 1), one 2-beat burst access is performed and this is not considered a “small access” case. See Section 22.4.2.9, Non-chip-select burst in 16-bit data bus mode, for this special DBM = 1 case.

2 Only supported for case of 64-bit internal AMBA data bus.

3 SBL = 1 (2-word burst case).

In most cases, the timing for small accesses is the same as for normal single-beat and burst accesses, except that multiple back-to-back external transfers are executed for each internal request. These transfers have no additional dead cycles in-between that are not present for back-to-back stand-alone transfers except for the case of writes with an internal request size of > 64 bits, discussed in Section 22.4.2.6.2, Small access example #2: 32-byte write with external TA.

The following sections show a few examples of small accesses. The timing for the remaining cases in Table 22-15 can be extrapolated from these and the other timing diagrams in this document.

22.4.2.6.1 Small access example #1: 32-bit write to 16-bit port

Figure 22-31 shows an example of a 32-bit write to a 16-bit port, requiring two 16-bit external transactions.

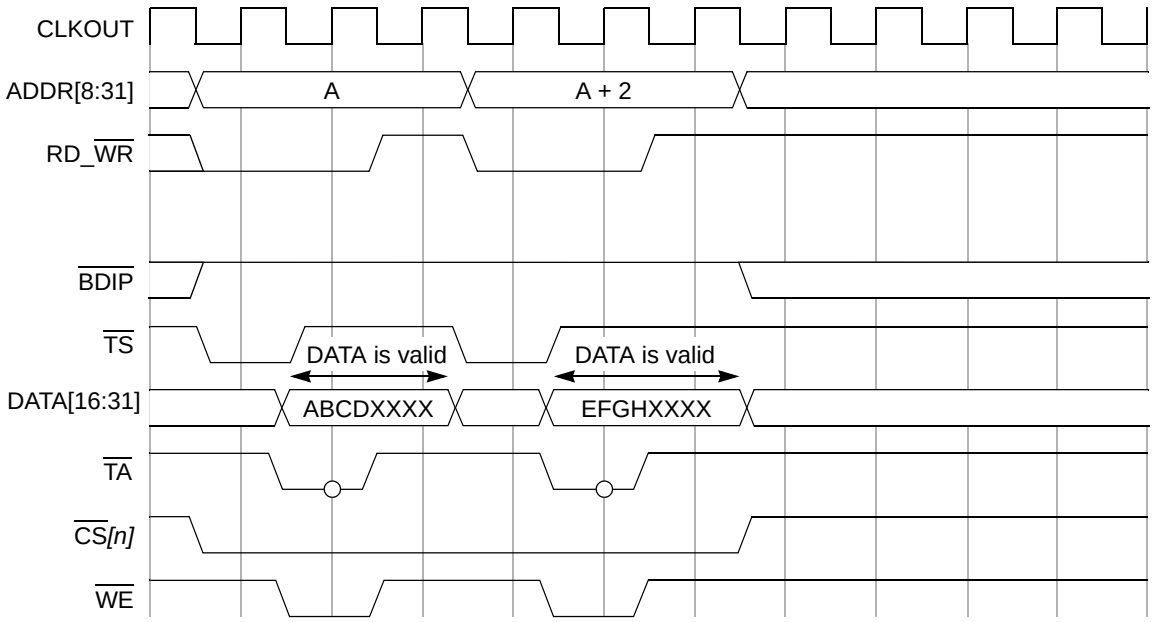
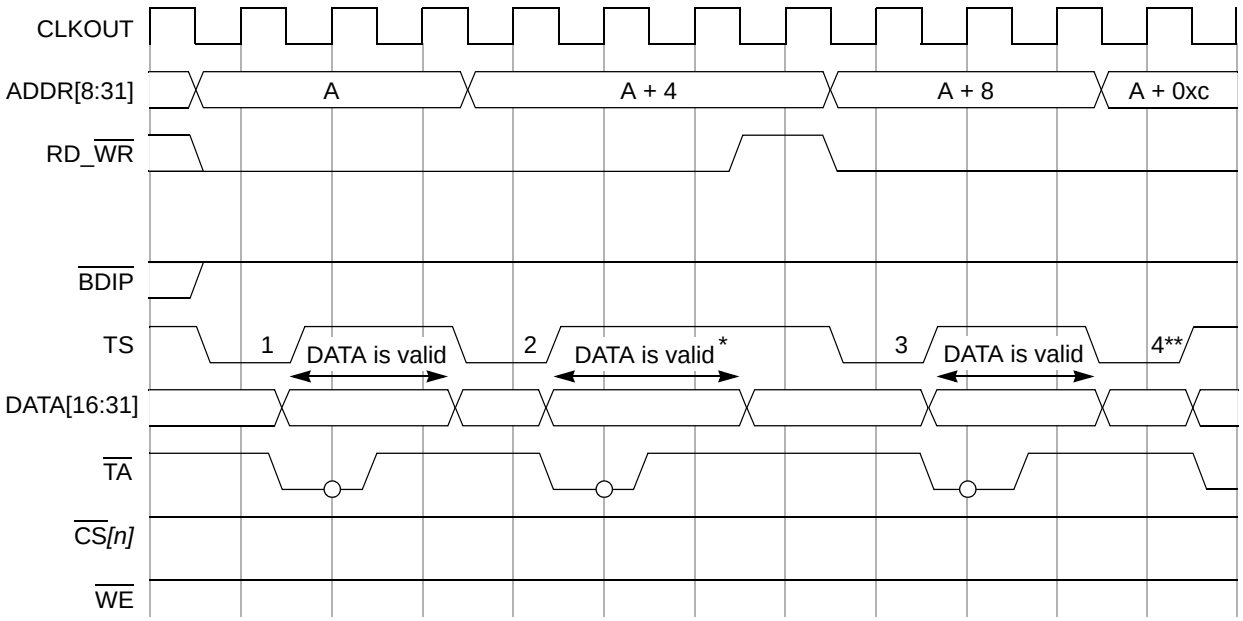


Figure 22-31. Single beat 32-bit write cycle, 16-bit port size, basic timing

22.4.2.6.2 Small access example #2: 32-byte write with external TA

Figure 22-32 shows an example of a 32-byte write to a non-chip-select device using external TA, requiring eight 32-bit external transactions. Note that due to the use of external TA, RD_WR does not toggle between the accesses unless that access is the end of a 64-bit boundary. In this case, an extra cycle is required between TA and the next TS in order to get the next 64 bits of write data internally and RD_WR negates during this extra cycle.



* This extra cycle is required after accesses 2, 4, and 6 to get the next 64-bits of internal write data.

** Four more external accesses (not shown) are required to complete the internal 32-byte request. The timing of these is the same as accesses 1-4 shown in this diagram.

Figure 22-32. 32-byte write cycle with external ta, basic timing

22.4.2.6.3 Small access example #3: 32-byte read to 32-bit port with BL = 1

Figure 22-33 shows an example of a 32-byte read to a 32-bit burst enabled port with burst length of 4 words, requiring two 4-word (16-byte) external transactions. For this case, the address for the second 4-word burst access is calculated by adding 0x10 to the lower 5 bits of the first address (no carry), and then masking out the lower 4 bits to fix them at zero.

Table 22-16. Examples of 4-word burst addresses

1st address	Lower 5 bits of 1st address + 0x10 (no carry)	Final 2nd address (after masking lower 4 bits)
0x000	0x10	0x10
0x008	0x18	0x10
0x010	0x00	0x00
0x018	0x08	0x00
0x020	0x30	0x30
0x028	0x38	0x30
0x030	0x20	0x20
0x038	0x28	0x20

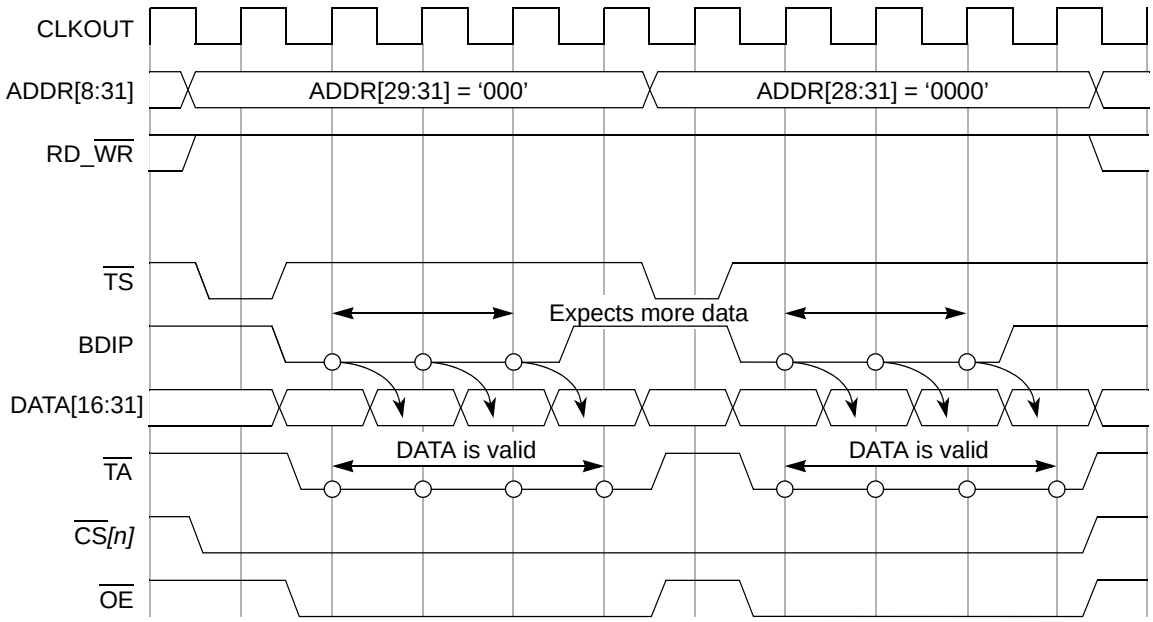
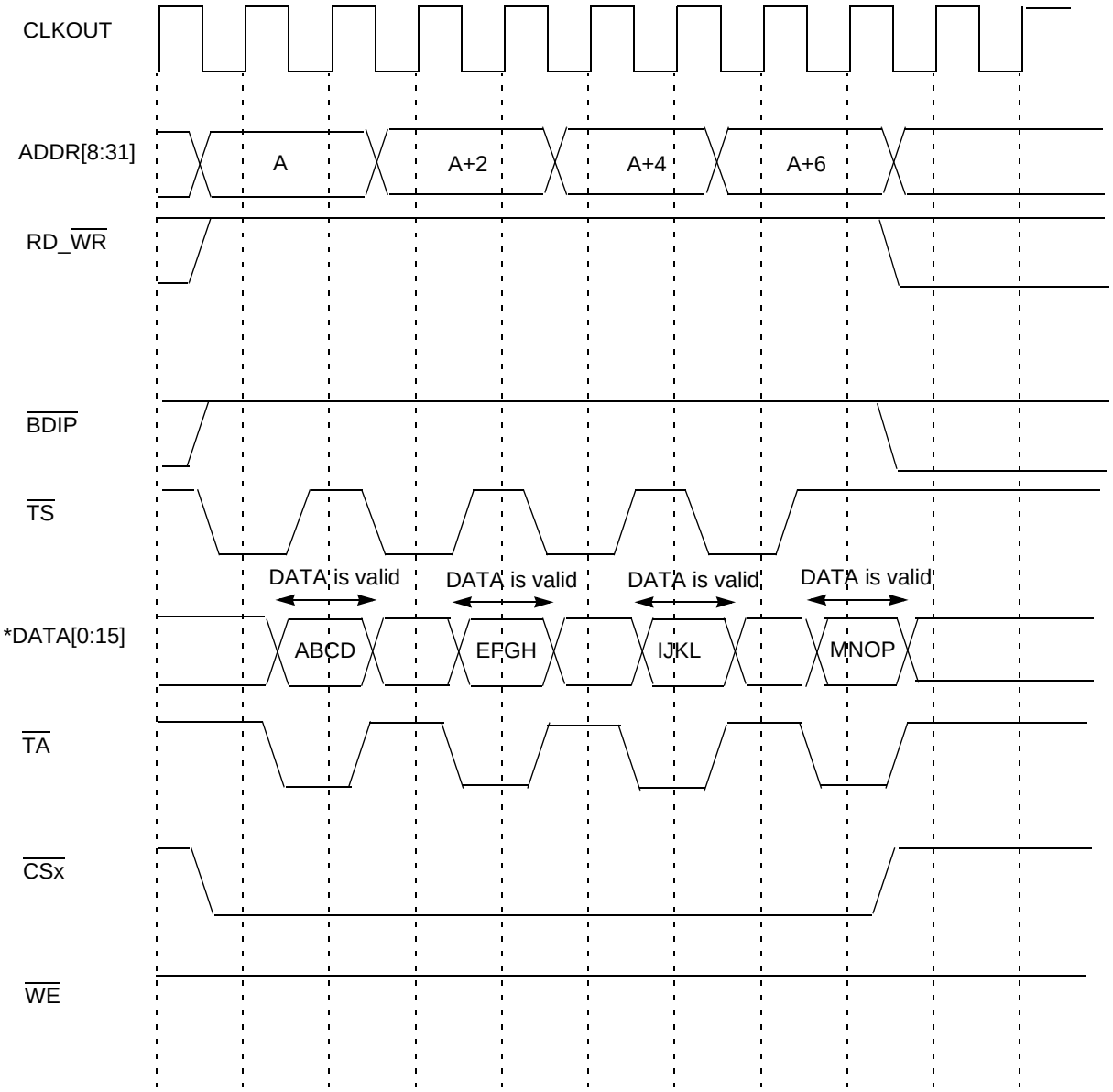


Figure 22-33. 32-byte read with b-t-b 4-word bursts to 32-bit port, zero wait states

22.4.2.6.4 Small access example #4: 64-bit read to 16-bit port

Figure 22-34 shows an example of a 64-bit read to a 16-bit port, requiring four 16-bit external transactions.



* Or DATA[16:31], based on D16_31 bit in EBI_MCR.

Figure 22-34. Single beat 64-bit read cycle, 16-bit port size, basic timing

22.4.2.6.5 Small access example #5: 32-byte read to 32-bit port with SBL = 1

Figure 22-33 shows an example of a 32-byte read to a 32-bit burst-enabled port with burst length of 2 words, requiring four 8-byte external transactions. For this case, the address for the second through fourth 2-word burst accesses is calculated by adding 0x08 to the lower 5 bits of the first address (no carry), and then masking out the lower 3 bits to fix them at zero.

Table 22-17. Examples of 2-word burst addresses

1st address	Lower 5 bits of 1st address + 0x08 (no carry)	Final 2nd-4th addresses (after masking lower 3 bits)
0x000	0x08	0x08, 0x10, 0x18
0x008	0x10	0x10, 0x18, 0x00
0x010	0x18	0x18, 0x00, 0x08
0x018	0x00	0x00, 0x8, 0x10
0x020	0x28	0x28, 0x30, 0x38
0x028	0x30	0x30, 0x38, 0x20
0x030	0x38	0x38, 0x20, 0x28
0x038	0x20	0x20, 0x28, 0x30

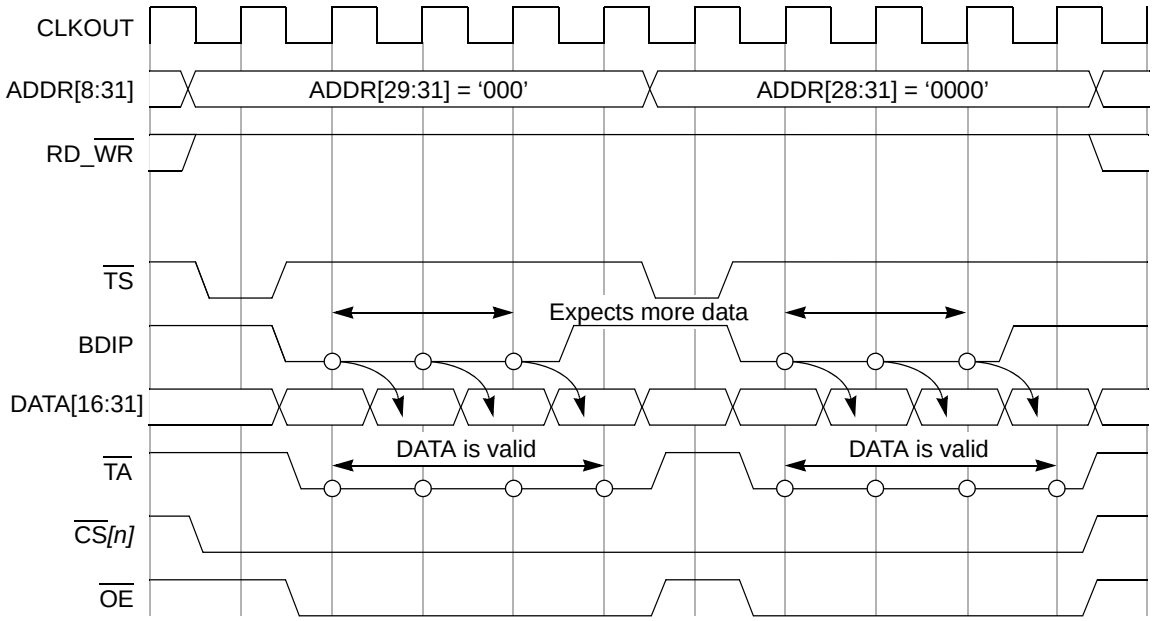


Figure 22-35. 32-byte read with 4 B-T-B 2-word bursts to 32-bit port, zero wait states (SBL = 1)

22.4.2.7 Size, alignment, and packaging on transfers

Table 22-18 shows the allowed sizes that an internal master can request from the EBI. The behavior of the EBI for request sizes not shown below is undefined. No error signal is asserted for these erroneous cases.

Table 22-18. Transaction sizes supported by EBI

# bytes (internal master)	# bytes (external master)
1	1
2	2
4	4

Table 22-18. Transaction sizes supported by EBI (continued)

# bytes (internal master)	# bytes (external master)
3[1]	
8	
32[2]	

1 Some misaligned access cases may result in 3-byte writes. These cases are treated as power-of-2 sized requests by the EBI, using WE_BE[0:3] to make sure only the appropriate 3 bytes get written.

2 Only supported for case of 64-bit internal AMBA data bus.

Even though misaligned non-burst transfers from internal masters are supported, the EBI naturally aligns the accesses when it sends them out to the external bus, splitting them into multiple aligned accesses if necessary. See Section 22.4.2.10, Misaligned access support, for these cases.

Natural alignment for the EBI means:

- Byte access can have any address
- 16-bit access, address bit 31 must be 0
- 32-bit access, address bits 30-31 must be 0
- For burst accesses of any size, address bits 29-31 must be 0

The EBI never generates a misaligned external access, so a multi-master system with two e200-based MCUs can never have a misaligned external access from one to the other. In the erroneous case that an externally initiated misaligned access does occur, the EBI errors the access (by asserting TEA externally) and does not initiate the access on the internal bus.

The EBI requires that the portion of the data bus used for a transfer to/from a particular port size be fixed. A 32-bit port must reside on data bus bits 0-31, and a 16-bit port must reside on bits 0-15.

In the following figures and tables the following convention is adopted:

- The most significant byte of a 32-bit operand is OP0, and OP3 is the least significant byte.
- The two bytes of a 16-bit operand are OP0 (most significant) and OP1, or OP2 (most significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

This can be seen in Figure 22-36.

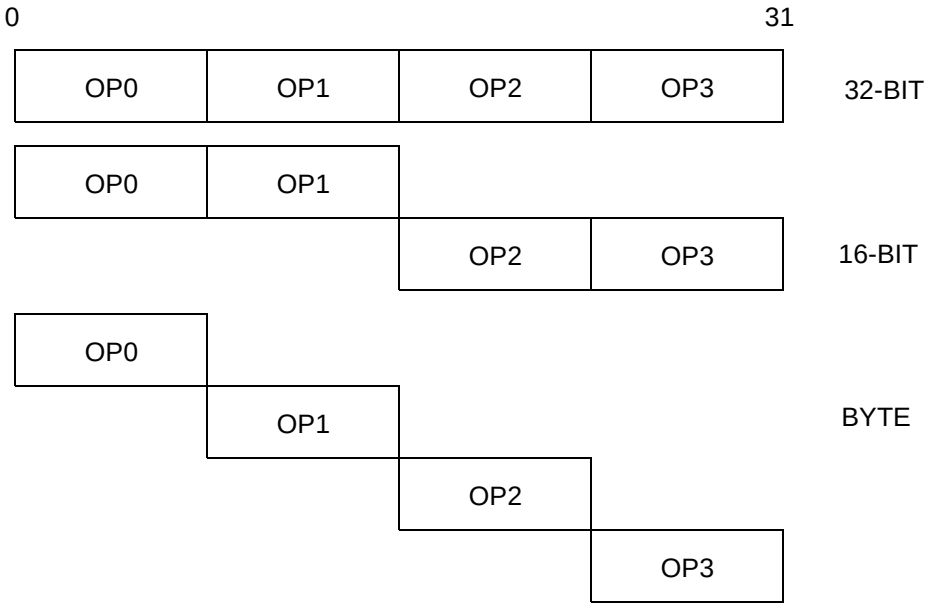


Figure 22-36. Internal operand representation

Figure 22-37 shows the device connections on the DATA[16:31] bus.

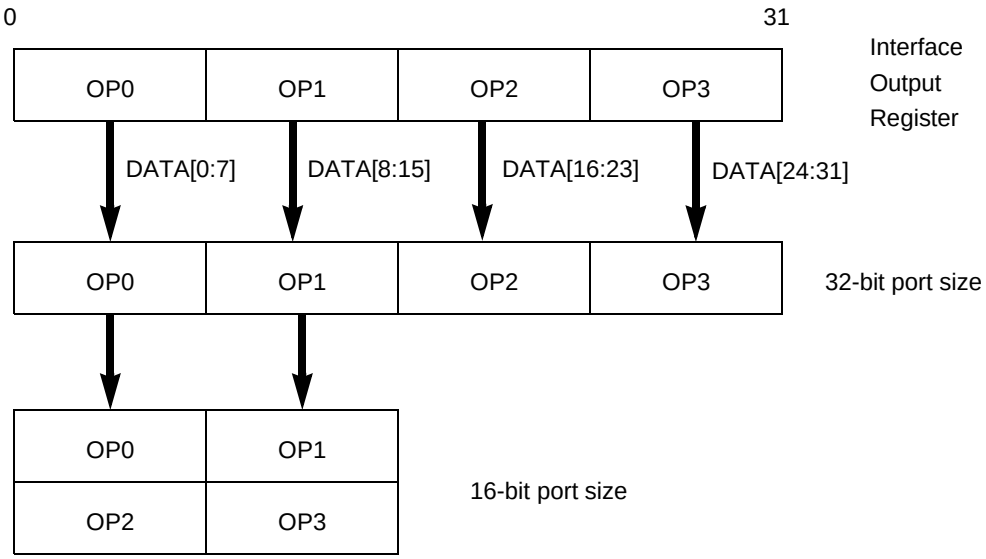


Figure 22-37. Interface to different port size devices

Table 22-19 lists the bytes required on the data bus for read cycles. The bytes indicated as '—' are not required during that read cycle.

Table 22-19. Data bus requirements for read cycles

Transfer size	TSIZ[0:1]	Address		32-bit port size				16-bit port size[1]	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7[2]	D8:D15[3]
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	—	OP1	—	—	—	OP1
	01	1	0	—	—	OP2	—	OP2	—
	01	1	1	—	—	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2[4]	OP1/OP3

1 Also applies when DBM = 1 for 16-bit data bus mode.

2 For address/data muxed transfers, DATA[16:23] are used externally, not DATA[0:7].

3 For address/data muxed transfers, DATA[24:31] are used externally, not DATA[8:15].

4 This case consists of two 16-bit external transactions, the first fetching OP0 and OP1, the second fetching OP2 and OP3.

Table 22-20 lists the patterns of the data transfer for write cycles when accesses are initiated by the MCU. The bytes indicated as '—' are not driven during that write cycle.

Table 22-20. Data bus contents for write cycles

Transfer size	TSIZ[0:1]	Address		32-bit port size				16-bit port size[1]	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7[2]	D8:D15[3]
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	—	OP1	—	—	—	OP1
	01	1	0	—	—	OP2	—	OP2	—
	01	1	1	—	—	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2[4]	OP1/OP3

1 Also applies when DBM = 1 for 16-bit data bus mode.

2 For address/data muxed transfers, DATA[16:23] are used externally, not DATA[0:7].

3 For address/data muxed transfers, DATA[24:31] are used externally, not DATA[8:15].

4 This case consists of two 16-bit external transactions, the first writing OP0 and OP1, the second writing OP2 and OP3.

22.4.2.8 Termination signals protocol

The termination signals protocol was defined in order to avoid electrical contention on lines that can be driven by various sources. In order to do that, a slave must not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave must disconnect from signals immediately after it acknowledges the cycle and not later than the termination of the next address phase cycle.

For EBI-mastered non-chip-select accesses, the EBI requires assertion of $\overline{\text{TA}}$ from an external device to signal that the bus cycle is complete. The EBI uses a latched version of $\overline{\text{TA}}$ (1 cycle delayed) for these

accesses to help make timing at high frequencies. This results in the EBI driving the address and control signals 1 cycle longer than required, as seen in Figure 22-38. However, the DATA does not need to be held 1 cycle longer by the slave, because the EBI latches DATA every cycle during non-chip-select accesses. During these accesses, the EBI does not drive the $\overline{TA}$ signal, leaving it up to an external device (or weak internal pullup) to drive $\overline{TA}$.

For EBI-mastered chip-select accesses, when the SETA bit is 0, the EBI drives $\overline{TA}$ the entire cycle, asserting according to internal wait state counters to terminate the cycle. When the SETA bit is 1, the EBI samples the $\overline{TA}$ for the entire cycle. During idle periods on the external bus, the EBI drives $\overline{TA}$ negated as long as it is granted the bus; when it no longer owns the bus, it lets go of $\overline{TA}$.

If no device responds by asserting $\overline{TA}$ within the programmed timeout period (BMT in EBI_BMCR) after the EBI initiates the bus cycle, the internal Bus Monitor (if enabled) asserts $\overline{TEA}$ to terminate the cycle. An external device may also drive $\overline{TEA}$ when it detects an error on an external transaction. $\overline{TEA}$ assertion causes the cycle to terminate and the processor to enter exception processing for the error condition. To properly control termination of a bus cycle for a bus error with external circuitry, $\overline{TEA}$ must be asserted at the same time or before (external) $\overline{TA}$ is asserted. $\overline{TEA}$ must be negated before the second rising edge after it was sampled asserted in order to avoid the detection of an error for the following bus cycle initiated. $\overline{TEA}$ is only driven by the EBI during the cycle where the EBI is asserting $\overline{TEA}$ and the cycle immediately following this assertion (for fast negation). During all other cycles, the EBI relies on a weak internal pullup to hold $\overline{TEA}$ negated. This allows an external device to assert $\overline{TEA}$ when it needs to indicate an error. External devices must follow the same protocol as the EBI, only driving $\overline{TEA}$ during the assertion cycle and 1 cycle afterwards for negation.

When $\overline{TEA}$ is asserted from an external source, the EBI uses a latched version of $\overline{TEA}$ (1 cycle delayed) to help make timing at high frequencies. This means that for any accesses in which the EBI drives $\overline{TA}$ (chip-select accesses with SETA = 0), a $\overline{TEA}$ assertion that occurs 1 cycle before or during the last $\overline{TA}$ of the access could be ignored by the EBI, since it will have completed the access internally before it detects the latched $\overline{TEA}$ assertion. This means that non-burst chip-select accesses with no wait states (SCY = 0) cannot be reliably terminated by external $\overline{TEA}$. If external error termination is required for such a device, the EBI must be configured for SCY ≥ 1.

NOTE

For the cases discussed above where $\overline{TEA}$ “could be ignored”, this is not guaranteed. For some small access cases (which always use chip-select and internally driven $\overline{TA}$), a $\overline{TEA}$ that occurs 1 cycle before or during the $\overline{TA}$ cycle or for SCY = 0 may in fact lead to terminating the cycle with error. However, proper error termination is not guaranteed for these cases, so $\overline{TEA}$ must always be asserted at least 2 cycles before an internally driven $\overline{TA}$ cycle for proper error termination.

External $\overline{TEA}$ assertion that occurs during the same cycle that $\overline{TS}$ is asserted by the EBI is always treated as an error (terminating the access) regardless of SCY.

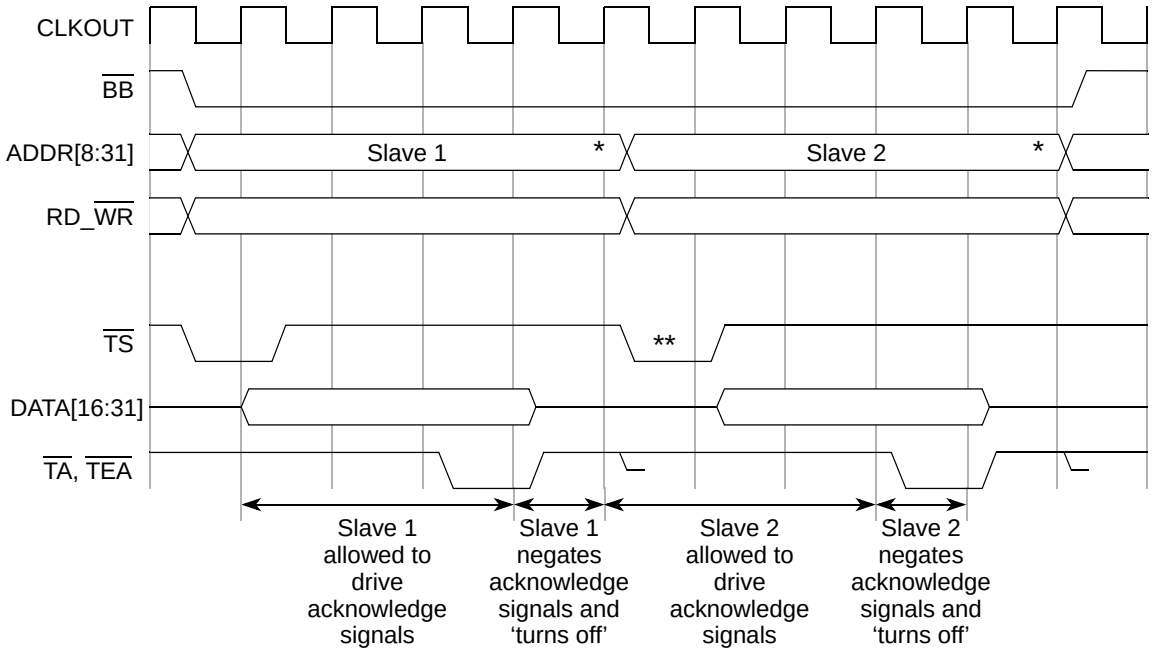
Table 22-21 summarizes how the EBI recognizes the termination signals provided from an external device.

Table 22-21. Termination signals provided from external device

TEA[1]	TA[1]	Action
Negated	Negated	No Termination
Asserted	X	Transfer Error Termination
Negated	Asserted	Normal Transfer Termination

1 Latched version (1 cycle delayed) used for externally driven $\overline{TEA}$ and $\overline{TA}$.

Figure 22-38 shows an example of the termination signals protocol for back-to-back reads to two different slave devices who properly “take turns” driving the termination signals. This assumes a system using slave devices that drive termination signals.



* The EBI drives address and control signals an extra cycle because it uses a latched version of $\overline{TA}$ (1 cycle delayed) to terminate the cycle. An external master is not required to do this.

** This is the earliest that the EBI can start another transfer, in the case of continuing a set of small accesses. For all other cases, an extra cycle is needed before the EBI can start another $\overline{TS}$.

Figure 22-38. termination signals protocol timing diagram

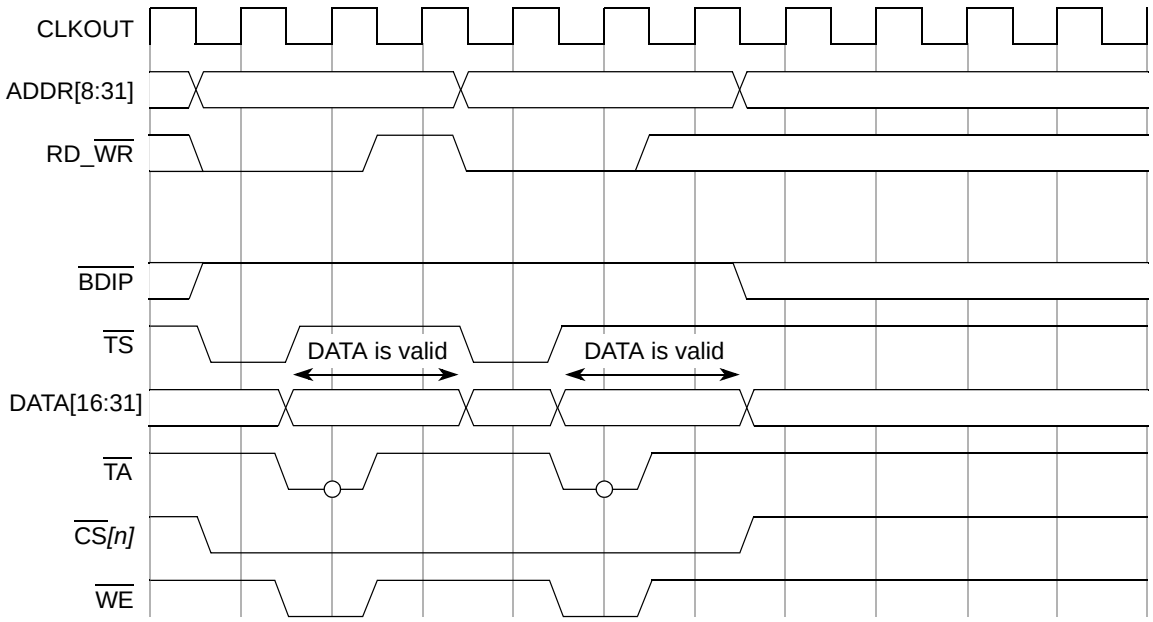


Figure 22-39. External master back-to-back writes to same CS bank

22.4.2.9 Non-chip-select burst in 16-bit data bus mode

The timing diagrams in this section apply only to the special case of a non-chip-select 32-bit access in 16-bit data bus mode (DBM = 1 in EBI_MCR).

NOTE

For this case, a special 2-beat burst protocol is used for reads and writes, so that a slave device (using the same EBI) can internally generate one 32-bit read or write access (thus 32-bit coherent), as opposed to two separate 16-bit accesses.
If the device does not support multi-master systems, the original intent of this protocol does not apply. However, this 2-beat burst protocol can also occur in a single-master system, if a non-chip-select 32-bit access to a 16-bit port is performed.

Figure 22-40 shows a 32-bit (and non-chip-select in a single-master system) read from an external master in 16-bit data bus mode.

Figure 22-41 shows a 32-bit (and non-chip-select in a single-master system) write from an external master in 16-bit data bus mode.

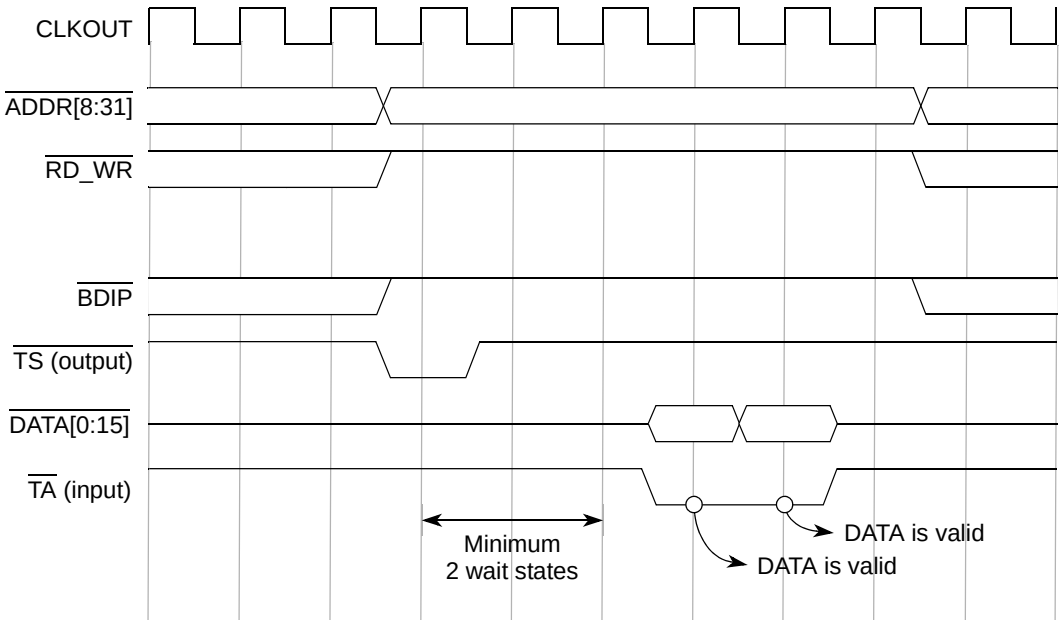


Figure 22-40. 32-bit read from MCU with DBM = 1

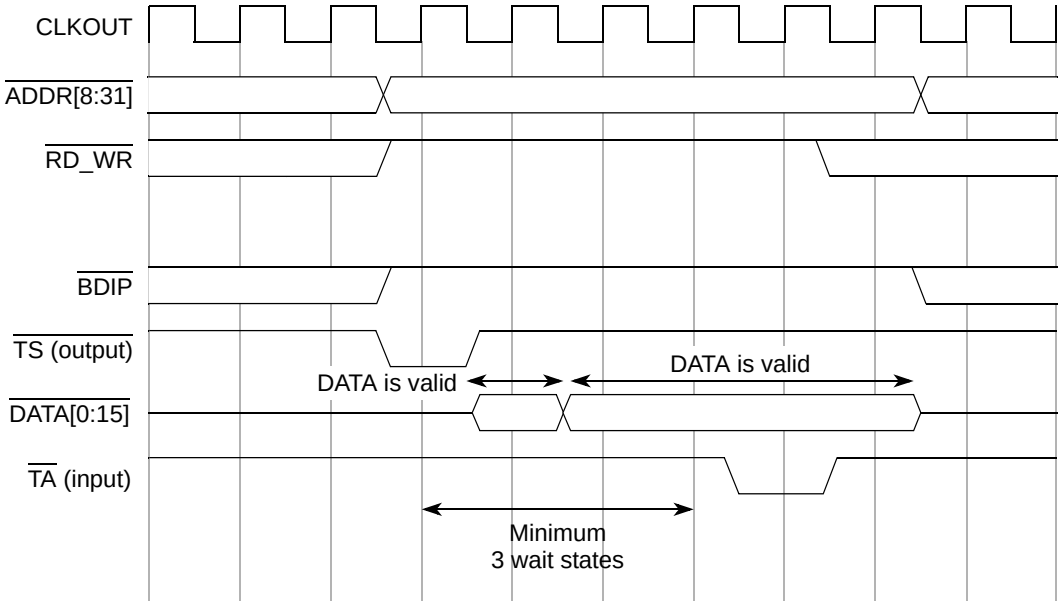


Figure 22-41. 32-bit write to MCU with DBM = 1

22.4.2.10 Misaligned access support

This section describes all the misaligned cases supported by the EBI. These cases are a subset of the full set of cases allowed by the AMBA AHB V6 specification. The EBI works under the assumption that all internal masters on the device do not produce any misaligned access cases (to the EBI) other than the ones below.

22.4.2.10.1 Misaligned access support (64-bit AMBA)

Table 22-22 shows all the misaligned access cases supported by the EBI (using a 64-bit AMBA implementation), as seen on the internal master AMBA bus. All other misaligned cases are not supported. If an unsupported misaligned access to the EBI is attempted (such as non-chip-select or burst misaligned access), the EBI errors the access on the internal bus and does not start the access (nor assert $\overline{\text{TEA}}$) externally.

Table 22-22. Misalignment cases supported by a 64-bit AMBA EBI (internal bus)

No.[1]	Program size and byte offset	Address [29:31][2]	Data bus byte strobes[3]	HSIZE[4]	HUNALIGN[5]
1	Half @0x1,0x9	001	0110_0000	10	1
2	Half @0x3,0xB	011	0001_1000	11	1
3	Half @0x5,0xD	101	0000_0110	10	1
4 —	Half @0x7, 0xF (2 AHB transfers)	111 000	0000_0001 1000_0000	01[6] 00	1 0
5	Word @0x1,0x9	001	0111_1000	11	1
6	Word @0x2,0xA	010	0011_1100	11	1
7	Word @0x3,0xB	011	0001_1110	11	1
8 —	Word @0x5,0xD (2 AHB transfers)	101 000	0000_0111 1000_0000	10 00	1 0
9 —	Word @0x6, 0xE (2 AHB transfers)	110 000	0000_0011 1100_0000	10[7] 01	1 0
10 11	Word @0x7,0xF (2 AHB transfers)	111 000	0000_0001 1110_0000	10[6] 10	1 1
12 —	Doubleword @0x4,0x8 (2 AHB transfers)	100 000	0000_1111 1111_0000	11[8] 10	1 0
13 —	Doubleword @0x2,0xA (2 AHB transfers)	010 000	0011_1111 1100_0000	11 01	1 0
14 15	Doubleword 0x6,0xE (2 AHB transfers)	110 000	0000_0011 1111_1100	11[7] 11	1 1

1 Misaligned case number. Only transfers where HUNALIGN = 1 are numbered as misaligned cases.

2 Address on internal master AHB bus, not necessarily address on external ADDR pins.

3 Internal byte strobe signals on AHB bus. Shown with Big-Endian byte ordering in this table, even though internal master AHB bus uses Little-Endian byte-ordering (EBI flips order internally).

4 Internal signal on AHB bus; 00 = 8-bits, 01 = 16 bits, 10 = 32 bits, 11 = 64 bits. HSIZE is driven according to the smallest aligned container that contains all the requested bytes. This results in extra EBI external transfers in some cases.

5 Internal signal on AHB bus that indicates that this transfer is misaligned (when 1).

6 For this case, the EBI internally treats HSIZE as 00 (1-byte access).

7 For this case, the EBI internally treats HSIZE as 01 (2-byte access).

8 For this case, the EBI internally treats HSIZE as 10 (4-byte access).

Table 22-23 shows which external transfers are generated by the EBI for the misaligned access cases in Table 22-22, for each port size.

The number of external transfers for each internal AHB master request is determined by the HSIZE value for that request relative to the port size. For example, a half-word write to @011 (misaligned case #2) with 16-bit port size results in 4 external 16-bit transfers because the HSIZE is 64 bits. For cases where two or more external transfers are required for one internal transfer request, these external accesses are considered part of a “small access” set, as described in Section 22.4.2.6, Small accesses (small port size and short burst length).

Since all transfers are aligned on the external bus, normal timing diagrams and protocol apply. Note that the TSIZ[0:1] signals are not intended to be used for misaligned accesses, so they are not specified in Table 22-23.

Table 22-23. Misalignment cases supported by a 64-bit AMBA EBI (external bus)

No.[1]	PS[2]	Program size and byte offset	ADDR[29:31][3]	$\overline{\text{WE\_BE}}$[0:3][4]
1	0	Half @0x1,0x9	000	1001
	1		000 010	1011 0111
2	0	Half @0x3,0xB	000 100	1110 0111
	1		010 100	1011 0111
3	0	Half @0x5,0xD	100	1001
	1		100 110	1011 0111
4	0	Half @0x7,0xF (2 AHB transfers)	111[5]	1110
—			000	0111
4	1		110	1011
—			000	0111
5	0	Word @0x1,0x9	000 100	1000 0111
	1		000 010 100	1011 0011 0111
6	0	Word @0x2,0xA	000 100	1100 0011
	1		010 100	0011 0011
7	0	Word @0x3,0xB	000 100	1110 0001
	1		010 100 110	1011 0011 0111

Table 22-23. Misalignment cases supported by a 64-bit AMBA EBI (external bus) (continued)

No.[1]	PS[2]	Program size and byte offset	ADDR[29:31][3]	$\overline{\text{WE\_BE}}$[0:3][4]
8	0	Word @0x5,0xD (2 AHB transfers)	100	1000
—			000	0111
8	1		100 110	1011 0011
—			000	0111
9	0	Word @0x6,0xE (2 AHB transfers)	110[6]	1100
—			000	0011
9	1		110[6]	0011
—			000	0011
10	0	Word @0x7,0xF (2 AHB transfers)	111[5]	1110
11			000	0001
10	1		111[5]	1011
11			000 010	0011 0111
12	0	Doubleword @0x4,0xC (2 AHB transfers)	100[7]	0000
—			000	0000
12	1		100[7] 110	0011 0011
—			000 010	0011 0011
13	0	Doubleword @0x2,0xA (2 AHB transfers)	000 100	1100 0000
—			000	0011
13	1		010 100 110	0011 0011 0011
—			000	0011
14	0	Doubleword @0x6,0xE (2 AHB transfers)	110[6]	1100
15			000 100	0000 0011
14	1		110[6]	0011
15			000 010 100	0011 0011 0011

1 Misaligned case number, from Table 22-22.

2 Port size; 0 = 32 bits, 1 = 16 bits.

3 External ADDR pins, not necessarily the address on internal master AHB bus.

4 External $\overline{\text{WE\_BE}}$ pins. Note that these pins have negative polarity, opposite of the internal byte strobes in Table 22-22.

5 Treated as 1-byte access.

6 Treated as 2-byte access.

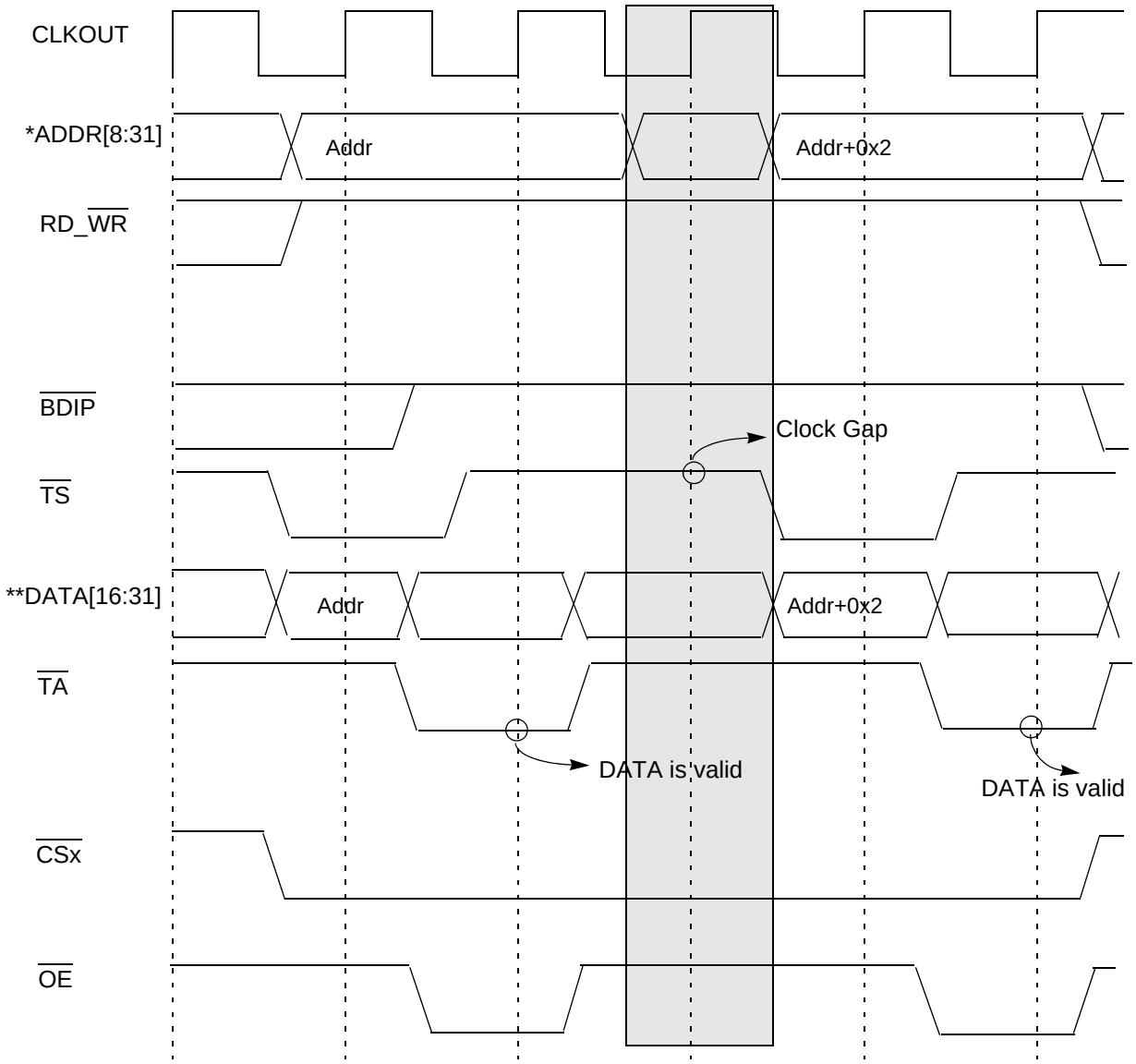
7 Treated as 4-byte access.

22.4.2.11 Address/data multiplexing

Address/data multiplexing enables the design of a system with reduced pin count. In such a system, multiplexed address/data functions (on DATA pins) are used, instead of having separate address and data pins. Compared to the normal EBI specification (for example, 24 address pins + 32 data pins), only 32 data pins are required. Compared to a 16-bit bus implementation, as on the MPC5675K device, only 24 pins are required (for example, ADDR[8:15] + ADDR[16:31]/DATA[16:31]).

When performing a small access read, as described in Section 22.4.2.6, Small accesses (small port size and short burst length), with A/D multiplexing enabled for this access, the EBI inserts an idle clock cycle with $\overline{\text{OE}}$ negated and $\overline{\text{CS}}$ asserted, to allow for the memory to three-state the bus prior to the EBI driving the address on the next clock. This clock gap already exists (for other reasons) for non-small-access transfers, so no additional clock gap is inserted for those cases. See Figure 22-42 for an example of a small access read with A/D multiplexing enabled.

In general, timing diagrams in A/D multiplexing mode are very similar to other diagrams in this document (including support for Burst accesses), except for the behavior of the ADDR and DATA buses, which can be seen in Figure 22-42.



* While the EBI drives all of ADDR[8:31] to valid address, typically only ADDR[3:15] (or less) are used in the system, as DATA[16:31] (or DATA[0:15]) would be used for address and data on an external muxed device.

** Or DATA[0:15], based on D16_31 bit in EBI_MCR.

Figure 22-42. Small access (32-bit read to 16-bit port) on address/data multiplexed bus

22.5 Initialization/application information

22.5.1 Booting from external memory

The EBI block does not support booting directly to external memory (i.e., fetching the first instruction after reset externally). One common method for an MCU to resemble an external boot with this EBI is to use an internal Boot Assist Module on the MCU, which fetches the first instruction internally and configures

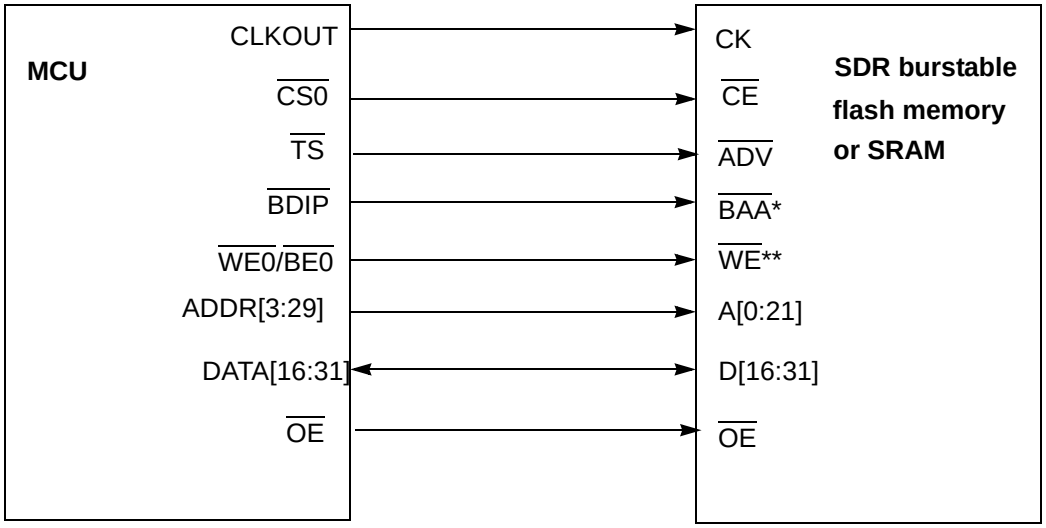
EBI registers before branching to an external address to “boot” externally. Refer to the device-specific documentation to see how/if external boot is supported for a particular MCU.

If code in external memory needs to write EBI registers, this must be done in a way that avoids modifying EBI registers while external accesses are being performed, such as the following method:

- Copy the code that is doing the register writes (plus a return branch) to internal SRAM.
- Branch to internal SRAM to run this code, ending with a branch back to external flash memory.

22.5.2 Running with SDR (Single Data Rate) burst memories

This includes flash memory and SRAM with a compatible burst interface. $\overline{\text{BDIP}}$ is required only for some SDR memories. Figure 22-43 shows a block diagram of an MCU connected to a 32-bit SDR burst memory.



* May or may not be connected, depending on the memory used.

** Flash memories typically use one $\overline{\text{WE}}$ signal as shown. RAMs use 2 or 4 (16-bit or 32-bit)

Figure 22-43. MCU connected to SDR burst memory

See Figure 22-27 for an example of the timing of a typical Burst Read operation to an SDR burst memory. See Figure 22-15 for an example of the timing of a typical Single Write operation to SDR memory.

22.5.3 Running with asynchronous memories

The EBI also supports asynchronous memories. In this case, the CLKOUT, $\overline{\text{TS}}$, and $\overline{\text{BDIP}}$ pins are not used by the memory and bursting is not supported. However, the EBI still drives these outputs, and always drives and latches all signals at the positive edge of CLKOUT (i.e., there is no “asynchronous mode” for the EBI). The data timing is controlled by setting the SCY bits in the appropriate Option Register to the proper number of wait states to work with the access time of the asynchronous memory, just as done for a synchronous memory.

22.5.3.1 Example wait state calculation

This example applies to any chip-select memory, synchronous or asynchronous.

As an example, say we have a memory with 50 ns access time, and we are running the external bus at 66 MHz (CLKOUT period: 15.2 ns). Assume the input data spec for the MCU is 4 ns.

number of wait states = (access time) / (CLKOUT period) + (0 or 1) (depending on setup time)

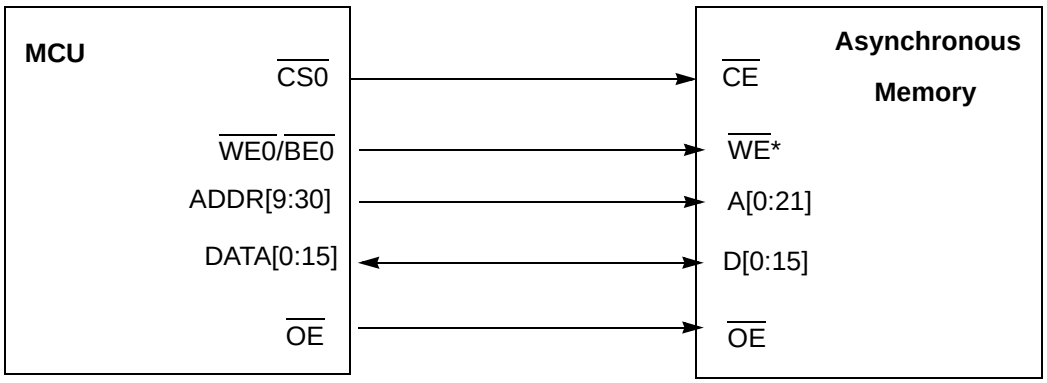
50/15.2 = 3 with 4.4 ns remaining (so we need at least 3 wait states, now check setup time)

15.2 – 4.4 = 10.8 ns (this is the achieved input data setup time)

Since actual input setup (10.8 ns) is greater than the input setup spec (4.0 ns), 3 wait states is sufficient. If the actual input setup was less than 4.0 ns, we would have to use 4 wait states instead.

22.5.3.2 Timing and connections for asynchronous memories

The connections to an asynchronous memory are the same as for a synchronous memory, except that the CLKOUT, $\overline{TS}$, and $\overline{BDIP}$ signals are not used. Figure 22-44 shows a block diagram of an MCU connected to an asynchronous memory.



* Flash memories typically use one $\overline{WE}$ signal as shown, RAMs use 2 or 4 (16-bit or 32-bit)

Figure 22-44. MCU connected to asynchronous memory

Figure 22-45 shows a timing diagram of a read operation to a 16-bit asynchronous memory using 3 wait states.

Figure 22-46 shows a timing diagram of a write operation to a 16-bit asynchronous memory using 3 wait states.

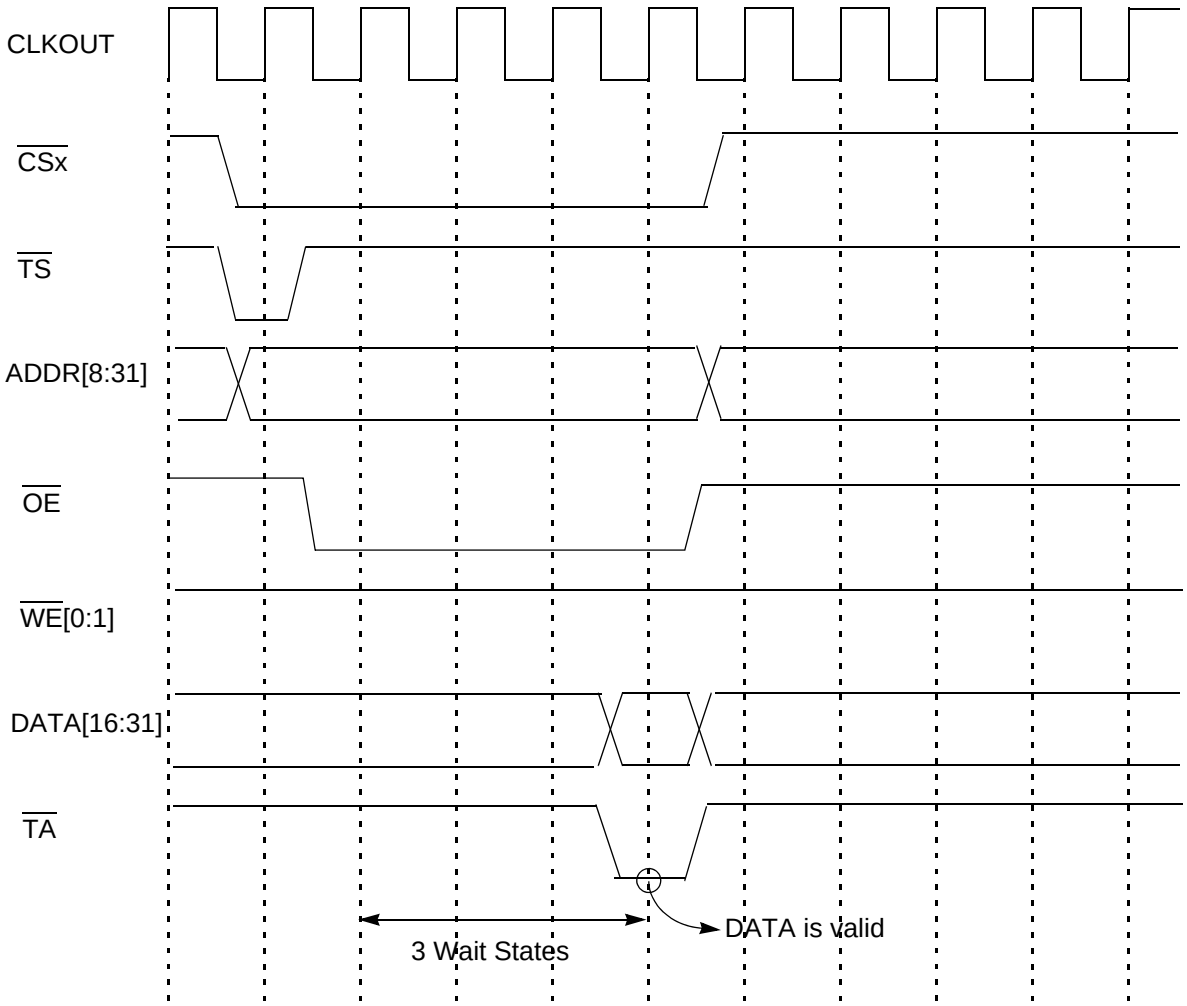


Figure 22-45. Read operation to asynchronous memory, three initial wait states

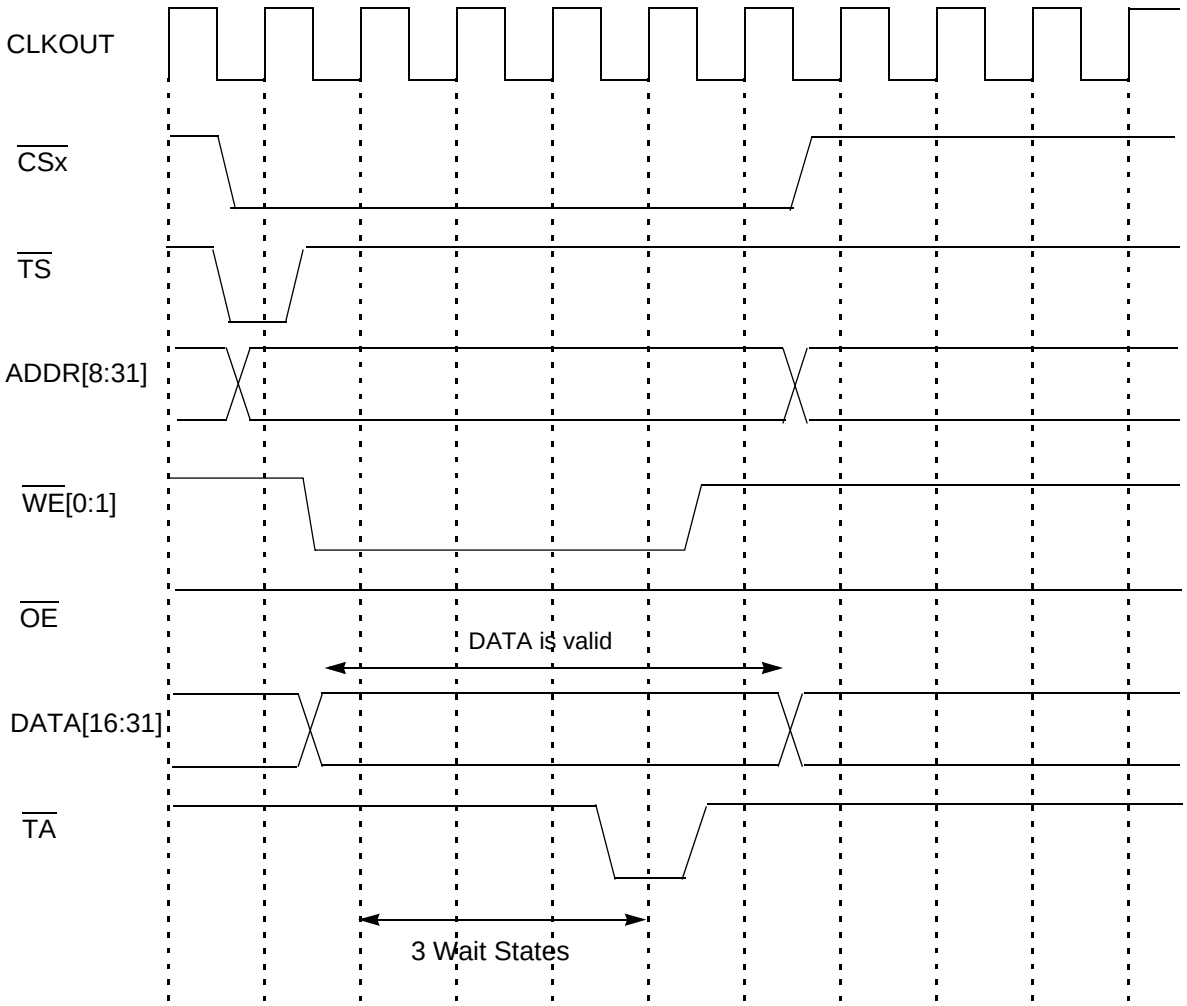
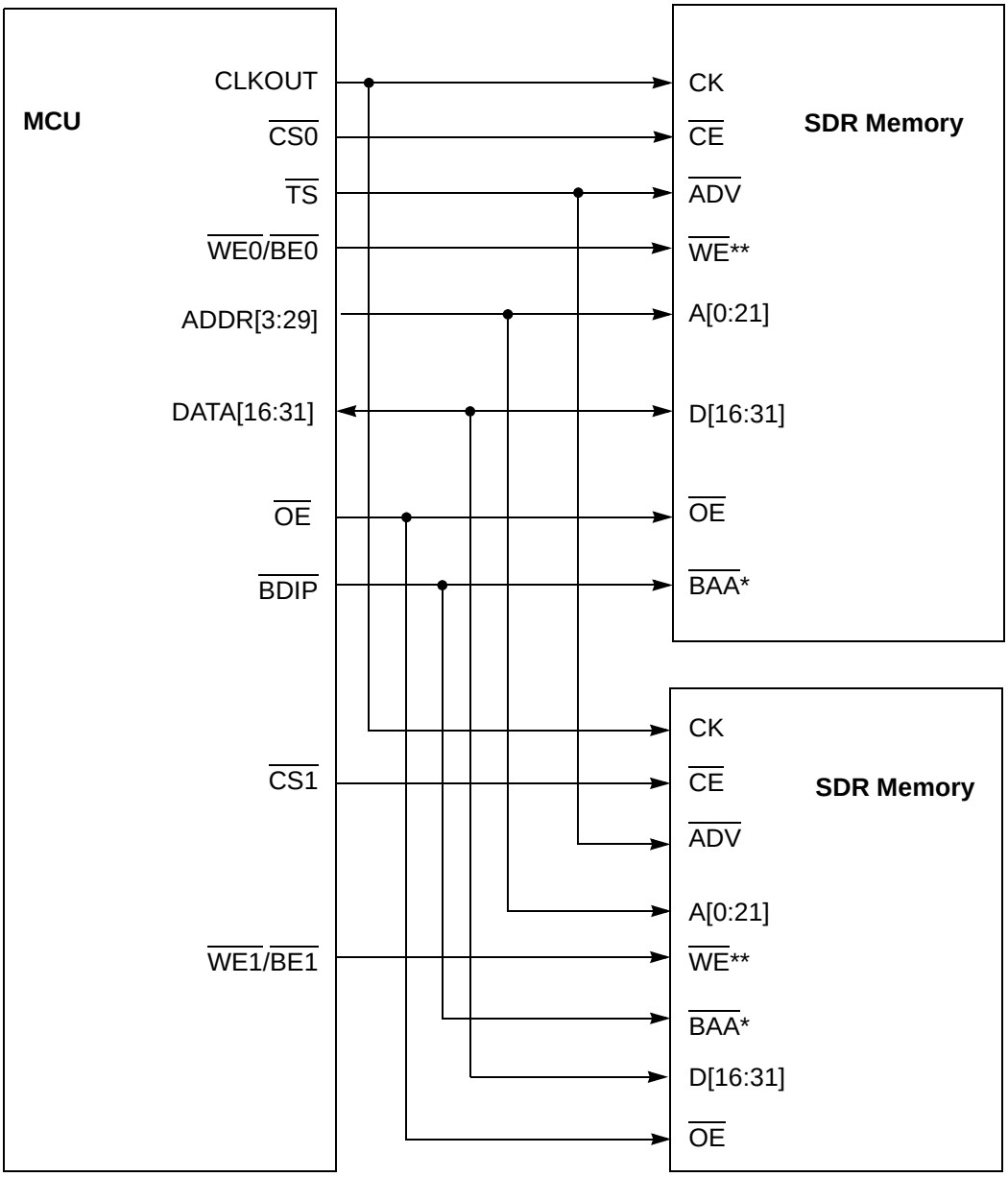


Figure 22-46. Write operation to asynchronous memory, three initial wait states

22.5.4 Connecting an MCU to multiple memories

The MCU can be connected to more than one memory at a time.

Figure 22-47 shows an example of how two memories could be connected to one MCU.



* May or may not be connected, depending on the memory used.

** Flash memories typically use one $\overline{WE}$ signal as shown, RAMs use 2 or 4 (16-bit or 32-bit)

Figure 22-47. MCU connected to multiple memories

22.5.5 EBI operation with reduced pinout MCUs

Some MCUs with this EBI may not have all the pins described in this document pinned out for a particular package. Some of the most common pins to be removed are DATA[16:31], arbitration pins ($\overline{BB}$, $\overline{BG}$, $\overline{BR}$),

and TSIZ[0:1]. This section describes how to configure dual-MCU systems for each of those scenarios, as well as describing limitations to EBI operation when other pins are missing ($\overline{TA}$, $\overline{TEA}$, $\overline{BDIP}$). More than one section may apply if the applicable pins are not present on one or both MCUs.

22.5.5.1 Connecting 16-bit MCU to 32-bit MCU (Master/Master or Master/Slave)

This scenario is straightforward. Connect DATA[0:15] between both MCUs, and configure both for 16-bit Data Bus Mode operation (DBM = 1 in EBI_MCR). Note that 32-bit external memories are not supported in this scenario.

22.5.5.2 Arbitration with no Arb Pins (Master/Slave only)

To implement a master/slave system with an MCU that has no arbitration pins ($\overline{BB}$, $\overline{BG}$, $\overline{BR}$), the user must configure the master MCU for internal arbitration (EARB = 0 in EBI_MCR) and the slave MCU for external arbitration (EARB = 1). Internally on an MCU with no arbitration pins, the $\overline{BR}$, $\overline{BG}$, and $\overline{BB}$ signals to the EBI will be tied negated. This means that the slave MCU will never receive bus grant asserted, so it will never attempt to start an access on the external bus. The master MCU will never receive bus request or bus busy asserted, so it will maintain ownership of the bus without any arbitration delays. In the erroneous case that the slave MCU executes internal code that attempts to access external address space, that access will never get external and will eventually time out in the slave MCU.

22.5.5.3 Transfer size with no TSIZ pins (Master/Master or Master/Slave)

Since there are no TSIZ pins to communicate transfer size from master MCU to slave MCU, the internal SIZE field of the EBI_MCR must be used on the slave MCU (by setting SIZEN = 1 in slave's EBI_MCR). Anytime the master MCU needs to read or write the slave MCU with a different transfer size than the current value of the slave's SIZE field, the master MCU must first write the slave's SIZE field with the correct size for the subsequent transaction.

22.5.5.4 No Transfer Acknowledge ($\overline{TA}$) pin

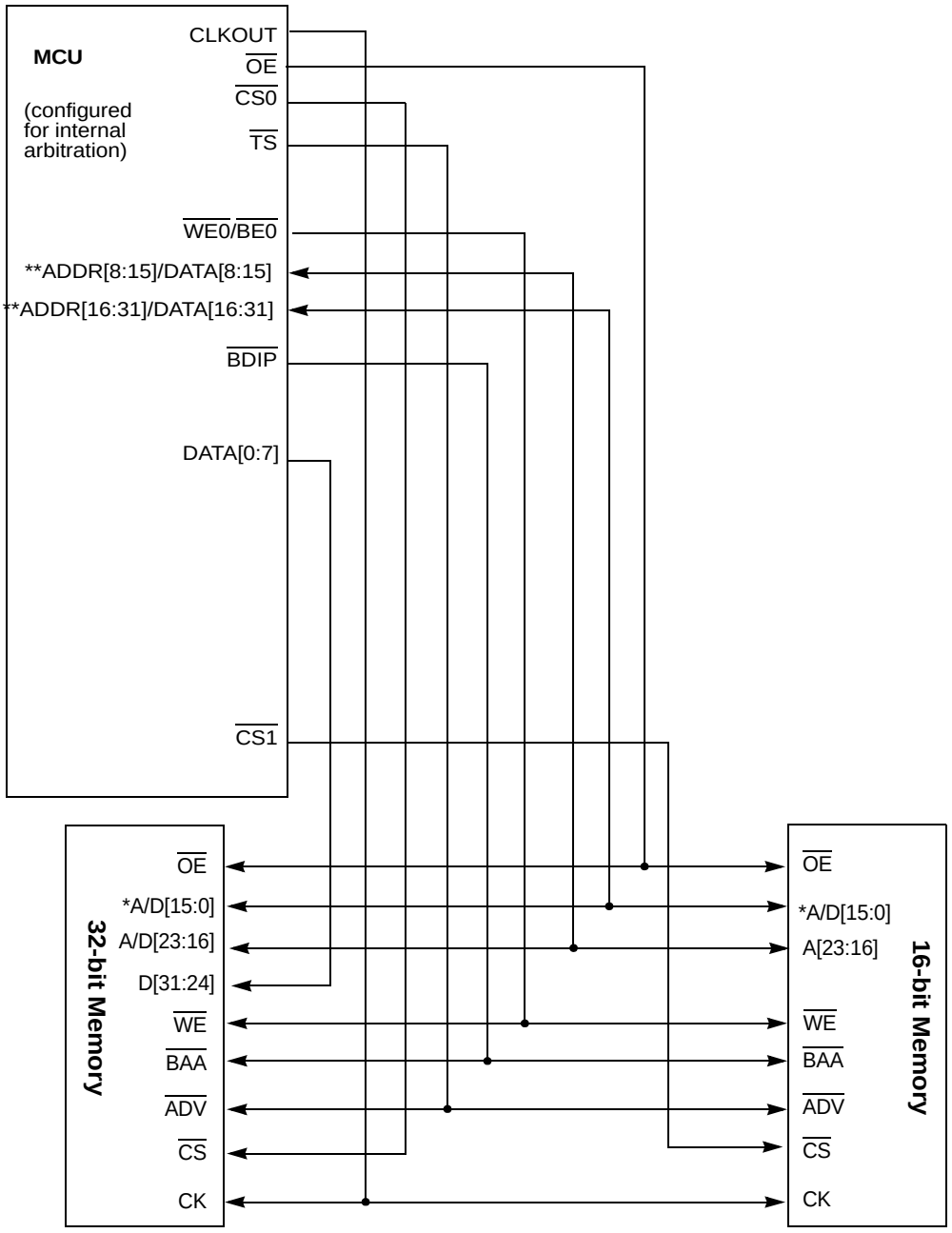
If an MCU has no $\overline{TA}$ pin available, this restricts the MCU to chip-select accesses only (no MCU $\rightarrow$ MCU transfers are possible). Non-chip-select accesses have no way for the EBI to know which cycle to latch the data. The EBI has no built-in protection to prevent non-chip-select accesses in this scenario; it is up to the user to make certain to set up chip-selects and external memories correctly, and ensure that all external accesses fall in a valid chip-select region.

22.5.5.5 No Transfer Error ($\overline{TEA}$) pin

If an MCU has no $\overline{TEA}$ pin available, this eliminates the feature of terminating an access with $\overline{TEA}$. This means if an access times out in the EBI bus monitor, the EBI (master) still terminates the access early, but there will be no external visibility of this termination, so the slave device might end up driving data much later, when a subsequent access is already underway. Therefore, the EBI bus monitor should be disabled when no $\overline{TEA}$ pin exists.

22.5.5.6 No Burst Data in Progress ($\overline{\text{BDIP}}$) pin

If an MCU has no $\overline{\text{BDIP}}$ pin available, this eliminates burst support only if the burstable memory being used requires $\overline{\text{BDIP}}$ to burst. Many external memories use a self-timed configurable burst mechanism that does not require a dynamic burst indicator. Check the applicable external memory specification to see if $\overline{\text{BDIP}}$ is required in your system.



* Most memories have the LSB signified by bit 0, this must match the LSB of the MCU, signified by bit 31. If the memory is word-addressable, other bits of the data bus can be used. For example, a 32-bit flash memory might need A[2] as the LSB, where A/D1 and A/D0 are not used for address decoding.

** These refer to the DATA pins of the MCU, which are used for both address and data functions in this system.

Figure 22-48. Address/Data Multiplexing with both 16-bit and 32-bit memories in single master system

22.5.6 Summary of differences from MPC5xx

Below is a summary list of the significant differences between this EBI and that of the MPC5xx parts.

- No memory controller support for external masters
  - Must configure each master in multi-master system to drive its own chip selects
  - Rationale: save complexity, no requirement for this feature
- Burst mechanism updated to be compatible with e200z core with 32-byte cache line
  - Rationale: required for performance and compatibility with e200z core
- Removed these variable timing attributes from Option Register:
  - CSNT, ACS, TRLX, EHTR
  - Rationale: reduces tester edgesets and complexity, no clear requirements for these features
- Removed reservation support on external bus
  - Rationale: reservation not supported on internal bus, useless to support on external
- Removed Address Type (AT), Write-Protect (WP), and dual-mapping features
  - Rationale: these functions can be replicated by Memory Management Unit (MMU) in e200z core
- Removed support for 8-bit ports
  - Rationale: reduces complexity and not required
- Removed boot chip-select operation
  - Rationale: on-chip Boot Assist Module (BAM) handles boot (and configuration of EBI registers)
- Open drain mode and pullup resistors no longer required for multi-master systems, extra cycle needed to switch between masters
  - Rationale: saves customer hassle for multi-master system setup, at negligible performance cost
- Address decoding for external master accesses uses 4-bit code to determine internal slave instead of straight address decode
  - Rationale: needed for compatibility with internal bridge address decoding and memory map
- Removed support for 3-master systems
  - Rationale: very difficult to manage with internal bridge address decoding method and keep memory maps unique; not an essential feature to justify complexity of supporting
- Removed LBDIP Base Register bit, now late $\overline{\text{BDIP}}$ assertion is default behavior
  - Rationale: unaware of any memories that require $\overline{\text{BDIP}}$ to assert earlier than LBDIP timing, so reduce number of CS control bits and complexity
- Modified arbitration protocol to require extra cycles when switching between masters
  - Rationale: could not use exact Oak protocol and make timing for full-speed operation; adding dead cycles to protocol allows bus to run full-speed in external master mode and makes this feature not limit overall EBI frequency
- Modified TSIZ[0:1] functionality to only indicate size of current transfer, not give information on ensuing transfers that may be part of the same atomic sequence

  - Rationale: simpler and more intuitive functionality, no clear requirement for anything else
- Added support for 32-bit coherent read and write non-chip-select accesses in 16-bit data bus mode
  - Rationale: some internal registers must be accessed all 32 bits at once to function as expected
- Added misaligned access support
  - Rationale: some eSys cores require use of misaligned accesses for optimum performance
- Added calibration access support
  - Rationale: support related device logic added to multiple eSys devices's, requested customer feature
- Added support for larger external address bus (up to 24 bits)
  - Rationale: support larger external memory sizes
- Added support for address/data multiplexing
  - Rationale: new feature to reduce minimum pin count
- Added support for using either half of data bus for 16-bit port transfers
  - Rationale: helps A/D muxed usability, while maintaining backwards compatibility

Chapter 23
Error Correction Status Module (ECSM)

23.1 Introduction

The Error Correction Status Module (ECSM) provides miscellaneous control functions for the device including program-visible information about the platform configuration and revision levels, a reset status register, and information on memory errors reported by error-correcting codes and/or generic access error information.

23.2 Features

The ECSM includes these features:

- Program-visible information on the platform device configuration and revision
- Miscellaneous Reset Status Register (MRSR)
- Registers for capturing information on memory errors
- Registers to specify the generation of single-bit and double-bit memory data inversions for test purposes if error-correcting codes are implemented
- Access address information for faulted memory accesses

23.3 Memory map and register description

This section details the programming model for the Error Correction Status Module. This is an on-platform 128-byte space mapped to the region serviced by an IPS bus controller. Some of the control registers have a 64-bit width. These 64-bit registers are implemented as two 32-bit registers, and include “H” and “L” suffixes, indicating the “high” and “low” portions of the control function.

The Error Correction Status Module does not include any logic that provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

23.3.1 Memory map

Table 23-3 is a 32-bit view of the ECSM’s memory map. The addresses are the offsets relative to the ESCM base address.

Table 23-1. ECSM module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	ECSM_0	0xFFF4_0000
	ECSM_1	
Decoupled Parallel Mode (DPM)	ECSM_0	0xFFF4_0000 (same as LSM)
	ECSM_1	0x8FF4_0000

Table 23-2. ECSM registers

Offset from ECSM_BASE	Register	Access[1]	Reset Value[2, 3]	Location
0x0000	Processor Core Type register (PCT)	R	0xE760	on page 676
0x0002	Revision register (REV)	R	0x0000	on page 676
0x0004	Platform XBAR Master Configuration (PLAMC) register	R	0x004F	on page 677
0x0006	Platform XBAR Slave Configuration (PLASC) register	R	0x808D	on page 677
0x0008	IPS Module Configuration (IMC)	R	0xC843_F800	on page 678
0x000C–0x000E	Reserved			
0x000F	Miscellaneous Reset Status Register (MRSR)	R	0x40	on page 679
0x0010–0x001E	Reserved			
0x001F	Miscellaneous Interrupt Register (MIR)	R/W	0x00	on page 679
0x0020	Reserved			
0x0024	Miscellaneous User-Defined Control Register (MUDCR)	R/W	0x4000_0000	on page 680
0x0028–0x0042	Reserved			
0x0043	ECC Configuration Register (ECR)	R/W	0x00	on page 683
0x0044–0x0046	Reserved			
0x0047	ECC Status Register (ESR)	R/W	0x00	on page 683
0x0048–0x0049	Reserved			
0x004A	ECC Error Generation Register (EEGR)	R/W	0x0000	on page 685
0x004C–0x004F	Reserved			
0x0050	Flash ECC Address Register (FEAR)	R	UUUU_UUUU	on page 688
0x0055	Reserved			
0x0056	Flash ECC Master Number Register (FEMR)	R	UUUU_UUUU	on page 689
0x0057	Flash ECC Attributes Register (FEAT)	R	UUUU_UUUU	on page 690
0x0058	Flash ECC Data Register High (FEDRH)	R	UUUU_UUUU	on page 691
0x005C	Flash ECC Data Register Low (FEDRL)	R	UUUU_UUUU	on page 691
0x0060	RAM ECC Address Register (REAR)	R	UUUU_UUUU	on page 692
0x0064	Reserved			
0x0065	RAM ECC Syndrome Register (RESR)	R	UU	on page 692
0x0066	RAM ECC Master Number Register (REMR)	R	UU	on page 694
0x0067	RAM ECC Attributes (REAT) register	R	UU	on page 694
0x0068	RAM ECC Data Register High (REDRH)	R	UUUU_UUUU	on page 695

Table 23-2. ECSM registers (continued)

Offset from ECSM_BASE	Register	Access[1]	Reset Value[2, 3]	Location
0x006C	RAM ECC Data Register Low (REDRL)	R	UUUU_UUUU	on page 695
0x0070–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

Table 23-3. ECSM 32-bit memory map

ECSM Offset	Register		
0x0000	Processor Core Type (PCT)	Revision (REV)	
0x0004	Platform XBAR Master Configuration (PLAMC) register	Platform XBAR Slave Configuration (PLASC) register	
0x0008	IPS Module Configuration (IMC)		
0x000C	Reserved		Misc Reset Status (MRSR)
0x0010	Reserved		
0x0014	Reserved		
0x0018	Reserved		
0x001C	Reserved		Misc Interrupt (MIR)
0x0020	Reserved		
0x0024	Miscellaneous User-Defined Control Register (MUDCR)		
0x0028	Reserved		
0x002C – 0x003C	Reserved		
0x0040	Reserved		ECC Configuration (ECR)
0x0044	Reserved		ECC Status (ESR)
0x0048	Reserved	ECC Error Generation (EEGR)	
0x004C	Reserved		
0x0050	Flash ECC Address (FEAR)		
0x0054	Reserved	Flash ECC Master (FEMR)	Flash ECC Attributes (FEAT)
0x0058	Flash ECC Data High (FEDRH)		

Table 23-3. ECSM 32-bit memory map (continued)

ECSM Offset	Register			
0x005C	Flash ECC Data Low (FEDRL)			
0x0060	RAM ECC Address (REAR)			
0x0064	Reserved	RAM ECC Syndrome (RESR)	RAM ECC Master (REMR)	RAM ECC Attributes (REAT)
0x0068	RAM ECC Data High (REDRH)			
0x006C	RAM ECC Data Low (REDRL)			
0x0070 - 0x007C	Reserved			

23.3.2 Registers description

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, *writes to the programming model must match the size of the register*, for example, an *n*-bit register only supports *n*-bit writes. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

23.3.2.1 Processor Core Type (PCT) register

The PCT is a 16-bit read-only register specifying the architecture of the processor core in the MPC5675K. See Figure 23-1 and Table 23-4 for the Processor Core Type definition.

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PCT															
W																
Reset	1	1	1	0	0	1	1	1	0	1	1	0	0	0	0	0

Figure 23-1. Processor Core Type (PCT) register

Table 23-4. PCT field descriptions

Field	Description
PCT	Processor Core Type. 0xE760 indicates z760 Power Architecture core.

23.3.2.2 Revision (REV) register

The REV register specifies a revision number. It can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 23-2 and Table 23-5 for the REV definition.

Address: Base + 0x0002 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REV															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-2. Revision (REV) register

Table 23-5. REV field descriptions

Field	Description
REV	Revision This field is a software-visible revision number.

23.3.2.3 Platform XBAR Master Configuration (PLAMC) register

The PLAMC identifies the presence/absence of bus master connections to the platform's AMBA-AHB Crossbar Switch (XBAR). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	AMC7	AMC6	AMC5	AMC4	AMC3	AMC2	AMC1	AMC0
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1

Figure 23-3. Platform XBAR Master Configuration (PLAMC) register

Table 23-6. PLAMC field descriptions

Field	Description
AMC*n*	XBAR master configuration 0 Bus master connection to XBAR input port *n* is absent. 1 Bus master connection to XBAR input port *n* is present.

23.3.2.4 Platform XBAR Slave Configuration (PLASC) register

The PLASC identifies the presence/absence of bus slave connections to the platform's AMBA-AHB Crossbar Switch (XBAR), plus a 1-bit flag defining the internal platform datapath width (DP64). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address: Base + 0x0006 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP64	0	0	0	0	0	0	0	ASC7	ASC6	ASC5	ASC4	ASC3	ASC2	ASC1	ASC0
W																
Reset	1	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1

Figure 23-4. Platform XBAR Slave Configuration (PLASC) register

Table 23-7. PLASC field descriptions

Field	Description
DP64	Datapath width 0 Platform datapath width is 32 bits. 1 Platform datapath width is 64 bits.
ASC*n*	XBAR slave configuration 0 Bus slave connection to XBAR output port *n* is absent. 1 Bus slave connection to XBAR output port *n* is present.

23.3.2.5 IPS Module Configuration (IMC) register

The IMC is a 32-bit read-only register identifying the presence or absence of the 11 low-order IPS peripheral modules connected to the primary slave bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 23-5 and Table 23-8 for the IPS Module Configuration definition.

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MC[0:15]															
W																
Reset	1	1	0	0	1	0	0	0	0	1	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MC[16:31]															
W																
Reset	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 23-5. IPS Module Configuration (IMC) register

Table 23-8. IMC field descriptions

Field	Description
MC[0:31]	IPS Module Configuration 0 An IPS module connection to decoded slot *n* is absent. 1 An IPS module connection to decoded slot *n* is present. Module mappings are as follows: 0 PBRIDGE 1 XBAR 4 MPU 9 SEMA4 14 SWT 15 STM 16 ECSM 17 DMA 18 INTC 19 FEC 20 Coherency Module

23.3.2.6 Miscellaneous Reset Status Register (MRSR)

The Miscellaneous Reset Status Register (MRSR) contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the MRSR, reflecting the cause of the most recent reset as signaled by the device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 23-6 and Table 23-9 for the MRSR definition.

Address: Base + 0x000F Access: User read-only

	0	1	2	3	4	5	6	7
R	POR	OFPLR	0	0	0	0	0	0
W								
Reset	0	1	0	0	0	0	0	0

Figure 23-6. Miscellaneous Reset Status Register (MRSR)

Table 23-9. MRSR field descriptions

Field	Description
POR	Power-On Reset 1 The last recorded event was caused by a power-on reset (based on a device input signal).
OFPLR	Device Input Reset 1 The last recorded event was a reset caused by an off-platform reset.

23.3.2.7 Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the Miscellaneous Interrupt Register (MIR). This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the MIR must be explicitly cleared. See Figure 23-7 and Table 23-10 for the MIR definition.

Address: Base + 0x001F Access: User read/write

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	FB2AI	FB2SI	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c		
Reset	0	0	0	0	0	0	0	0

Figure 23-7. Miscellaneous Interrupt Register (MIR)

Table 23-10. Miscellaneous Interrupt (MIR) field descriptions

Field	Description
FB0AI	Flash Bank 0 Abort Interrupt 0 A flash bank 0 abort has not occurred. 1 A flash bank 0 abort has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.
FB0SI	Flash Bank 0 Stall Interrupt 0 A flash bank 0 stall has not occurred. 1 A flash bank 0 stall has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.
FB1AI	Flash Bank 1 Abort Interrupt 0 A flash bank 1 abort has not occurred. 1 A flash bank 1 abort has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.
FB1SI	Flash Bank 1 Stall Interrupt 0 A flash bank 1 stall has not occurred. 1 A flash bank 1 stall has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.
FB2AI	Flash Bank 2 Abort Interrupt 0 A flash bank 2 abort has not occurred. 1 A flash bank 2 abort has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.
FB2SI	Flash Bank 2 Stall Interrupt 0 A flash bank 2 stall has not occurred. 1 A flash bank 2 stall has occurred. The interrupt request is cleared by writing a 1 to this bit. Writing a 0 has no effect.

23.3.2.8 Miscellaneous User-Defined Control Register (MUDCR)

The Miscellaneous User-Defined Control Register (MUDCR) provides a program-visible register for user-defined control functions. The contents of this register is output from the ECSM to other modules where the user-defined control functions are implemented. See Figure 23-8 and Table 23-11 for the MUDCR definition.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MUDCR0	MUDCR1	0	0	0	0	0	DRCLR13	DRCLR12	DRCLR11	DRCLR10	DRCLR9	DRCLR8	DRCLR7	DRCLR6	DRCLR5
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DRCLR4	DRCLR3	DRCLR2	DRCLR1	DRCLR0	ACCERR	WBEN	RBEN	FXSBE7	0	0	0	FXSBE3	FXSBE2	0	FXSBE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-8. Miscellaneous User-Defined Control Register (MUDCR)

Table 23-11. MUDCR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MUDCR0</td><td>XBAR_2 arbitration mode select. This bit allows setting XBAR_2 into the round-robin mode of arbitration.
0 XBAR_2 is in fixed priority mode.
1 XBAR_2 is in round robin mode.</td></tr>
<tr><td>MUDCR1</td><td>PRAM wait-state control. This bit selects whether the PRAM controller inserts one wait state into every read access made to the RAM arrays.
0 The PRAM controller operates as a 0-wait state controller.
1 The PRAM controller operates as a 1-wait state controller.</td></tr>
<tr><td>DRCLRn</td><td>DMA Request Clear n. Write a 1 to the bit and then write a 0 to clear a stuck DMA request. Bits are allocated as shown.
<table>
<tr><th>Bit</th><th>DMA channel</th><th>Peripheral</th><th>Bit</th><th>DMA channel</th><th>Peripheral</th></tr>
<tr><td>DRCLR0 (bit 20)</td><td>20</td><td>ADC0</td><td>DRCLR7 (bit 13)</td><td>17</td><td>eTimer_1 channel 1</td></tr>
<tr><td>DRCLR1 (bit 19)</td><td>21</td><td>ADC1</td><td>DRCLR8 (bit 12)</td><td>12</td><td>FlexPWM_0 comp_val</td></tr>
<tr><td>DRCLR2 (bit 18)</td><td>33</td><td>ADC2</td><td>DRCLR9 (bit 11)</td><td>13</td><td>FlexPWM_0 capt</td></tr>
<tr><td>DRCLR3 (bit 17)</td><td>34</td><td>ADC3</td><td>DRCLR10 (bit 10)</td><td>26</td><td>FlexPWM_1 comp_val</td></tr>
<tr><td>DRCLR4 (bit 16)</td><td>14</td><td>eTimer_0 channel 0</td><td>DRCLR11 (bit 9)</td><td>27</td><td>FlexPWM_1 capt</td></tr>
<tr><td>DRCLR5 (bit 15)</td><td>15</td><td>eTimer_0 channel 1</td><td>DRCLR12 (bit 8)</td><td>39</td><td>FlexPWM_2 comp_val</td></tr>
<tr><td>DRCLR6 (bit 14)</td><td>16</td><td>eTimer_1 channel 0</td><td>DRCLR13 (bit 7)</td><td>40</td><td>FlexPWM_2 capt</td></tr>
</table>
Note: If a module generates more DMA requests than the number that DMA module is programmed to serve, the last DMA request stalls at the DMA's input. Stopping the module will not clear this extra DMA request. To clear the request, write a 1 and then write a 0 to the corresponding bit after completion of the DMA operation or before enabling the next DMA operation.
Note: The DRCLRn bits are implemented on cut2 of the MPC5675K, but not on cut1.</td></tr>
<tr><td>ACCERR</td><td>Accumulate error. This bit determines whether an error response for the first half of the write burst is accumulated to the second half of the write burst or discarded. In order to complete the burst, the FEC interface to the system bus responds by indicating that the first half of the burst completed without error before it actually writes the data so that it can fetch the second half of the write data from the FIFO. When actually written onto the system bus, the first half of the write burst can have an error. Because this half initially responded without an error to the FIFO, the error is discarded or accumulated with the error response for the second half of the burst.
0 Any error to the first half of the write burst is discarded.
1 Any actual error response to the first half of the write burst is accumulated in the second half's response. In other words, an error response to the first half is seen in the response to the second half, even if the second half does not have an error.</td></tr>
<tr><td>WBEN</td><td>Global write burst enable to XBAR slave port designated by FXSBEn
0 Write bursting to all XBAR slave ports is disabled.
1 Write bursting is enabled to any XBAR slave port whose FXSBEn bit is asserted.</td></tr>
<tr><td>RBEN</td><td>Global read burst enable from XBAR slave port designated by FXSBEn
0 Read bursting from all XBAR slave ports is disabled.
1 Read bursting is enabled from any XBAR slave port whose FXSBEn bit is asserted.</td></tr>
</table>

Table 23-11. MUDCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FXSBEn</td><td>FEC XBAR slave burst enable. FXSBEn enables bursting by the FEC interface to the XBAR slave port controlled by that respective FXSBEn bit. If FXSBEn is asserted, then that XBAR slave port enabled by the bit can accept the bursts allowed by RBEN and WBEN. Otherwise, the FEC interface will not burst to the XBAR slave port controlled by that respective FXSBEn bit. Read bursts from that XBAR slave port are enabled by RBEN. Write bursts to that XBAR slave port are enabled by WBEN.
<table>
<tr><th>Bit</th><th>Slave port</th></tr>
<tr><td>FXSBE0 (bit 31)</td><td>Slave Port S0</td></tr>
<tr><td>FXSBE1 (bit 30)</td><td>not implemented on MPC5675K</td></tr>
<tr><td>FXSBE2 (bit 29)</td><td>Slave Port S2</td></tr>
<tr><td>FXSBE3 (bit 28)</td><td>Slave Port S3</td></tr>
<tr><td>FXSBE4 (bit 27)</td><td>not implemented on MPC5675K</td></tr>
<tr><td>FXSBE5 (bit 26)</td><td>not implemented on MPC5675K</td></tr>
<tr><td>FXSBE6 (bit 25)</td><td>not implemented on MPC5675K</td></tr>
<tr><td>FXSBE7 (bit 24)</td><td>Slave Port S7</td></tr>
</table>
Note: Slave Port assignment is subject to LSM/DPM mode configuration. See Chapter 9, Multi-Layer AHB Crossbar Switch (XBAR), for more information.</td></tr>
</table>

23.3.3 ECC registers

There are a number of registers for the sole purpose of reporting and logging memory failures. These registers include the following:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Flash ECC Address Register (FEAR)
- Flash ECC Master Number Register (FEMR)
- Flash ECC Attributes Register (FEAT)
- Flash ECC Data Register (FEDR)
- RAM ECC Address Register (REAR)
- RAM ECC Syndrome Register (RESR)
- RAM ECC Master Number Register (REMR)
- RAM ECC Attributes Register (REAT)
- RAM ECC Data Register (REDR)

The details on the ECC registers are provided in the subsequent sections.

23.3.3.1 ECC Configuration Register (ECR)

The ECR is an 8-bit control register for specifying which types of memory errors are reported to the FCCU.

See Figure 23-9 and Table 23-12 for the ECR definition.

Address: Base + 0x0043 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-9. ECC Configuration (ECR) Register

Table 23-12. ECR field descriptions

Field	Description
ER1BR	Enable RAM 1-bit Reporting to the FCCU 0 Reporting of single-bit RAM corrections is disabled. 1 Reporting of single-bit RAM corrections is enabled.
EF1BR	Enable Flash 1-bit Reporting to the FCCU 0 Reporting of single-bit flash corrections is disabled. 1 Reporting of single-bit flash corrections is enabled.
ERNCR	Enable RAM Non-Correctable Reporting to the FCCU 0 Reporting of non-correctable RAM errors is disabled. 1 Reporting of non-correctable RAM errors is enabled.
EFNCR	Enable Flash Non-Correctable Reporting to the FCCU 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is enabled.

NOTE

If you choose to use the ECC error reporting functionality, the user software should enable this immediately after initialization of the core registers such as IVOR and MMU. An ECC error occurring before this functionality is enabled will not be reported. Note that the flash block also needs to enable this functionality by setting UT0[SBCE].

23.3.3.2 ECC Status Register (ESR)

The ECC Status Register (ESR) is an 8-bit control register for signaling which types of properly enabled ECC events have been detected. The ESR signals the last properly enabled memory event to be detected. ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection, and the combination of the two as defined by the following boolean equations:

```
ECSM_ECC1BIT_IRQ
    =  ECR[ER1BR] &  ESR[R1BC]              // ram, 1-bit correction
    |  ECR[EF1BR] &  ESR[F1BC]              // flash, 1-bit correction
ECSM_ECCRNCR_IRQ
    =  ECR[ERNCR] &  ESR[RNCE]              // ram, noncorrectable error
ECSM_ECCFNCR_IRQ
    =  ECR[EFNCR] &  ESR[FNCE]              // flash, noncorrectable error
ECSM_IRQ35
    =  ECSM_ECCRNCR_IRQ                     // ram, noncorrectable error
```

```
    |  ECSM_ECCFNCR_IRQ                    // flash, noncorrectable error
ECSM_IRQ36
    =  ECSM_ECC1BIT_IRQ                  // 1-bit correction
    |  ECSM_ECC2BIT_IRQ                  // noncorrectable error
```

where the combination of a correctly enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of a properly enabled ECC event. If there is a pending ECC interrupt and another properly enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus configuring the assertion of only a single flag.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

See Figure 23-10 and Table 23-13 for the ESR definition.

Address: Base + 0x0047 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W			w1c	w1c			w1c	w1c
Reset	0	0	0	0	0	0	0	0

Figure 23-10. ECC Status Register (ESR)

Table 23-13. ESR field descriptions

Field	Description
R1BC	RAM 1-bit Correction 0 No reportable single-bit RAM correction has been detected. 1 A reportable single-bit RAM correction has been detected. This bit can only be set if ECR[ER1BR] is asserted. The occurrence of a properly enabled single-bit RAM correction generates an ECSM ECC interrupt request. The address, attributes, and data are also captured in the REAR, RESR, REMR, REAT, and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
F1BC	Flash 1-bit Correction 0 No reportable single-bit flash correction has been detected. 1 A reportable single-bit flash correction has been detected. This bit can only be set if ECR[EF1BR] is asserted. The occurrence of a properly enabled single-bit flash correction generates an ECSM ECC interrupt request. The address, attributes, and data are also captured in the FEAR, FEMR, FEAT, and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.

Table 23-13. ESR field descriptions (continued)

Field	Description
RNCE	RAM Non-Correctable Error 0 No reportable non-correctable RAM error has been detected. 1 A reportable non-correctable RAM error has been detected. The occurrence of a properly enabled non-correctable RAM error generates an ECSM ECC interrupt request. The faulting address, attributes, and data are also captured in the REAR, RESR, REMR, REAT, and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
FNCE	Flash Non-Correctable Error 0 No reportable non-correctable flash error has been detected. 1 A reportable non-correctable flash error has been detected. The occurrence of a properly enabled non-correctable flash error generates an ECSM ECC interrupt request. The faulting address, attributes, and data are also captured in the FEAR, FEMR, FEAT, and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

NOTE

The ECSM also provides ECC error injection registers through which non-correctable ECC errors can be injected in the RAM. The injection of errors via these register would trigger a non-correctable ECC interrupt (IRQ35).

Since the non-correctable error occurs in RAM/flash memory, the reaction is determined by the RGM_FERD[D_FL_ECC_RCC] bit. Depending on this bit, the ECC event may cause a reset, safe mode entry, or an interrupt.

To generate IRQ 35, the non-correctable ECC interrupt RGM (IRQ35), FCCU reset and ECC reset should be programmed as SAFE mode entry and not as reset, using the FERD register. When a corrupted ECC location is read, IRQ35 + machine check exception + NMI exception is generated. Thus, the handlers need to be written in a way that all exceptions and interrupts are serviced properly, and the handlers bring the MPC5675K back to DRUN mode from SAFE mode and also clear all the flags, including flags set in the FCCU.

23.3.3.3 ECC Error Generation Register (EEGR)

The ECC Error Generation Register (EEGR) is a 16-bit control register used to force the generation of single-bit and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the RAM, similar capabilities exist for the flash, i.e., the ability to program the nonvolatile memory with single-bit or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

Single-bit errors are corrected but not reported. Double-bit errors are reported, but by definition are non-correctable.

See Figure 23-11 and Table 23-14 for the EEGR definition.

Address: Base + 0x004A Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRCAP	0	FRC1BI	FR11BI	0	0	FRCNCI	FR1NCI	0	ERRBIT[0:6]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-11. ECC Error Generation Register (EEGR)

Table 23-14. EEGR field descriptions

Field	Description
FRCAP	Force Platform RAM Error Injection Access Protection 0 All masters are able to generate RAM ECC errors via the EEGR register. 1 Only the master defined as having hmaster = 0 (usually the core) can generate RAM ECC errors via the EEGR register. The assertion of this bit ensures that RAM data inversions can only occur from the master module with the master ID of 0. Since this is usually the core, this protects the RAM from errant or multiple simultaneous attempted data inversions from other master modules, and in the case of a multi-core system, ensures that only one core can issue a RAM data inversion. In LSM, the system is seen as having only one core. In DPM, only the core in lake_A, which is core_0, can perform inversions by setting this bit. The reset value of the bit is 0 and as a result, RAM data inversions can be requested from any master module. Software must ensure the proper setting of this bit.
FRC1BI	Force RAM Continuous 1-bit Data Inversions 0 No RAM continuous 1-bit data inversions are generated. 1 1-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[0:6], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to correctly re-enable the error generation logic.

Table 23-14. EEGR field descriptions (continued)

Field	Description
FR11BI	Force RAM One 1-bit Data Inversion 0 No RAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[0:6], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.
FRCNCI	Force RAM Continuous Non-Correctable Data Inversions 0 No RAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.
FR1NCI	Force RAM One Non-Correctable Data Inversions 0 No RAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[0:6] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.

Table 23-14. EEGR field descriptions (continued)

Field	Description
ERRBIT	Error Bit Position. The vector defines the bit position that is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The platform RAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 64-bit RAM implementation. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] is inverted. if ERRBIT = 1, then RAM[1] is inverted. . . . if ERRBIT = 63, then RAM[63] is inverted. if ERRBIT = 64,then ECC Parity[0] is inverted. if ERRBIT = 65,then ECC Parity[1] is inverted. . . . if ERRBIT = 70,then ECC Parity[6] is inverted. if ERRBIT = 71,then ECC Parity[7] of the even bank is inverted. For ERRBIT values between 72 and 98, no bit position is inverted. To accommodate address bus inversions, the ERRBIT values start at 99 as defined: if ERRBIT = 99, then ADDR[3] is inverted. if ERRBIT = 100, then ADDR[4] is inverted. . . . if ERRBIT = 126, then ADDR[30] is inverted. if ERRBIT = 127, then ADDR[31] is inverted.

Inversions of the address bus must be configured as non-correctable for the inversion to properly function. Address bus inversions defined as 1-bit inversions are ignored.

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion is generated.

The only allowable values for the four control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in unpredictable operations.

23.3.3.4 Flash ECC Address Register (FEAR)

The FEAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-12 and Table 23-15 for the Flash ECC Address Register definition.

Address: Base + 0x0050 Access: User read-only

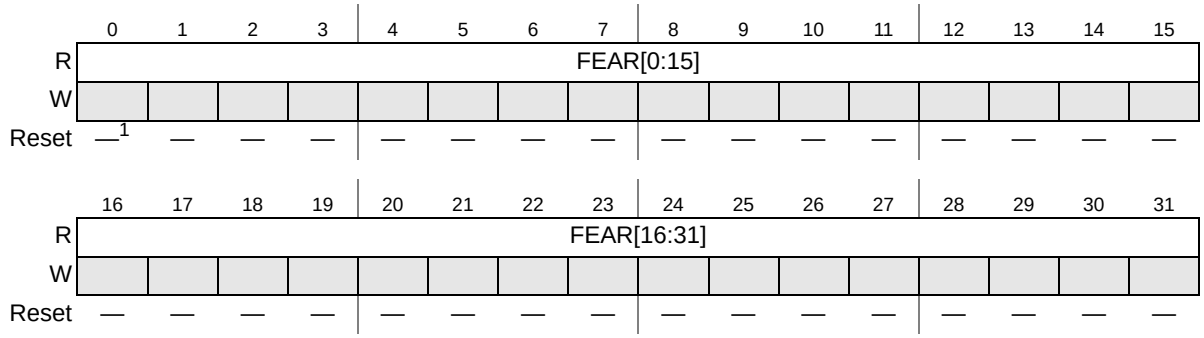


Figure 23-12. Flash ECC Address Register (FEAR)

1 Value is undefined at reset.

Table 23-15. FEAR field descriptions

Field	Description
FEAR[0:31]	Flash ECC Address Register This 32-bit bitfield contains the faulting access address of the last properly enabled flash ECC event.

23.3.3.5 Flash ECC Master Number Register (FEMR)

The Flash ECC Master Number Register (FEMR) is an 8-bit register for capturing the XBAR bus master number of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-13 and Table 23-16 for the FEMR definition.

Address: Base + 0x0056 Access: User read-only

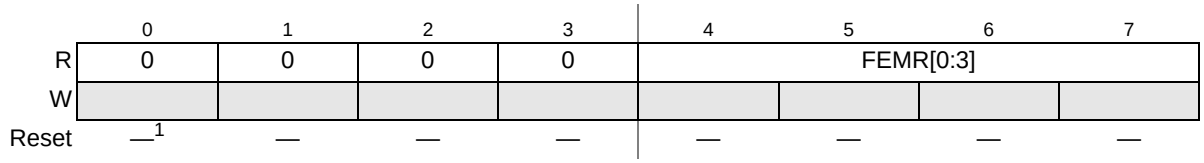


Figure 23-13. Flash ECC Master Number Register (FEMR)

1 Value is undefined at reset.

Table 23-16. FEMR field descriptions

Field	Description
FEMR[0:3]	Flash ECC Master Number Register This bitfield contains the XBAR bus master number of the faulting access of the last properly enabled flash ECC event. 0000 z7d_0 core complex (LSM and DPM) z7d_1 core complex (LSM only) 0001 z7d_1 core complex (DPM only) 0010 DMA_0 (LSM and DPM) DMA_1 (LSM only) 0011 FlexRay (LSM and DPM) 0100 FEC (LSM and DPM) 0101 PDI (LSM and DPM) 0110 DMA_1 (DPM only) 0111 Reserved 1000 z7d_0 core Nexus (LSM and DPM) z7d_1 core Nexus (LSM only) 1001 z7d_1 core Nexus (DPM only)

23.3.3.6 Flash ECC Attributes (FEAT) register

The Flash ECC Attributes (FEAT) register is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-14 and Table 23-17 for the FEAT definition.

Address: Base + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	Write	Size[0:2]			Protection[0:3]			
W								
Reset	—[1]	—	—	—	—	—	—	—

Figure 23-14. Flash ECC Attributes (FEAT) Register

[1] Value is undefined at reset.

Table 23-17. FEAT field descriptions

Field	Description
Write	AMBA-AHB HWRITE 0 AMBA-AHB read access. 1 AMBA-AHB write access.
Size[0:2]	AMBA-AHB HSIZE[0:2] 000 8-bit AMBA-AHB access. 001 16-bit AMBA-AHB access. 010 32-bit AMBA-AHB access. 1*xx* Reserved.

Table 23-17. FEAT field descriptions (continued)

Field	Description
Protection[0:3]	AMBA-AHB HPROT[0:3] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable Protection[1]: Mode 0 User mode 1 Supervisor mode Protection[0]: Type 0 I-Fetch 1 Data

23.3.3.7 Flash ECC Data Register High/Low (FEDRH/L)

The Flash ECC Data Register High/Low (FEDRH/L) is a pair of 32-bit registers that capture the data associated with the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register (ESR) to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-15 and Table 23-18 for the FEDR definition.

Address: Base + 0x0058 (FEDRH)
Base + 0x005C (FEDRL)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEDR[0:15]															
W																
Reset	—[1]	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEDR[16:31]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 23-15. Flash ECC Data Register High/Low (FEDRH/L)

1 Value is undefined at reset.

Table 23-18. FEDRH/L field descriptions

Field	Description
FEDR[0:31]	Flash ECC Data Register This bitfield contains the data associated with the faulting access of the last properly enabled flash ECC event. The register contains the data value taken directly from the data bus.

23.3.3.8 RAM ECC Address Register (REAR)

The RAM ECC Address Register (REAR) is a 32-bit register for capturing the address of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-16 and Table 23-19 for the REAR definition.

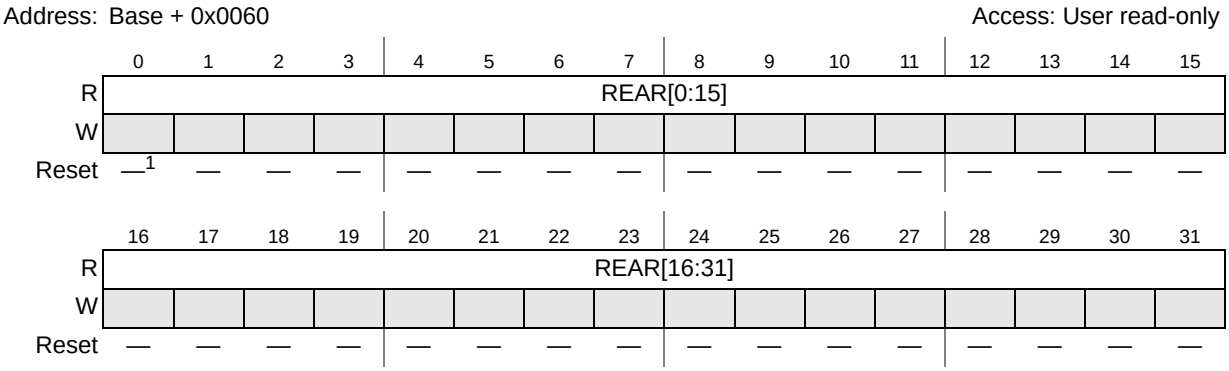


Figure 23-16. RAM ECC Address Register (REAR)

1 Value is undefined at reset.

Table 23-19. REAR field descriptions

Field	Description
REAR[0:31]	RAM ECC Address Register. This bitfield contains the faulting access address of the last properly enabled RAM ECC event.

23.3.3.9 RAM ECC Syndrome Register (RESR)

The RAM ECC Syndrome Register (RESR) is an 8-bit register for capturing the error syndrome of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the REAR, RESR, REMR, REAT, and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-17 and Table 23-20 for the RESR definition.

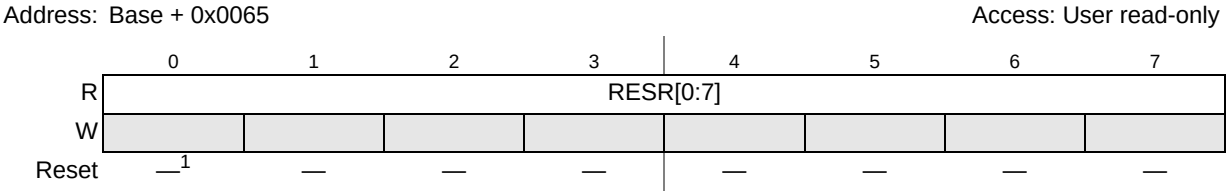


Figure 23-17. RAM ECC Syndrome Register (RESR)

1 Value is undefined at reset.

Table 23-20. RESR field descriptions

Field	Description
RESR[0:7]	RAM ECC Syndrome Register This 8-bit syndrome field includes optimized syndrome encoding for the entire 72-bit (64-bit data + 8 bit ECC) code word of each bank for single-bit errors. Syndrome values for non-correctable errors are not defined. For correctable single-bit errors, the mapping shown in Table 23-21 associates the 8 bits of the syndrome with the data or ECC bit in error.

Table 23-21. RAM Syndrome Mapping for Single-Bit Correctable Errors

PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error
0x00	No Error	0x34	DATA[41]	0x8A	DATA[14]
0x01	ECC[0]	0x37	DATA[24]	0x8C	DATA[7]
0x02	ECC[1]	0x38	DATA[29]	0x91	DATA[33]
0x04	ECC[2]	0x40	ECC[6]	0x92	DATA[44]
0x07	DATA[42]	0x43	DATA[1]	0x94	DATA[21]
0x08	ECC[3]	0x45	DATA[49]	0x98	DATA[37]
0x0B	DATA[13]	0x46	DATA[32]	0x9E	DATA[2]
0x0D	DATA[26]	0x49	DATA[36]	0xA1	DATA[8]
0x0E	DATA[12]	0x4A	DATA[6]	0xA2	DATA[58]
0x10	ECC[4]	0x4C	DATA[5]	0xA4	DATA[25]
0x13	DATA[50]	0x51	DATA[60]	0xA7	DATA[46]
0x15	DATA[4]	0x52	DATA[48]	0xA	DATA[45]
0x16	DATA[22]	0x54	DATA[47]	0xB0	DATA[55]
0x19	DATA[16]	0x58	DATA[31]	0xC1	DATA[0]
0x1A	DATA[17]	0x61	DATA[35]	0xC2	DATA[62]
0x1C	DATA[57]	0x62	DATA[52]	0xC4	DATA[27]
0x20	ECC[5]	0x64	DATA[23]	0xC8	DATA[43]
0x23	DATA[54]	0x68	DATA[39]	0xD0	DATA[63]
0x25	DATA[10]	0x70	DATA[53]	0xD9	DATA[9]
0x26	DATA[18]	0x75	DATA[28]	0xE0	DATA[59]
0x29	DATA[20]	0x80	ECC[7]	0xEA	DATA[19]
0x2A	DATA[15]	0x83	DATA[3]	0xF8	DATA[11]
0x2C	DATA[61]	0x85	DATA[51]	Others	Multiple bit error
0x31	DATA[56]	0x86	DATA[34]		
0x32	DATA[40]	0x89	DATA[38]		

23.3.3.10 RAM ECC Master Number Register (REMR)

The RAM ECC Master Number Register (REMR) is an 8-bit register for capturing the XBAR bus master number of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-18 and Table 23-22 for the REMR definition.

Address: Base + 0x0066 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR[0:3]			
W								
Reset	—[1]	—	—	—	—	—	—	—

Figure 23-18. RAM ECC Master Number Register (REMR)

[1] Value is undefined at reset.

Table 23-22. REMR field descriptions

Field	Description
REMR[0:3]	RAM ECC Master Number Register This bitfield contains the XBAR bus master number of the faulting access of the last correctly enabled RAM ECC event. Logical master IDs are defined in Table 9-1.

23.3.3.11 RAM ECC Attributes (REAT) register

The RAM ECC Attributes (REAT) register is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register (ESR) to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-19 and Table 23-23 for the REAT register definition.

Address: Base + 0x0067 Access: User read-only

	0	1	2	3	4	5	6	7
R	Write	Size[0:2]			Protection[0:3]			
W								
Reset	—[1]	—	—	—	—	—	—	—

Figure 23-19. RAM ECC Attributes (REAT) Register

[1] Value is undefined at reset.

Table 23-23. REAT field descriptions

Field	Description
Write	AMBA-AHB HWRITE 0 AMBA-AHB read access. 1 AMBA-AHB write access.
Size[0:2]	AMBA-AHB HSIZE[0:2] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
Protection[0:3]	AMBA-AHB HPROT[0:3] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable Protection[1]: Mode 0 User mode 1 Supervisor mode Protection[0]: Type 0 I-Fetch 1 Data

23.3.3.12 RAM ECC Data Register High/Low (REDRH/L)

The RAM ECC Data Register High/Low (REDRH/L) is a pair of 32-bit registers that capture the data associated with the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register (ECR), an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register (ESR) to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 23-20 and Table 23-24 for the REDR definition.

Address: Base + 0x0068 (REDRH)
Base + 0x006C (REDRL)

Access: User read-only

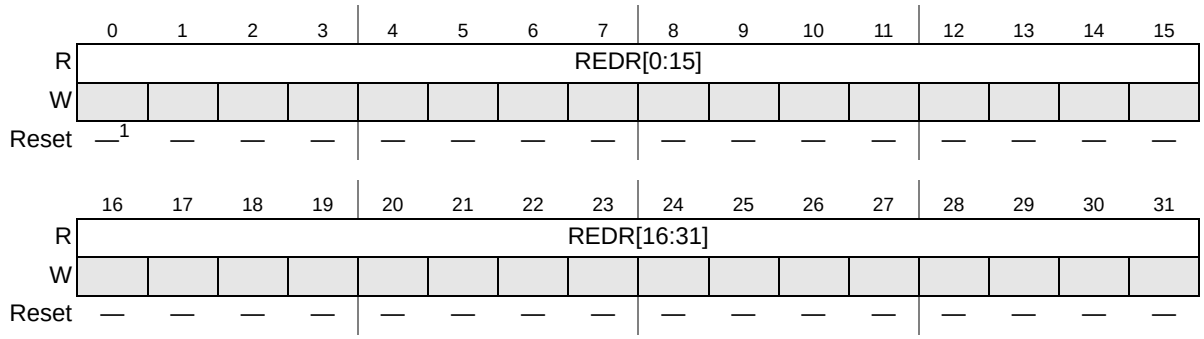


Figure 23-20. RAM ECC Data Register High/Low (REDR)

[1] Value is undefined at reset.

Table 23-24. REDRH/L field descriptions

Field	Description
REDR[0:31]	RAM ECC Data Register This bitfield contains the data associated with the faulting access of the last properly enabled RAM ECC event. The register contains the data value taken directly from the data bus.

Chapter 24
Enhanced Motor Control Timer (eTimer)

24.1 Introduction

24.1.1 Overview

The MPC5675K device provides three eTimer modules:

- eTimer_0
- eTimer_1
- eTimer_2

Each eTimer module contains six identical counter/timer channels. Each 16-bit counter/timer channel contains a prescaler, a counter, a load register, a hold register, two queued capture registers, two compare registers, two compare preload registers, and four control registers. In addition, eTimer_0 provides an additional watchdog timer function.

NOTE

This document uses the terms “Timer” and “Counter” interchangeably because the counter/timers may perform either or both tasks.

The Load register provides the initialization value to the counter when the counter’s terminal value has been reached. For true modulo counting, the counter can also be initialized by the CMPLD1 or CMPLD2 registers.

The Hold register captures the counter’s value when other counters are being read. This feature supports the reading of cascaded counters coherently.

The Capture registers enable an external signal to take a “snap shot” of the counter’s current value.

The COMP1 and COMP2 registers provide the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

The Prescaler provides different time bases useful for clocking the counter/timer.

The Counter provides the ability to count internal or external events.

Within the eTimer module (set of six timer/counter channels) the input pins are shareable.

24.1.2 Features

The eTimer module design includes these features:

- Six 16-bit counters/timers
- Count up/down
- Counters are cascadable

- Enhanced programmable up/down modulo counting
- Max count rate equals peripheral clock/2 for external clocks
- Max count rate equals peripheral clock for internal clocks
- Count once or repeatedly
- Counters are preloadable
- Compare registers are preloadable
- Counters can share available input pins
- Separate prescaler for each counter
- Each counter has capture and compare capability
- Continuous and single shot capture for enhanced speed measurement
- DMA support of capture registers and compare registers
- 32-bit watchdog capability to detect stalled quadrature counting
- OFLAG comparison for safety critical applications
- Programmable operation during debug mode and stop mode
- Programmable input filter
- Counting start can be synchronized across counters
- Interrupt generation

24.1.3 Module block diagram

The eTimer block diagram is shown in Figure 24-1.

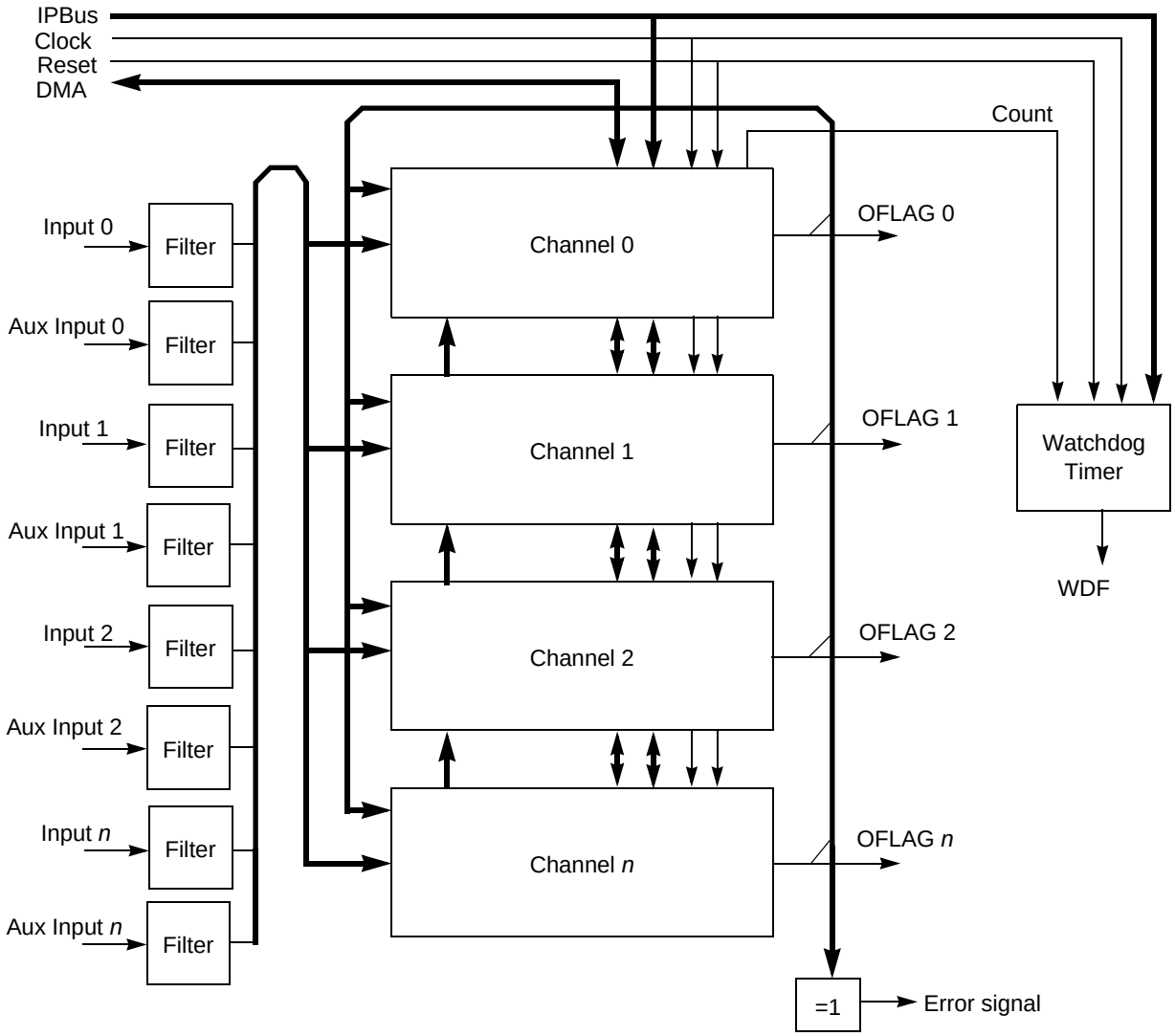


Figure 24-1. eTimer block diagram

24.1.4 Channel block diagram

Each of the timer/counter channels within the eTimer are shown in Figure 24-2.

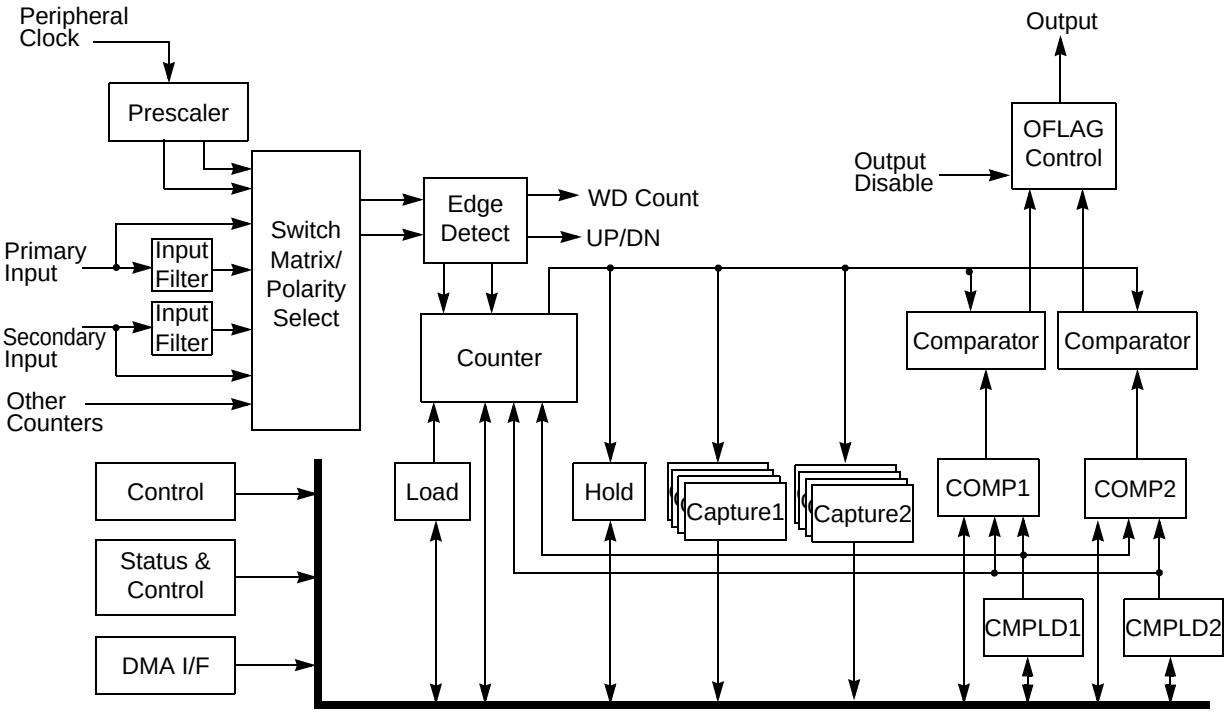


Figure 24-2. eTimer channel block diagram

24.2 External signal descriptions

The eTimer module has six external signals that can be used either as inputs or outputs. In addition, four auxiliary inputs are available. The eTimer also interfaces to the peripheral bus.

24.2.1 TIO[5:0]—Timer Input/Outputs

These pins can be independently configured to be either timer input sources or output flags.

24.2.2 TAI[3:0]—Timer Auxiliary Inputs

These pins act as alternate input choices for the timer channels.

24.3 Memory map and registers

Table 24-1 shows the base addresses for the eTimer modules. Addresses are the same for Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). Table 24-2 shows the memory map for the eTimer program-visible registers.

Table 24-1. eTimer module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM) Decoupled Parallel Mode (DPM)	eTimer_0	0xFFE1_8000
	eTimer_1	0xFFE1_C000
	eTimer_2	0xFFE2_0000

24.3.1 Module memory map

Table 24-2. eTimer memory map

Offset from ETIMER_BASE	Register	Access[1]	Reset Value[2]	Location
Channel 0				
0x0000	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x0002	Compare Register 2 (COMP2)	R/W	0x0000	on page 705
0x0004	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x0006	Capture Register 2 (CAPT2)	R	0x0000	on page 706
0x0008	Load Register (LOAD)	R/W	0x0000	on page 707
0x000A	Hold Register (HOLD)	R	0x0000	on page 707
0x000C	Counter Register (CNTR)	R/W	0x0000	on page 707
0x000E	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x0010	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x0012	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x0014	Status Register (STS)	R/W	0x0000	on page 713
0x0016	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x0018	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x001A	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x001C	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x001E	Input Filter Register (FILT)	R/W	0x0000	on page 718
Channel 1				
0x0020	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x0022	Compare Register 2 (COMP2)	R/W	0x0000	on page 705
0x0024	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x0026	Capture Register 2 (CAPT2)	R	0x0000	on page 706
0x0028	Load Register (LOAD)	R/W	0x0000	on page 707
0x002A	Hold Register (HOLD)	R	0x0000	on page 707

Table 24-2. eTimer memory map (continued)

Offset from ETIMER_BASE	Register	Access[1]	Reset Value[2]	Location
0x002C	Counter Register (CNTR)	R/W	0x0000	on page 707
0x002E	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x0030	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x0032	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x0034	Status Register (STS)	R/W	0x0000	on page 713
0x0036	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x0038	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x003A	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x003C	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x003E	Input Filter Register (FILT)	R/W	0x0000	on page 718
Channel 2				
0x0040	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x0042	Compare Register 2 (COMP2)	R/W	0x0000	on page 705
0x0044	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x0046	Capture Register 2 (CAPT2)	R	0x0000	on page 706
0x0048	Load Register (LOAD)	R/W	0x0000	on page 707
0x004A	Hold Register (HOLD)	R	0x0000	on page 707
0x004C	Counter Register (CNTR)	R/W	0x0000	on page 707
0x004E	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x0050	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x0052	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x0054	Status Register (STS)	R/W	0x0000	on page 713
0x0056	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x0058	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x005A	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x005C	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x005E	Input Filter Register (FILT)	R/W	0x0000	on page 718
Channel 3				
0x0060	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x0062	Compare Register 2 (COMP2)	R/W	0x0000	on page 705
0x0064	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x0066	Capture Register 2 (CAPT2)	R	0x0000	on page 706

Table 24-2. eTimer memory map (continued)

Offset from ETIMER_BASE	Register	Access[1]	Reset Value[2]	Location
0x0068	Load Register (LOAD)	R/W	0x0000	on page 707
0x006A	Hold Register (HOLD)	R	0x0000	on page 707
0x006C	Counter Register (CNTR)	R/W	0x0000	on page 707
0x006E	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x0070	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x0072	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x0074	Status Register (STS)	R/W	0x0000	on page 713
0x0076	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x0078	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x007A	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x007C	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x007E	Input Filter Register (FILT)	R/W	0x0000	on page 718
Channel 4				
0x0080	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x0082	Compare Register 2 (COMP2)	R/W	0x0000	on page 705
0x0084	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x0086	Capture Register 2 (CAPT2)	R	0x0000	on page 706
0x0088	Load Register (LOAD)	R/W	0x0000	on page 707
0x008A	Hold Register (HOLD)	R	0x0000	on page 707
0x008C	Counter Register (CNTR)	R/W	0x0000	on page 707
0x008E	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x0090	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x0092	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x0094	Status Register (STS)	R/W	0x0000	on page 713
0x0096	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x0098	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x009A	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x009C	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x009E	Input Filter Register (FILT)	R/W	0x0000	on page 718
Channel 5				
0x00A0	Compare Register 1 (COMP1)	R/W	0x0000	on page 705
0x00A2	Compare Register 2 (COMP2)	R/W	0x0000	on page 705

Table 24-2. eTimer memory map (continued)

Offset from ETIMER_BASE	Register	Access[1]	Reset Value[2]	Location
0x00A4	Capture Register 1 (CAPT1)	R	0x0000	on page 706
0x00A6	Capture Register 2 (CAPT2)	R	0x0000	on page 706
0x00A8	Load Register (LOAD)	R/W	0x0000	on page 707
0x00AA	Hold Register (HOLD)	R	0x0000	on page 707
0x00AC	Counter Register (CNTR)	R/W	0x0000	on page 707
0x00AE	Control Register 1 (CTRL1)	R/W	0x0000	on page 708
0x00B0	Control Register 2 (CTRL2)	R/W	0x0000	on page 710
0x00B2	Control Register 3 (CTRL3)	R/W	0x0F00	on page 712
0x00B4	Status Register (STS)	R/W	0x0000	on page 713
0x00B6	Interrupt and DMA Enable Register (INTDMA)	R/W	0x0000	on page 714
0x00B8	Comparator Load Register 1 (CMPLD1)	R/W	0x0000	on page 715
0x00BA	Comparator Load Register 2 (CMPLD2)	R/W	0x0000	on page 715
0x00BC	Compare and Capture Control Register (CCCTRL)	R/W	0x0000	on page 716
0x00BE	Input Filter Register (FILT)	R/W	0x0000	on page 718
0x00C0–0x00FF	Reserved			
Watchdog registers[3]				
0x0100	Watchdog Timeout Low Register (WDTOL)	R/W	0x0000	on page 718
0x0102	Watchdog Timeout High Register (WDTOH	R/W	0x0000	on page 718
0x0104–0x10B	Reserved			
0x010C	Channel Enable Register (ENBL)	R/W	0x003F	on page 719
0x010E–0x10F	Reserved			
0x0110	DMA Request 0 Select Register (DREQ0)	R/W	0x0000	on page 720
0x0112	DMA Request 1 Select Register (DREQ1)	R/W	0x0000	on page 720
0x0114–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 Watchdog timer is implemented on eTimer_0 only. On eTimer_1 and eTimer_2, this register space is reserved.

24.3.2 Register descriptions

The address of a register is the sum of a base address and an address offset. The base address is defined at the chip level and the address offset is defined at the module level. There are a set of registers for each timer channel, a set for the watchdog timer on eTimer0, and a set of configuration registers.

24.3.3 Timer channel registers

These registers are repeated for each timer channel. The base address of channel 0 is the same as the base address of the eTimer module as a whole. The base address of channel 1 is 0x20. This is the base address of the eTimer module plus an offset based on the number of bytes of registers in a timer channel. The base address of each subsequent timer channel is equal to the base address of the previous channel plus this same offset of 0x20.

24.3.3.1 Compare Register 1 (COMP1)

The read/write Compare Register 1 (COMP1) stores the value used for comparison with the counter value. This register is not byte-accessible. More explanation on the use of COMP1 can be found in Section 24.4.2.12.1, Usage of Compare registers.

Address: Base + 0x0000 (Channel 0: COMP1) Base + 0x0060 (Channel 3: COMP1)
Base + 0x0020 (Channel 1: COMP1) Base + 0x0080 (Channel 4: COMP1)
Base + 0x0040 (Channel 2: COMP1) Base + 0x00A0 (Channel 5: COMP1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	COMP1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-3. Compare Register 1 (COMP1)

Table 24-3. COMP1 field descriptions

Field	Description
COMP1[15:0]	Value used for comparison with the counter value.

24.3.3.2 Compare Register 2 (COMP2)

The read/write Compare Register 2 (COMP2) stores the value used for comparison with the counter value. This register is not byte-accessible. More explanation on the use of COMP2 can be found in Section 24.4.2.12.1, Usage of Compare registers.

Address: Base + 0x0002 (Channel 0: COMP2) Base + 0x0062 (Channel 3: COMP2)
Base + 0x0022 (Channel 1: COMP2) Base + 0x0082 (Channel 4: COMP2)
Base + 0x0042 (Channel 2: COMP2) Base + 0x00A2 (Channel 5: COMP2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	COMP2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-4. Compare Register 2 (COMP2)

Table 24-4. COMP2 field descriptions

Field	Description
COMP2[15:0]	Value used for comparison with the counter value.

24.3.3.3 Capture Register 1 (CAPT1)

The read-only Capture Register 1 (CAPT1) stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT1MODE bits. This is actually a two-deep FIFO and not a single register. Once the CAPT1 FIFO is full, new data is discarded until a register read occurs to empty a spot in the FIFO.

This register is not byte-accessible.

Address: Base + 0x0004 (Channel 0: CAPT1) Base + 0x0064 (Channel 3: CAPT1)
Base + 0x0024 (Channel 1: CAPT1) Base + 0x0084 (Channel 4: CAPT1)
Base + 0x0044 (Channel 2: CAPT1) Base + 0x00A4 (Channel 5: CAPT1) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPT1[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-5. Capture Register 1 (CAPT1)

Table 24-5. CAPT1 field descriptions

Field	Description
CAPT1[15:0]	Value captured from the counter.

24.3.3.4 Capture Register 2 (CAPT2)

The read-only Capture Register 2 (CAPT2) stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT2MODE bits. This is actually a two-deep FIFO and not a single register. Once the CAPT2 FIFO is full, new data is discarded until a register read occurs to empty a spot in the FIFO.

This register is not byte-accessible.

Address: Base + 0x0006 (Channel 0: CAPT2) Base + 0x0066 (Channel 3: CAPT2)
Base + 0x0026 (Channel 1: CAPT2) Base + 0x0086 (Channel 4: CAPT2)
Base + 0x0046 (Channel 2: CAPT2) Base + 0x00A6 (Channel 5: CAPT2) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPT2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-6. Capture Register 2 (CAPT2)

Table 24-6. CAPT2 field descriptions

Field	Description
CAPT2[15:0]	Value captured from the counter.

24.3.3.5 Load Register (LOAD)

The read/write Load Register (LOAD) stores the value used to initialize the counter. This register is not byte-accessible.

Address: Base + 0x0008 (Channel 0: LOAD) Base + 0x0068 (Channel 3: LOAD)
Base + 0x0028 (Channel 1: LOAD) Base + 0x0088 (Channel 4: LOAD)
Base + 0x0048 (Channel 2: LOAD) Base + 0x00A8 (Channel 5: LOAD) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LOAD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-7. Load Register (LOAD)

Table 24-7. LOAD field descriptions

Field	Description
LOAD[15:0]	Value used to initialize the counter.

24.3.3.6 Hold Register (HOLD)

This read only register stores the counter's value whenever any of the other counters within a module are read. This is used to support coherent reading of cascaded counters. In addition, this read only register stores the counter's value if any of the Compare and Capture Control Registers (CCCTRL) within a module are read.

The read-only Hold Register (HOLD) stores the counter's value either by reading a CNTR register and also updated when reading a CCCTRL register.

Address: Base + 0x000A (Channel 0: HOLD) Base + 0x006A (Channel 3: HOLD)
Base + 0x002A (Channel 1: HOLD) Base + 0x008A (Channel 4: HOLD)
Base + 0x004A (Channel 2: HOLD) Base + 0x00AA (Channel 5: HOLD) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HOLD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-8. Hold Register (HOLD)

Table 24-8. HOLD field descriptions

Field	Description
HOLD[15:0]	Stores the counter's value whenever any of the other counters within a module are read.

24.3.3.7 Counter Register (CNTR)

The read/write Counter Register (CNTR) is the counter for this channel of the timer module. This register is not byte-accessible.

Address: Base + 0x000C (Channel 0: CNTR) Base + 0x006C (Channel 3: CNTR)
Base + 0x002C (Channel 1: CNTR) Base + 0x008C (Channel 4: CNTR)
Base + 0x004C (Channel 2: CNTR) Base + 0x00AC (Channel 5: CNTR) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTR[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-9. Counter Register (CNTR)

Table 24-9. CNTR field descriptions

Field	Description
CNTR[15:0]	Counter for this channel of the timer module.

24.3.3.8 Control Register 1 (CTRL1)

Address: Base + 0x000E (Channel 0: CTRL1) Base + 0x006E (Channel 3: CTRL1)
Base + 0x002E (Channel 1: CTRL1) Base + 0x008E (Channel 4: CTRL1)
Base + 0x004E (Channel 2: CTRL1) Base + 0x00AE (Channel 5: CTRL1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTMODE[2:0]			PRISRC[4:0]					ONCE	LENGTH	DIR	SECSRC[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-10. Control Register 1 (CTRL1)

Table 24-10. CTRL1 field descriptions

Field	Description
CNTMODE	Count Mode. These bits control the basic counting and behavior of the counter. 000 No Operation. 001 Count edges of primary source. Rising edges counted when PIPS = 0. Falling edges counted when PIPS = 1. If primary count source is IP bus clock, only rising edges are counted regardless of PIPS value. 010 Count rising and falling edges of primary source. IP Bus clock divide-by-1 cannot be used as a primary count source in edge count mode. 011 Count rising edges of primary source while secondary input high active. 100 Quadrature count mode, uses primary and secondary sources. 101 Count primary source edges, secondary source specifies count direction (1 = down). Rising edges counted when PIPS = 0. Falling edges counted when PIPS = 1. 110 Edge of secondary source triggers primary count until compare. 111 Cascaded counter mode, up/down. Primary count source must be set to one of the counter outputs.

Table 24-10. CTRL1 field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>PRISRC</td><td>Primary Count Source. These bits select the primary count source.
<table>
<tr><th>PRISRC</th><th>Meaning</th><th>PRISRC</th><th>Meaning</th></tr>
<tr><td>00000</td><td>Counter #0 input pin</td><td>10000</td><td>Counter #0 output</td></tr>
<tr><td>00001</td><td>Counter #1 input pin</td><td>10001</td><td>Counter #1 output</td></tr>
<tr><td>00010</td><td>Counter #2 input pin</td><td>10010</td><td>Counter #2 output</td></tr>
<tr><td>00011</td><td>Counter #3 input pin</td><td>10011</td><td>Counter #3 output</td></tr>
<tr><td>00100</td><td>Counter #4 input pin</td><td>10100</td><td>Counter #4 output</td></tr>
<tr><td>00101</td><td>Counter #5 input pin</td><td>10101</td><td>Counter #5 output</td></tr>
<tr><td>00110</td><td>Reserved</td><td>10110</td><td>Reserved</td></tr>
<tr><td>00111</td><td>Reserved</td><td>10111</td><td>Reserved</td></tr>
<tr><td>01000</td><td>Auxiliary input #0 pin</td><td>11000</td><td>IP Bus clock divide by 1 prescaler</td></tr>
<tr><td>01001</td><td>Auxiliary input #1 pin</td><td>11001</td><td>IP Bus clock divide by 2 prescaler</td></tr>
<tr><td>01010</td><td>Auxiliary input #2 pin</td><td>11010</td><td>IP Bus clock divide by 4 prescaler</td></tr>
<tr><td>01011</td><td>Auxiliary input #3 pin</td><td>11011</td><td>IP Bus clock divide by 8 prescaler</td></tr>
<tr><td>01100</td><td>Reserved</td><td>11100</td><td>IP Bus clock divide by 16 prescaler</td></tr>
<tr><td>01101</td><td>Reserved</td><td>11101</td><td>IP Bus clock divide by 32 prescaler</td></tr>
<tr><td>01110</td><td>Reserved</td><td>11110</td><td>IP Bus clock divide by 64 prescaler</td></tr>
<tr><td>01111</td><td>Reserved</td><td>11111</td><td>IP Bus clock divide by 128 prescaler</td></tr>
</table>
Note: A timer selecting its own output as its primary count source is not a legal choice. The result is no counting.</td></tr>
<tr><td>ONCE</td><td>Count Once. This bit selects continuous or one-shot counting mode.
0 Count repeatedly.
1 Count until compare and then stop. When output mode 0x4 is used, the counter re-initializes after reaching the COMP1 value and continues to count to the COMP2 value, then stops.</td></tr>
<tr><td>LENGTH</td><td>Count Length. This bit determines whether the counter counts to the compare value and then re-initializes itself to the value specified in the LOAD, CMPLD1, or CMPLD2 registers, or the counter continues counting past the compare value, to the binary rollover.
0 Continue counting to rollover.
1 Count until compare, then reinitialize.
The value that the counter is reinitialized with depends on the settings of CLC1 and CLC2. If neither of these indicates the counter is to be loaded from one of the CMPLD registers, then the LOAD register reinitializes the counter upon matching either COMP register. If one of CLC1 or CLC2 indicates that the counter is to be loaded from one of the CMPLD registers, then the counter reinitializes to the value in the appropriate CMPLD register upon a match with the appropriate COMP register. If both of the CLC1 and CLC2 fields indicate that the counter is to be loaded from the CMPLD registers, then CMPLD1 has priority if both compares happen at the same value.
When output mode 0x4 is used, alternating values of COMP1 and COMP2 are used to generate successful comparisons. For example, the counter counts until COMP1 value is reached, re-initializes, then counts until COMP2 value is reached, re-initializes, then counts until COMP1 value is reached, etc.</td></tr>
</table>

Table 24-10. CTRL1 field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DIR</td><td>Count Direction. This bit selects either the normal count direction up, or the reverse direction, down.
0 Count up.
1 Count down.</td></tr>
<tr><td>SECSRC</td><td>Secondary Count Source. These bits identify the source to be used as a count command or timer command. The selected input can trigger the timer to capture the current value of the CNTR register. The selected input can also be used to specify the count direction. The polarity of the signal can be inverted by the SIPS bit of the CTRL2 register. A counter may not select its own output as the secondary source.
<table>
<tr><th>Value</th><th>Meaning</th><th>Value</th><th>Meaning</th></tr>
<tr><td>00000</td><td>Counter #0 input pin</td><td>01101</td><td>Reserved</td></tr>
<tr><td>00001</td><td>Counter #1 input pin</td><td>01110</td><td>Reserved</td></tr>
<tr><td>00010</td><td>Counter #2 input pin</td><td>01111</td><td>Reserved</td></tr>
<tr><td>00011</td><td>Counter #3 input pin</td><td>10000</td><td>Counter #0 output</td></tr>
<tr><td>00100</td><td>Counter #4 input pin</td><td>10001</td><td>Counter #1 output</td></tr>
<tr><td>00101</td><td>Counter #5 input pin</td><td>10010</td><td>Counter #2 output</td></tr>
<tr><td>00110</td><td>Reserved</td><td>10011</td><td>Counter #3 output</td></tr>
<tr><td>00111</td><td>Reserved</td><td>10100</td><td>Counter #4 output</td></tr>
<tr><td>01000</td><td>Auxiliary input #0 pin</td><td>10101</td><td>Counter #5 output</td></tr>
<tr><td>01001</td><td>Auxiliary input #1 pin</td><td>10110</td><td>Reserved</td></tr>
<tr><td>01010</td><td>Auxiliary input #2 pin</td><td>10111</td><td>Reserved</td></tr>
<tr><td>01011</td><td>Auxiliary input #3 pin</td><td>11000–11111</td><td>Reserved</td></tr>
<tr><td>01100</td><td>Reserved</td><td colspan="2"></td></tr>
</table>
</td></tr>
</table>

24.3.3.9 Control Register 2 (CTRL2)

Address: Base + 0x0010 (Channel 0: CTRL2) Base + 0x0070 (Channel 3: CTRL2)
Base + 0x0030 (Channel 1: CTRL2) Base + 0x0090 (Channel 4: CTRL2)
Base + 0x0050 (Channel 2: CTRL2) Base + 0x00B0 (Channel 5: CTRL2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OEN	RDNT	INPUT	VAL	0	COFRC	COINIT		SIPS	PIPS	OPS	MSTR	OUTMODE[3:0]			
W					FORCE											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-11. Control Register 2 (CTRL2)

Table 24-11. CTRL2 field descriptions

Field	Description
OEN	Output Enable. This bit determines the direction of the external pin. 0 The external pin is configured as an input. 1 OFLAG output signal is driven on the external pin. Other timer channels using this external pin as their input see the driven value. The polarity of the signal is determined by the OPS bit.
RDNT	Redundant Channel Enable. This bit enables redundant channel checking between adjacent channels (0 and 1, 2 and 3, 4 and 5). When this bit is cleared, the RCF bit in this channel cannot be set. When this bit is set, the RCF bit is set by a miscompare between the OFLAG of this channel and the OFLAG of its redundant adjacent channel, which causes the output of this channel to go inactive (logic 0 prior to consideration of the OPS bit). 0 Disable redundant channel checking. 1 Enable redundant channel checking.
INPUT	External input signal. This read-only bit reflects the current state of the signal selected via SECSRC after application of the SIPS bit and filtering.
VAL	Forced OFLAG Value. This bit determines the value of the OFLAG output signal when a software triggered FORCE command occurs or when the OFLAG is forced from another channel due to the COFRC bit being set.
FORCE	Force the OFLAG output. This write-only bit forces the current value of the VAL bit to be written to the OFLAG output. This bit always reads as a zero. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if OUTMODE is 0000 (software controlled). **Note:** Setting this bit while the OUTMODE is a different value may yield unpredictable results.
COFRC	Co-channel OFLAG Force. This bit enables the compare from another channel within the module to force the state of this counter's OFLAG output signal. 0 Other channels cannot force the OFLAG of this channel. 1 Other channels may force the OFLAG of this channel.
COINIT	Co-channel Initialization. These bits enable another channel within the module to force the re-initialization of this channel when the other channel has an active compare event. 00 Other channels cannot force re-initialization of this channel. 01 Other channels may force a re-initialization of this channel's counter using the LOAD reg. 10 Other channels may force a re-initialization of this channel's counter with the CMPLD2 reg when this channel is counting down or the CMPLD1 reg when this channel is counting up. 11 Reserved.
SIPS	Secondary Source Input Polarity Select. This bit inverts the polarity of the signal selected by the SECSRC bits. 0 True polarity. 1 Inverted polarity.
PIPS	Primary Source Input Polarity Select. This bit inverts the polarity of the signal selected by the PRISRC bits. This only applies if the signal selected by PRISRC is not the prescaled IP Bus clock. 0 True polarity. 1 Inverted polarity.
OPS	Output Polarity Select. This bit inverts the OFLAG output signal polarity. 0 True polarity. 1 Inverted polarity.

Table 24-11. CTRL2 field descriptions (continued)

Field	Description
MSTR	Master Mode. This bit enables the compare function's output to be broadcasted to the other channels in the module. The compare signal then can be used to reinitialize the other counters and/or force their OFLAG signal outputs. 0 Disable broadcast of compare events from this channel. 1 Enable broadcast of compare events from this channel.
OUTMODE	Output Mode. These bits determine the mode of operation for the OFLAG output signal. 0000 Software controlled. 0001 Clear OFLAG output on successful compare (COMP1 or COMP2). 0010 Set OFLAG output on successful compare (COMP1 or COMP2). 0011 Toggle OFLAG output on successful compare (COMP1 or COMP2). 0100 Toggle OFLAG output using alternating compare registers. 0101 Set on compare with COMP1, cleared on secondary source input edge. 0110 Set on compare with COMP2, cleared on secondary source input edge. 0111 Set on compare, cleared on counter rollover. 1000 Set on successful compare on COMP1, cleared on successful compare on COMP2. 1001 Asserted while counter is active, cleared when counter is stopped. 1010 Asserted when counting up, cleared when counting down. 1011 Reserved. 1100 Reserved. 1101 Reserved. 1110 Reserved. 1111 Enable gated clock output while counter is active.

24.3.3.10 Control Register 3 (CTRL3)

Address: Base + 0x0012 (Channel 0: CTRL3) Base + 0x0072 (Channel 3: CTRL3)
Base + 0x0032 (Channel 1: CTRL3) Base + 0x0092 (Channel 4: CTRL3)
Base + 0x0052 (Channel 2: CTRL3) Base + 0x00B2 (Channel 5: CTRL3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STP EN	ROC		0	1	1	1	1	C2FCNT[2:0]			C1FCNT[2:0]			DBGEN [1:0]	
W																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Figure 24-12. Control Register 3 (CTRL3)

Table 24-12. CTRL3 field descriptions

Field	Description
STPEN	Stop Actions Enable. This bit allows the tri-stating of the timer output during stop mode. 0 Output enable is unaffected by stop mode. 1 Output enable is disabled during stop mode.
ROC	Reload on Capture. These bits enable the capture function to cause the counter to be reloaded from the LOAD register. 00 Do not reload the counter on a capture event. 01 Reload the counter on a capture 1 event. 10 Reload the counter on a capture 2 event. 11 Reload the counter on both a capture 1 event and a capture 2 event.
C2FCNT	CAPT2 FIFO Word Count. This field reflects the number of words in the CAPT2 FIFO.

Table 24-12. CTRL3 field descriptions (continued)

Field	Description
C1FCNT	CAPT1 FIFO Word Count. This field reflects the number of words in the CAPT1 FIFO.
DBGEN	Debug Actions Enable. These bits allow the counter channel to perform certain actions in response to the chip entering debug mode. 00 Continue with normal operation during debug mode (default). 01 Halt channel counter during debug mode. 10 Force OFLAG to logic 0 (prior to consideration of the OPS bit) during debug mode. 11 Both halt counter and force OFLAG to 0 during debug mode.

24.3.3.11 Status Register (STS)

Address: Base + 0x0014 (Channel 0: STS) Base + 0x0074 (Channel 3: STS)
Base + 0x0034 (Channel 1: STS) Base + 0x0094 (Channel 4: STS)
Base + 0x0054 (Channel 2: STS) Base + 0x00B4 (Channel 5: STS) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	WDF	RCF	ICF2	ICF1	IEHF	IELF	TOF	TCF2	TCF1	TCF
W							w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-13. Status Register (STS)

Table 24-13. STS field descriptions

Field	Description
WDF[1]	Watchdog Timeout Flag. This bit is set when the watchdog times out by counting down to zero. The watchdog must be enabled for timeout to occur and channel 0 must be in quadrature decode count mode (CNTMODE = 100). This bit is cleared by writing a 1 to this bit position after either writing a non-zero value to WDTOL and/or WDTOH, or else exiting quadrature decode counting mode. This bit is only in channel 0.
RCF	Redundant Channel Flag. This bit is set when a miscompare is detected between this channel's OFLAG value and the OFLAG value of the corresponding redundant channel. Corresponding channels are grouped together in the following pairs: 0 and 1, 2 and 3, or 4 and 5. This bit can only be set if the RDNT bit is set. This bit is only in even channels (0, 2, and 4).
ICF2	Input Capture 2 Flag. This bit is set when an input capture event (as defined by CPT2MODE) occurs and the word count of the CAPT2 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a one to this bit position if ICF2DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF2DE is set (DMA).
ICF1	Input Capture 1 Flag. This bit is set when an input capture event (as defined by CPT1MODE) occurs while the counter is enabled and the word count of the CAPT1 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a one to this bit position if ICF1DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF1DE is set (DMA).
IEHF	Input Edge High Flag. This bit is set when a positive input transition occurs (on an input selected by SECSRC) while the counter is enabled.
IELF	Input Edge Low Flag. This bit is set when a negative input transition occurs (on an input selected by SECSRC) while the counter is enabled.

Table 24-13. STS field descriptions (continued)

Field	Description
TOF	Timer Overflow Flag. This bit is set when the counter rolls over its maximum value 0xFFFF or 0x0000 (depending on count direction).
TCF2	Timer Compare 2 Flag. This bit is set when a successful compare occurs with COMP2.
TCF1	Timer Compare 1 Flag. This bit is set when a successful compare occurs with COMP1.
TCF	Timer Compare Flag. This bit is set when a successful compare occurs.

1 Watchdog timer is implemented on eTimer_0 only. On eTimer_1 and eTimer_2, this bit is reserved.

24.3.3.12 Interrupt and DMA Enable Register (INTDMA)

Address: Base + 0x0016 (Channel 0: INTDMA) Base + 0x0076 (Channel 3: INTDMA)
Base + 0x0036 (Channel 1: INTDMA) Base + 0x0096 (Channel 4: INTDMA)
Base + 0x0056 (Channel 2: INTDMA) Base + 0x00B6 (Channel 5: INTDMA) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ICF2DE	ICF1DE	CMPLD2DE	CMPLD1DE	0	0	WDFIE	RCFIE	ICF2IE	ICF1IE	IEHFIE	IELFIE	TOFIE	TCF2IE	TCF1IE	TCFIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-14. Interrupt and DMA Enable Register (INTDMA)

Table 24-14. INTDMA field descriptions

Field	Description
ICF2DE	Input Capture 2 Flag DMA Enable. Setting this bit enables DMA read requests for CAPT2 when the ICF2 bit is set. Do not set both this bit and the ICF2IE bit.
ICF1DE	Input Capture 1 Flag DMA Enable. Setting this bit enables DMA read requests for CAPT1 when the ICF1 bit is set. Do not set both this bit and the ICF1IE bit.
CMPLD2DE	Comparator Load Register 2 Flag DMA Enable. Setting this bit enables DMA write requests to the CMPLD2 register whenever data is transferred out of the CMPLD2 reg into either the CNTR, COMP1, or COMP2 registers.
CMPLD1DE	Comparator Load Register 1 Flag DMA Enable. Setting this bit enables DMA write requests to the CMPLD1 register whenever data is transferred out of the CMPLD1 reg into the CNTR, COMP1, or COMP2 registers.
WDFIE[1]	Watchdog Flag Interrupt Enable. Setting this bit enables interrupts when the WDF bit is set. This bit is only in channel 0.
RCFIE	Redundant Channel Flag Interrupt Enable. Setting this bit enables interrupts when the RCF bit is set. This bit is only in even channels (0, 2, and 4).
ICF2IE	Input Capture 2 Flag Interrupt Enable. Setting this bit enables interrupts when the ICF2 bit is set. Do not set both this bit and the ICF2DE bit.
ICF1IE	Input Capture 1 Flag Interrupt Enable. Setting this bit enables interrupts when the ICF1 bit is set. Do not set both this bit and the ICF1DE bit.

Table 24-14. INTDMA field descriptions (continued)

Field	Description
IEHFIE	Input Edge High Flag Interrupt Enable. Setting this bit enables interrupts when the IEHF bit is set.
IELFIE	Input Edge Low Flag Interrupt Enable. Setting this bit enables interrupts when the IELF bit is set.
TOFIE	Timer Overflow Flag Interrupt Enable. Setting this bit enables interrupts when the TOF bit is set.
TCF2IE	Timer Compare 2 Flag Interrupt Enable. Setting this bit enables interrupts when the TCF2 bit is set.
TCF1IE	Timer Compare 1 Flag Interrupt Enable. Setting this bit enables interrupts when the TCF1 bit is set.
TCFIE	Timer Compare Flag Interrupt Enable. Setting this bit enables interrupts when the TCF bit is set.

[1] Watchdog timer is implemented on eTimer_0 only. On eTimer_1 and eTimer_2, this bit is reserved.

24.3.3.13 Comparator Load Register 1 (CMPLD1)

The read/write Comparator Load Register 1 (CMPLD1) is the preload value for the COMP1 register. This register can also be used to load into the CNTR register. This register is not byte-accessible. More information on the use of this register can be found in Section 24.4.2.12.2, Usage of Compare Load registers.

Address: Base + 0x0018 (Channel 0: CMPLD1) Base + 0x0078 (Channel 3: CMPLD1)
Base + 0x0038 (Channel 1: CMPLD1) Base + 0x0098 (Channel 4: CMPLD1)
Base + 0x0058 (Channel 2: CMPLD1) Base + 0x00B8 (Channel 5: CMPLD1) Access: User read/write

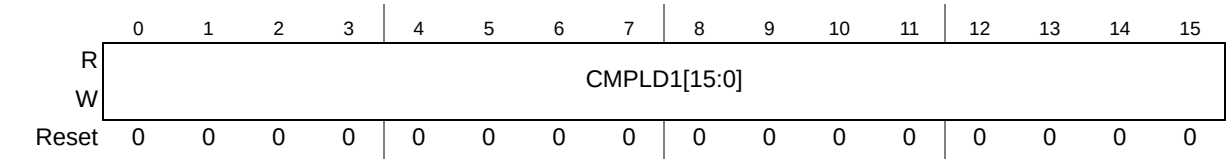


Figure 24-15. Comparator Load 1 (CMPLD1)

Table 24-15. CMPLD1 field descriptions

Field	Description
CMPLD1[15:0]	Preload value for the COMP1 register.

24.3.3.14 Comparator Load Register 2 (CMPLD2)

The read/write Comparator Load Register 2 (CMPLD2) is the preload value for the COMP2 register. This register can also be used to load into the CNTR register. This register is not byte-accessible. More information on the use of this register can be found in Section 24.4.2.12.2, Usage of Compare Load registers.

Address: Base + 0x001A (Channel 0: CMPLD2) Base + 0x007A (Channel 3: CMPLD2)
Base + 0x003A (Channel 1: CMPLD2) Base + 0x009A (Channel 4: CMPLD2)
Base + 0x005A (Channel 2: CMPLD2) Base + 0x00BA (Channel 5: CMPLD2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-16. Comparator Load 2 (CMPLD2)

Table 24-16. CMPLD2 field descriptions

Field	Description
CMPLD2[15:0]	Preload value for the COMP2 register.

24.3.3.15 Compare and Capture Control Register (CCCTRL)

Address: Base + 0x001C (Channel 0: CCCTRL) Base + 0x007C (Channel 3: CCCTRL)
Base + 0x003C (Channel 1: CCCTRL) Base + 0x009C (Channel 4: CCCTRL)
Base + 0x005C (Channel 2: CCCTRL) Base + 0x00BC (Channel 5: CCCTRL) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CLC2[1:0]				CLC1[1:0]		CMP MODE[1:0]		CPT2 MODE[1:0]		CPT1 MODE[1:0]		CFWM [1:0]		ONE SHOT	ARM
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-17. Compare and Capture Control Register (CCCTRL)

Table 24-17. CCCTRL field descriptions

Field	Description
CLC2	Compare Load Control 2. These bits control when COMP2 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved. 010 Load COMP2 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP2 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP2 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP2 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD2 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD2 upon successful compare with the value in COMP2.
CLC1	Compare Load Control 1. These bits control when COMP1 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved. 010 Load COMP1 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP1 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP1 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP1 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD1 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD1 upon successful compare with the value in COMP2.

Table 24-17. CCCTRL field descriptions (continued)

Field	Description
CMPMODE	Compare Mode. These bits control when the COMP1 and COMP2 registers are used in regards to the counting direction. 00 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting up. 01 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting up. 10 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting down. 11 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting down.
CPT2MODE	Capture 2 Mode Control. These bits control the operation of the CAPT2 register as well as the operation of the ICF2 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CPT1MODE	Capture 1 Mode Control. These bits control the operation of the CAPT1 register as well as the operation of the ICF1 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CFWM	Capture FIFO Water Mark. This field represents the water mark level for the CAPT1 and CAPT2 FIFOs. The capture flags, ICF1 and ICF2, are not set until the word count of the corresponding FIFO is greater than this water mark level.
ONESHOT	One-Shot Capture Mode. This bit selects between free-running and one-shot mode for the input capture circuitry. 0 Free-running mode is selected. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and capture circuit 1 is re-armed. The process continues indefinitely. If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit. 1 One-shot mode is selected. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and the ARM bit is cleared. No further captures are performed until the ARM bit is set again. If only one of the capture circuits is enabled, then a single capture occurs on the enabled capture circuit and the ARM bit is then cleared.
ARM	Arm Capture. Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self-cleared when in one-shot mode and the enabled capture circuit(s) has had a capture event(s). 0 Input capture operation is disabled. 1 Input capture operation as specified by the CPT1MODE and CPT2MODE bits is enabled.

24.3.3.16 Input Filter Register (FILT)

Address: Base + 0x001E (Channel 0: FILT)
Base + 0x003E (Channel 1: FILT)
Base + 0x005E (Channel 2: FILT)
Base + 0x007E (Channel 3: FILT)
Base + 0x009E (Channel 4: FILT)
Base + 0x00BE (Channel 5: FILT)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FILT_CNT[2:0]			FILT_PER[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-18. Input Filter Register (FILT)

Table 24-18. FILT field descriptions

Field	Description
FILT_CNT	Input Filter Sample Count. These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in Section 24.3.3.16.1, Input filter considerations.
FILT_PER	Input Filter Sample Period. These bits represent the sampling period (in IPBus clock cycles) of the eTimer input signal. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in Section 24.3.3.16.1, Input filter considerations.

24.3.3.16.1 Input filter considerations

The FILT_PER value should be set such that the sampling period is larger than the period of the expected noise. This way a noise spike only corrupts one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT + 3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of: (((FILT_CNT + 3) × FILT_PER) + 2) IPBus clock periods.

24.3.4 Watchdog timer registers

The watchdog registers are implemented on eTimer_0 only.

24.3.4.1 Watchdog Timeout Registers (WDTOL and WDTOH)

Address: Base + 0x0100
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDTOL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-19. Watchdog Timeout Low Word Register (WDTOL)

Address: Base + 0x0102 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WDTOH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-20. Watchdog Timeout High Word Register (WDTOH)

Table 24-19. WDTOL, WDTOH field descriptions

Field	Description
WDTOL, WDTOH	Watchdog Timeout. These registers are combined to form the 32-bit timeout count for the Timer watchdog function. This timeout count monitors for inactivity on the inputs when channel 0 is in the quadrature decode count mode. The watchdog function is enabled whenever WDTO contains a non-zero value (although actual counting only occurs if channel 0 is in quadrature decode counting mode). The watchdog timeout down counter is loaded whenever WDTOH is written. These registers are not byte-accessible. See Section 24.4.3.5, Watchdog timer, for more information on the use of the watchdog timer.

24.3.5 Configuration registers

24.3.5.1 Channel Enable Register (ENBL)

Address: Base + 0x010C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ENBL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 24-21. Channel Enable Register (ENBL)

Table 24-20. ENBL field descriptions

Field	Description
ENBL	Timer Channel Enable. These bits enable the prescaler (if it is being used) and counter in each channel. Multiple ENBL bits can be set at the same time to synchronize the start of separate channels. If an ENBL bit is set, then the corresponding channel starts counting as soon as the CNTMODE field has a value other than 000. When an ENBL bit is clear, the corresponding channel maintains its current value. 0 Timer channel is disabled. 1 Timer channel is enabled (default).

24.3.5.2 DMA Request Select Registers (DREQ0, DREQ1)

Address: Base + 0x0110 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	DREQ0				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-22. DMA Request 0 Select Register (DREQ0)

Table 24-21. DREQ0 field descriptions

Field	Description
DREQ0	DMA Request Select. These fields select which DMA request source is muxed onto one of the four module-level DMA request outputs. Each of the DREQ registers must be programmed with a different value. Otherwise, a single DMA source can cause multiple DMA requests. Enable a DMA request in the channel-specific INTDMA register after the DREQ*n* registers are programmed.

Address: Base + 0x0112 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	DREQ1				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-23. DMA Request 1 Select Register (DREQ1)

Table 24-22. DREQ1 field descriptions

Field	Description
DREQ1	DMA Request Select. These fields select which DMA request source is muxed onto one of the four module-level DMA request outputs. Each of the DREQ registers must be programmed with a different value. Otherwise, a single DMA source can cause multiple DMA requests. Enable a DMA request in the channel-specific INTDMA register after the DREQ*n* registers are programmed.

Table 24-23. Values for DREQ*n*

Value	Selected DMA Request
00000	Channel 0 CAPT1 DMA read request
00001	Channel 0 CAPT2 DMA read request
00010	Channel 0 CMPLD1 DMA write request
00011	Channel 0 CMPLD2 DMA write request
00100	Channel 1 CAPT1 DMA read request
00101	Channel 1 CAPT2 DMA read request
00110	Channel 1 CMPLD1 DMA write request
00111	Channel 1 CMPLD2 DMA write request

Table 24-23. Values for DREQ*n* (continued)

Value	Selected DMA Request
01000	Channel 2 CAPT1 DMA read request
01001	Channel 2 CAPT2 DMA read request
01010	Channel 2 CMPLD1 DMA write request
01011	Channel 2 CMPLD2 DMA write request
01100	Channel 3 CAPT1 DMA read request
01101	Channel 3 CAPT2 DMA read request
01110	Channel 3 CMPLD1 DMA write request
01111	Channel 3 CMPLD2 DMA write request
10000	Channel 4 CAPT1 DMA read request
10001	Channel 4 CAPT2 DMA read request
10010	Channel 4 CMPLD1 DMA write request
10011	Channel 4 CMPLD2 DMA write request
10100	Channel 5 CAPT1 DMA read request
10101	Channel 5 CAPT2 DMA read request
10110	Channel 5 CMPLD1 DMA write request
10111	Channel 5 CMPLD2 DMA write request
11000–11111	Reserved

NOTE

If the DREQ*n* registers contain the same value, then multiple DMA requests can be generated in response to a single flag/condition. To avoid this issue, ensure each of the DREQ*n* registers has a unique value prior to enabling DMA requests.

24.4 Functional description

24.4.1 General

Each channel has two basic modes of operation: it can time internal or external events, or it can count an internal clock source while an external input signal is asserted, thus timing the width of the external input signal.

- The counter/timer can count the rising, falling, or both edges of the selected input pin.
- The counter/timer can decode and count quadrature encoded input signals.
- The counter/timer can count up and down using dual inputs in a “count with direction” format.
- The counter/timer’s terminal count value (modulo) is programmable.

  - The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter/timer can count repeatedly, or it can stop after completing one count cycle.
- The counter/timer can be programmed to count to a programmed value and then immediately reinitialize, or it can count through the compare value until the count rolls over to zero.

The external inputs to each counter/timer are shareable among each of the six channels within the module. The external inputs can:

- Be used as count commands
- Be used as timer commands
- Trigger the current counter value to be “captured”
- Be used to generate interrupt requests

The polarity of the external inputs is selectable.

The primary output of each channel is the output signal OFLAG. The OFLAG output signal can be:

- Set, cleared, or toggled when the counter reaches the programmed value.
- The OFLAG output signal may be output to an external pin instead of having that pin serve as a timer input.
- The OFLAG output signal enables each counter to generate square waves, PWM, or pulse stream outputs.
- The polarity of the OFLAG output signal is programmable.

Any channel can be assigned as a master. A master’s compare signal can be broadcasted to the other channels within the module. The other channels can be configured to reinitialize their counters and/or force their OFLAG output signals to predetermined values when a master channel’s compare event occurs.

24.4.2 Counting modes

The selected external signals are sampled at the eTimer’s base clock rate and then run through a transition detector. The maximum count rate is one-half of the eTimer’s base clock rate when using an external signal. Internal clock sources can be used to clock the counters at the eTimer’s base clock rate.

If a counter is programmed to count to a specific value and then stop, the CNTMODE field in the CTRL1 register is cleared when the count terminates.

24.4.2.1 STOP mode

If the CNTMODE field is set to ‘000’, the timer is inert. No counting occurs. Stop mode also disables the interrupts caused by input transitions on a selected input pin.

24.4.2.2 COUNT mode

If the CNTMODE field is set to ‘001’, the timer counts the rising edges of the selected clock source. This mode is useful for generating periodic interrupts for timing purposes, or counting external events such as “widgets” on a conveyor belt passing a sensor. If the selected input is inverted by setting the PIPS bit, then the negative edge of the selected external input signal is counted.

See Section 24.4.2.9, CASCADE-COUNT mode, through Section 24.4.2.12, VARIABLE-FREQUENCY PWM mode, for additional capabilities of this operating mode.

24.4.2.3 EDGE-COUNT mode

If the CNTMODE field is set to ‘010’, the timer counts both edges of the selected external clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel.

24.4.2.4 GATED-COUNT mode

If the CNTMODE field is set to ‘011’, the timer counts while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted by setting the PIPS bit, then the timer counts while the selected secondary input is low.

NOTE

Delays in the edge detection circuitry can cause the rising edge on the primary source to be compared to the secondary source value one clock later in time. To prevent this problem, do the following:

On the primary source, set FILT[FILT_PER = 1] and FILT[FILT_CNT = 0]. On the secondary source, set FILT[FILT_PER = 1] and FILT[FILT_CNT = 1]. This introduces a 5 clock cycle latency on the primary source and a 6 cycle latency on the secondary source, which properly aligns the two signals.

To avoid a false count, ensure that the primary source is low before the secondary source goes high.

24.4.2.5 QUADRATURE-COUNT Mode

If the CNTMODE field is set to ‘100’, the timer decodes the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 24-24 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

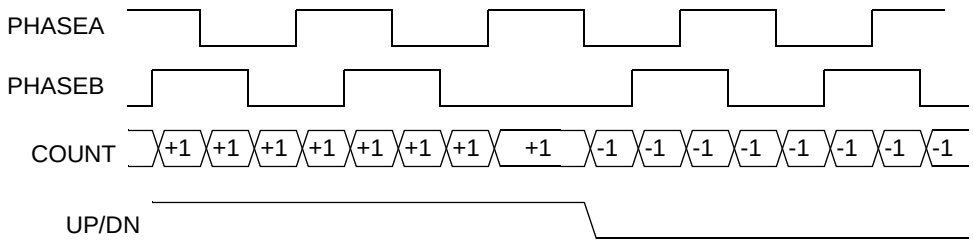


Figure 24-24. Quadrature incremental position encoder

24.4.2.6 SIGNED-COUNT mode

If the CNTMODE field is set to ‘101’, the timer counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

NOTE

Delays in the edge detection circuitry can cause the rising edge on the primary source to be compared to the secondary source value one clock later in time. To prevent this problem, do the following:

On the primary source, set FILT[FILT_PER = 1] and FILT[FILT_CNT = 0]. On the secondary source, set FILT[FILT_PER = 1] and FILT[FILT_CNT = 1]. This introduces a 5 clock cycle latency on the primary source and a 6 cycle latency on the secondary source, which properly aligns the two signals.

To avoid a false count, ensure that the primary source is low before the secondary source goes high.

24.4.2.7 TRIGGERED-COUNT mode

If the CNTMODE field is set to ‘110’, the timer begins counting the primary clock source after a positive transition (negative if SIPS = 1) of the secondary input occurs. The counting continues until a compare event occurs or another positive input transition is detected. Subsequent secondary positive input transitions continue to restart and stop the counting until a compare event occurs.

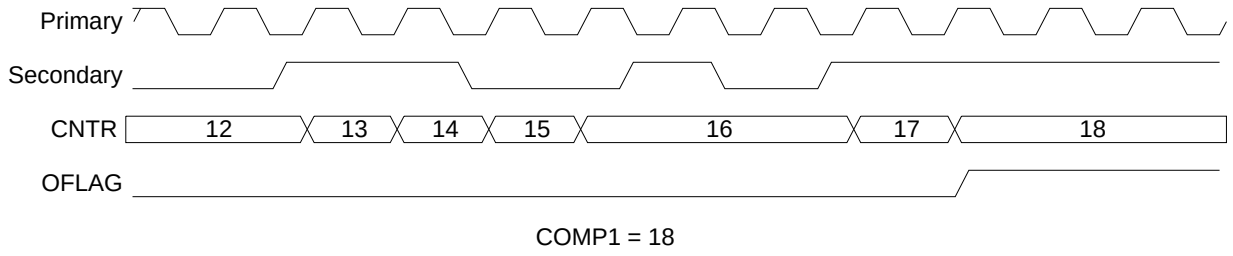


Figure 24-25. TRIGGERED-COUNT mode (Length=1)

24.4.2.8 ONE-SHOT mode

If the CNTMODE field is set to ‘110’, and the timer is set to reinitialize at a compare event (LENGTH = 1), and the OFLAG OUTMODE is set to ‘0101’ (cleared on init, set on compare), the timer works in a “One-Shot Mode”. An external event causes the timer to count. When the terminal count is reached, the output is asserted. This “delayed” output can be used to provide timing delays.

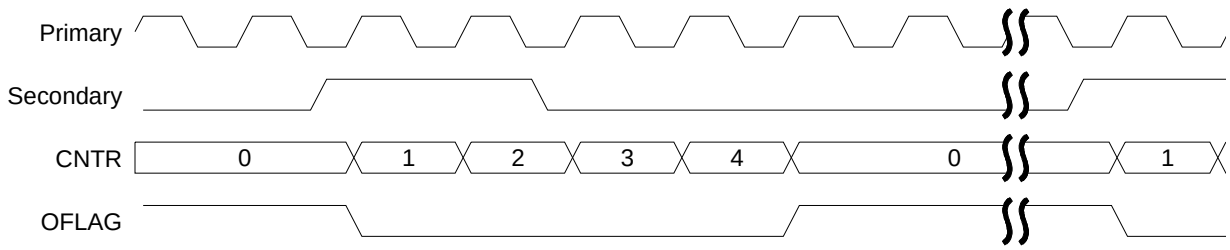


Figure 24-26. ONE-SHOT mode (Length = 1)

24.4.2.9 CASCADE-COUNT mode

If the CNTMODE field is set to ‘111’, the timer’s input is connected to the output of another selected timer. The timer counts up and down as compare events occur in the selected source timer. This “Cascade” or “Daisy-Chained” mode enables multiple timers to be cascaded to yield longer counter lengths. When operating in cascade mode, a special high speed signal path is used between modules rather than the OFLAG output signal. If the selected source timer is counting up and it experiences a compare event, the timer is incremented. If the selected source timer is counting down and it experiences a compare event, the timer is decremented.

Up to two counters may be cascaded to create a 32-bit wide synchronous counter.

Whenever any counter is read within a counter module, all of the counters’ values within the module are captured in their respective HOLD registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the HOLD registers of the other counters in the chain. The cascaded counter mode is synchronous.

NOTE

It is possible to connect counters together by using the other (non-cascade) counter modes and selecting the outputs of other counters as a clock source. In this case, the counters are operating in a “ripple” mode, where higher order counters transition a clock later than a purely synchronous design.

NOTE

A channel can be cascaded with any other channel, but more than 2 channels cannot be cascaded together. However, separate cascades of pairs of channels can be created. For example, you can cascade channels 0 and 1 and separately cascade channels 5 and 4. You cannot cascade channels 0, 1, and 5.

24.4.2.10 PULSE-OUTPUT mode

If the counter is set up for CNTMODE = 001, and the OFLAG OUTMODE is set to ‘1111’ (gated clock output), and the ONCE bit is set, then the counter outputs a stream of pulses that has the same frequency of the selected clock source, and the number of output pulses is equal to the compare value minus the init value. This mode is useful for driving step motor systems.

NOTE

This does not work if the PRISRC is set to 11000 (IP_bus/1).

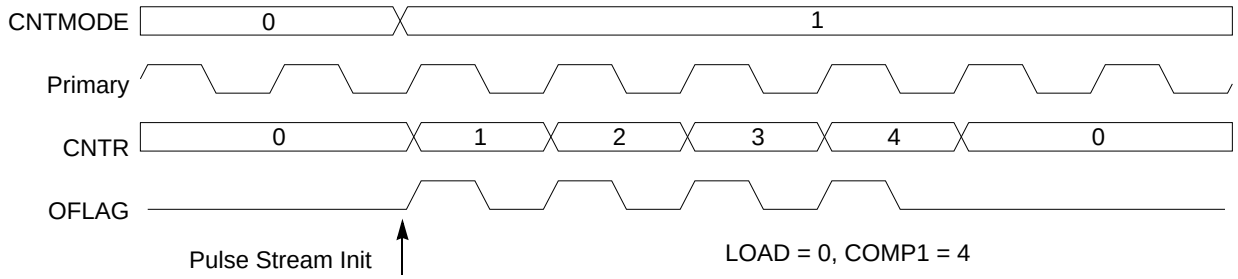


Figure 24-27. PULSE-OUTPUT mode

24.4.2.11 FIXED-FREQUENCY PWM mode

If the counter is set up for CNTMODE = 001, count through rollover (LENGTH = 0), continuous count (ONCE = 0), and the OFLAG OUTMODE is ‘0111’ (set on compare, cleared on counter rollover), then the counter output yields a Pulse Width Modulated (PWM) signal with a frequency equal to the count clock frequency divided by 65,536 and a pulse width duty cycle equal to the compare value divided by 65,536. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters.

24.4.2.12 VARIABLE-FREQUENCY PWM mode

If the counter is set up for CNTMODE = 001, count until compare (LENGTH = 1), continuous count (ONCE = 0), and the OFLAG OUTMMODE is ‘0100’ (toggle OFLAG and alternate compare registers), then the counter output yields a Pulse Width Modulated (PWM) signal whose frequency and pulse width is determined by the values programmed into the COMP1 and COMP2 registers, and the input clock frequency. This method of PWM generation has the advantage of allowing almost any desired PWM frequency and/or constant on or off periods. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters. The CMPLD1 and CMPLD2 registers are especially useful for this mode, as they allow the programmer time to calculate values for the next PWM cycle while the current PWM cycle is underway.

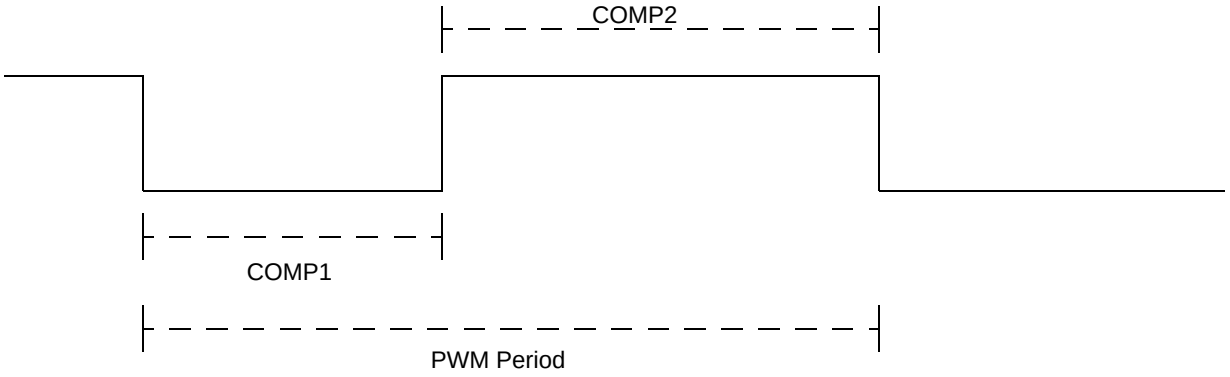


Figure 24-28. Variable PWM waveform

24.4.2.12.1 Usage of Compare registers

The dual compare registers (COMP1 and COMP2) provide a bidirectional modulo count capability.

The COMP1 register should be set to the desired maximum count value or 0xFFFF to indicate the maximum unsigned value prior to rollover, and the COMP2 register should be set to the minimum count value or 0x0000 to indicate the minimum unsigned value prior to rollunder.

If the output mode is set to 0100, the OFLAG toggles while using alternating compare registers. In this variable frequency PWM mode, the COMP2 value defines the desired pulse width of the on time, and the COMP1 register defines the off time. COMP1 is used when OFLAG = 0 and COMP2 is used when OFLAG = 1. OFLAG can be forced to a value using CTRL2[FORCE] and CTRL2[VAL].

Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it counts to 0xFFFF or 0x0000, rolls over, then begins counting toward the new value. The check is: CNTR = COMPx, *not* CNTR > COMP1 or CNTR < COMP2.

The use of the CMPLD1 and CMPLD2 registers to preload compare values help to minimize this problem.

24.4.2.12.2 Usage of Compare Load registers

The CMPLD1, CMPLD2 and CCCTRL registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality while using these registers we strongly suggest using the method described in this section.

The purpose of the compare load feature is to allow quicker updating of the compare registers. A compare register can be updated using interrupts. However, because of the latency between an interrupt event occurring and the service of that interrupt, the possibility exists that the counter may have already counted past the new compare value by the time the compare register is updated by the interrupt service routine. The counter would then continue counting until it rolled over and reached the new compare value.

To address this, the compare registers are updated in hardware in the same way the counter register is re-initialized to the value stored in the LOAD register. The compare load feature allows the user to calculate new compare values and store them in the comparator load registers. When a compare event occurs, the new compare values in the comparator load registers are written to the compare registers, eliminating the use of software to do this.

The compare load feature is intended to be used in variable frequency PWM mode. The COMP1 register determines the pulse width for the logic low part of OFLAG, and COMP2 determines the pulse width for the logic high part of OFLAG. The period of the waveform is determined by the COMP1 and COMP2 values and the frequency of the primary clock source. See Figure 24-28.

24.4.2.13 MODULO COUNTING mode

To create a modulo counter using COMP1 and COMP2 as the counter boundaries (instead of 0x0000 and 0xFFFF), set the registers in the following manner. Set CNTMODE to either 100 (quadrature count mode) or 101 (count with direction mode). Use count through rollover (LENGTH = 0) and continuous count (ONCE = 0). Set COMP1 and CMPLD1 to the upper boundary value. Set COMP2 and CMPLD2 to the lower boundary value. Set CMPMODE = 10 (COMP1 is used when counting up and COMP2 is used when

counting down). Set CLC2 = 110 (load CNTR with value of CMPLD2 on COMP1 compare) and CLC1 = 111 (load CNTR with value of CMPLD1 on COMP2 compare).

24.4.3 Other features

24.4.3.1 Redundant OFLAG checking

This mode allows the user to bundle two timer functions generating any pattern to compare their resulting OFLAG behaviors (output signal).

The redundant mode is used to support online checks for functional safety reasons. Whenever a mismatch between the two adjacent channels occurs, it is reported via an interrupt to the core and the two outputs are put into their inactive states. An error is flagged via the RCF flag.

This feature can be tested by forcing a transition on one of the OFLAGs using the VAL and FORCE bits of the channel.

24.4.3.2 Loopback checking

This mode is always available in that one channel can generate an OFLAG while another channel uses the first channels OFLAG as its input to be measured and verified to be as expected.

24.4.3.3 Input capture mode

Input capture is used to measure pulse width (by capturing the counter value on two successive input edges) or waveform period (by capturing the counter value on two consecutive rising edges or two consecutive falling edges). The Capture Registers store a copy of the counter's value when an input edge (positive, negative, or both) is detected. The type of edge to be captured by each circuit is determined by the CPT1MODE and CPT2MODE bits, whose functionality is listed in Table 24-17. Also, controlling the operation of the capture circuits is the arming logic, which allows captures to be performed in a free-running (continuous) or one-shot fashion. In free-running mode, the capture sequences are performed indefinitely. If both capture circuits are enabled, they work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one-shot mode, only one capture sequence is performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

24.4.3.4 Master/Slave mode

Any timer channel can be assigned as a master (MSTR = 1). A master's compare signal can be broadcasted to the other channels within the module. The other counters can be configured to reinitialize their counters (COINIT = 1) and/or force their OFLAG output signals (COFRC = 1) to predetermined values when a master counter compare event occurs.

24.4.3.5 Watchdog timer

The watchdog timer monitors for a stalled count when channel 0 is in quadrature count mode. When the watchdog is enabled, it loads the timeout value into a down counter. The down counter counts as long as channel 0 is in quadrature decode count mode. If this down counter reaches zero, an interrupt is asserted. The down counter is reloaded to the timeout value each time the counter value from channel 0 changes. If the channel 0 count value is toggling between two values (indicating a possibly stalled encoder), then the down counter is not reloaded.

NOTE

The watchdog timer is implemented only on eTimer_0.

24.5 Interrupts

Each of the channels within the eTimer can generate an interrupt from several sources. The watchdog logic also generates interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 24-24. Interrupt summary

Core interrupt	Interrupt flag	Interrupt enable	Name	Description
Channels 0–5	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter rollover or rollunder
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2
Watchdog[1]	WDF	WDFIE	Watchdog timeout interrupt	Watchdog has timed out
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel

1 Applicable only for eTimer_0.

24.6 DMA

Each channel can request a DMA read access for each of the capture registers and a DMA write request for each of the compare preload registers. The DREQ registers select between these 24 DMA request sources to generate the two top-level DMA request outputs.

When DMA is used to read the eTimer_CAPTn registers and the DMA has completed its programmed number of transfers, then the extra input capture events will set the eTimer_STS[ICFn] bits and also set

the internal DMA request signal. While the ICFn bits can be cleared by writing a 1 to their bit positions, the DMA request can only be cleared by the internal DMA done signal. This means that when a new DMA transfer is programmed, the eTimer will request a DMA read with possibly unwanted data. In cases where extra eTimer input capture events might occur, the following procedure can be used to prevent unwanted DMA read requests: Upon completion of the DMA transfer, clear the eTimer_INTDMA[ICFnDE] bits.

Table 24-25. DMA summary

DMA request	DMA enable	Name	Description
Channels 0–5	ICF1DE	CAPT1 read request	CAPT1 contains a value
	ICF2DE	CAPT2 read request	CAPT2 contains a value
	CMPLD1DE	CMPLD1 write request	CMPLD1 needs an update
	CMPLD2DE	CMPLD2 write request	CMPLD2 needs an update

24.7 eTimer initialization

Use this sequence to initialize the eTimer module.

1. If you are using multiple channels and want them to be synchronized, begin by setting the ENBL register = 0x0.
2. Configure the COMP1, COMP2, LOAD, CMPLD1, CMPLD2, CCCTRL, FILT, WDTOL, and WDTOH registers.
3. Configure the DREQ0 and DREQ1 registers.
4. Configure the INTDMA registers.
5. Configure the CTRL2 and CTRL3 registers.
6. Configure the CTRL1 registers. Specifically, the CNTMODE field should be programmed last.
7. Finally, set ENBL = 0x3F.

Chapter 25
Fault Collection and Control Unit (FCCU)

25.1 Introduction

The Fault Collection and Control Unit (FCCU) offers a programmable redundant hardware channel to collect errors and to lead the device in a controlled way to a safe state when a failure is present in the device. No CPU intervention is requested for collection and control operation.

The FCCU offers a systematic approach to manage fault detection and control.

The main functions supported by the module are:

- Redundant collection of hardware checker (for example, RCCU) results
- Redundant collection of error information from critical modules on the chip (such as the flash memory or the STCU)
- Collection of test results
- Configurable and graded fault control via user software control
- Internal responses:
  - No reset response
  - IRQ
  - Short “functional” reset
  - Long “functional” reset
  - Safe mode request
- External response (failure is reported to the outside world via output pins)
- Watchdog timer for the re-configuration phase
- Configuration lock
- NVM configuration loading
- Self-checking capabilities:
  - FCCU internal control logic redundancy
  - Additional parity check for the associated configuration registers

The FCCU circuitry is checked at startup by the self-checking procedure. The FCCU is operative with a default configuration (without CPU intervention) immediately after the completion of the self-checking procedure.

Two classes of faults — critical and non-critical — are identified based on the criticality and the related responses. Internal (short or long ‘functional’ reset, interrupt request) and external (FCCU_F signaling) responses are statically defined or programmable based on the fault criticality. The default configuration can be modified only in a specific FCCU state for application/test/debugging purposes.

25.1.1 Glossary and abbreviations

Table 25-1. Abbreviations

Abbreviation	Description
RCC	Redundancy control checkers
RCCU	Redundancy control checker unit
IPS	Internal peripheral system
NMI	Non-maskable interrupts
IRQ	Interrupt request
STCU	Self-testing control unit
FSM	Finite state machine
CF	Critical fault
NCF	Non-critical fault
WS	Wait state
SW	Software
NVM	Nonvolatile memory
CPU	Central processing unit
NCT	NCF timer
HNSHK	Handshake

25.2 Main features

- Management of:
  - 37 critical faults
  - 18 non-critical faults
- Hardware or software fault recovery management
- Fault detection collection
- Fault injection (fake faults)
- External response via fault-state dedicated system signals (FCCU_F pins)
- Internal (chip) responses (alarm state): interrupt request
- Internal (chip) responses (fault state):
  - Long ‘functional’ reset
  - Short ‘functional’ reset
  - Non-maskable interrupt (NMI)
  - Safe mode request
- Bi-Stable, Dual-Rail and Time Switching output protocols using FCCU_F pins
- Internal (to the FCCU) watchdog timer for the re-configuration phase
- Configuration lock

- NVM configuration loading
- Self-checking capabilities:
  - FSM redundancy
  - Parity check for the configuration registers

25.3 Block diagram

The top-level diagram of the FCCU is shown in Figure 25-1.

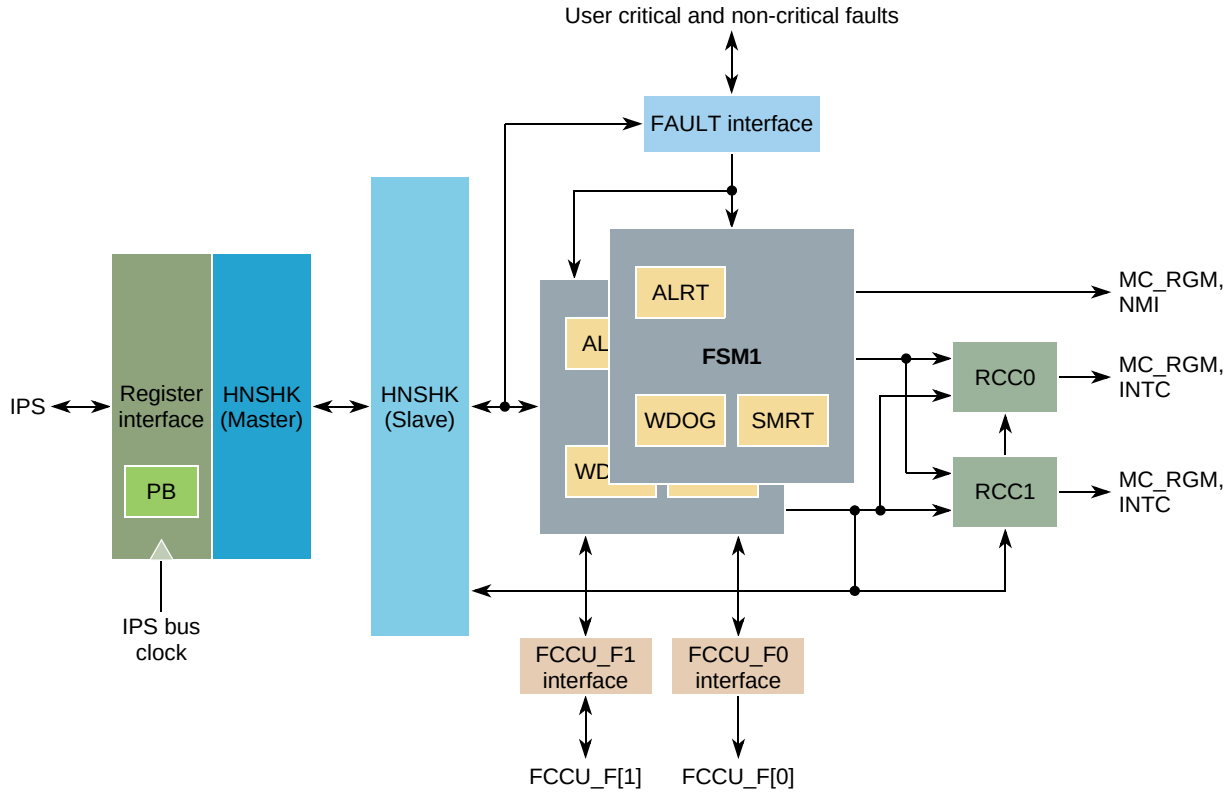


Figure 25-1. FCCU top-level diagram

As shown in Figure 25-1, the FCCU includes these sub-modules:

- Register Interface and HNSHK blocks include the register file, the IPS bus interface, the IRQ interface and the parity block (PB) for the configuration registers. The synchronization block synchronizes the IP Bus clock with the clock used by the internal FCCU logic circuitry (the internal RC oscillator clock).
- Finite State Machine (FSM) units implement the main functions of the FCCU. These units also include the WDOG, the Safe mode request timer (SMRT), and the alarm timer (ALRT).
- FAULT interface implements the interface for the fault conditioning and its management.
- FCCU_Fx units implement the output stage to manage the FCCU_F pins.
- RCCx units implement the redundancy control checker to monitor the FSM unit state and its configuration.

NOTE

The FCCU Redundancy Control Checkers (RCC) should not be confused with the Redundancy Control Checker Units (RCCU) instantiated on the MPC5675K to detect critical lockstep failure.

25.4 Signal description

The FCCU generates two external signals, FCCU_F[0] and FCCU_F[1]. These are described in Section 25.8.13, FCCU_F interface.

25.5 Debug mode

When the MCU is in debug mode, the FCCU behavior is unaffected and remains identical to its operation in normal mode.

25.6 Register interface

The register interface is a slave bus used for configuration purposes via the CPU.

These bus operations are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[0:15] or data[16:31]) data write/read operations to any registers
- Byte (8-bit, data[0:7] or data[8:15] or data[16:23] or data[24:31]) data write/read operations to any registers

Any other operation (free byte enables, misaligned word or half-word access or other operations) are not supported.

The FCCU generates a transfer error in these cases:

- Any write/read access executed outside the address space of the peripheral
- Any write/read operation different from byte/halfword/word (free byte enables, misaligned access, or other operations) on each register
- Any write access to the configuration registers not executed while the FCCU is not in CONFIG state

The registers of the FCCU are accessible (read/write) in each access mode: user, supervisor, or test.

25.7 Memory map and register description

The FCCU registers are listed in Table 25-2. The contents of the configuration registers (labeled as “W in CONFIG state only” in the Access column) can be locked by the OP16 operation as defined in the FCCU_CTRL register.

Table 25-2. FCCU memory map

Offset from FCCU_BASE (0xFFE6_C000)	Register	Access[1]	Reset value[2, 3]	Location
0x00	FCCU Control Register (FCCU_CTRL)	R/W	0x0000_0UU0	on page 736
0x04	FCCU CTRL Key Register (FCCU_CTRLK)	W	0x0000_0000	on page 739
0x08	FCCU Configuration Register (FCCU_CFG)	R always; W in CONFIG state only	0x0000_0008	on page 739
0x0C	FCCU CF Configuration Register 0 (FCCU_CF_CFG0)	R always; W in CONFIG state only	0xFFFF_FFFF	on page 741
0x10	FCCU CF Configuration Register 1 (FCCU_CF_CFG1)		0x0000_00FF	
0x14–0x1B	Reserved			
0x1C	FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0)	R always; W in CONFIG state only	0xFFFF_FFFF	on page 744
0x20–0x2B	Reserved			
0x2C	FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)	R always; W in CONFIG state only	0xAAAA_AAAA	on page 746
0x30	FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)		0xAAAA_800A	
0x34	FCCU CFS Configuration Register 2 (FCCU_CFS_CFG2)		0x0000_082A	
0x38–0x4B	Reserved			
0x4C	FCCU NCFS Configuration Register 0 (FCCU_NCFS_CFG0)	R always; W in CONFIG state only	0x0000_AAAA	on page 747
0x50	FCCU NCFS Configuration Register 1 (FCCU_NCFS_CFG1)		0x0000_0000	
0x54–0x6B	Reserved			
0x6C	FCCU CF Status Register 0 (FCCU_CFS0)	R/W	0x0000_0000	on page 748
0x70	FCCU CF Status Register 1 (FCCU_CFS1)		0x0000_0000	
0x74–0x7B	Reserved			
0x7C	FCCU CF Key Register (FCCU_CFK)	W	0x0000_0000	on page 750
0x80	FCCU NCF Status Register 0 (FCCU_NCFS0)	R/W	0x0000_0000	on page 751
0x84–0x8F	Reserved			
0x90	FCCU NCF Key Register (FCCU_NCFK)	W	0x0000_0000	on page 753
0x94	FCCU NCF Enable Register 0 (FCCU_NCFE0)	R always; W in CONFIG state only	0x06D0_1CFF	on page 753

Table 25-2. FCCU memory map (continued)

Offset from FCCU_BASE (0xFFE6_C000)	Register	Access[1]	Reset value[2, 3]	Location
0x98–0xA3	Reserved			
0xA4	FCCU NCF Timeout Enable Register 0 (FCCU_NCF_TOE0)	R always; W in CONFIG state only	0x0638_1FFF	on page 754
0xA8–0xB3	Reserved			
0xB4	FCCU NCF Timeout Register (FCCU_NCF_TO)	R always; W in CONFIG state only	0x0000_FFFF	on page 755
0xB8	FCCU CFG Timeout Register (FCCU_CFG_TO)	R always; W in CONFIG state only	0x0000_0006	on page 756
0xBC	FCCU I/O Control Register (FCCU_EINOUT)	R/W	0x0000_0000	on page 757
0xC0	FCCU Status Register (FCCU_STAT)	R	0x0000_0000	on page 758
0xC4–0xD0	Reserved			
0x00D4	FCCU SC Freeze Status Register (FCCU_SCFS)	R/W	0x0000_0000	on page 759
0xD8	FCCU CF Fake Register (FCCU_CFF)	W	0x0000_0000	on page 760
0xDC	FCCU NCF Fake Register (FCCU_NCFF)	W	0x0000_0000	on page 761
0xE0	FCCU IRQ Status Register (FCCU_IRQ_STAT)	R/W	0x0000_0000	on page 762
0xE4	FCCU IRQ Enable Register (FCCU_IRQ_EN)	R/W	0x0000_0000	on page 763
0xE8	FCCU XTMR Register (FCCU_XTMR)	R	0x0000_0000	on page 764
0xEC	FCCU MCS Register (FCCU_MCS)	R	0x0000_0000	on page 764

1 In this column, R/W = read/write, R = read-only, and W = write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

25.7.1 FCCU Control Register (FCCU_CTRL)

The FCCU Control Register (FCCU_CTRL) allows execution of these operations:

- Move the FCCU state from the NORMAL state into the CONFIG state
- Move the FCCU state from the CONFIG state into the NORMAL state
- Read or clear the CF status register
- Read or clear the NCF status register
- Read the FCCU FSM status register
- Read or clear the FCCU freeze registers
- Lock the FCCU configuration registers

- Read the ALARM timer
- Read the SMRT timer
- Read the Watchdog timer
- Load the NVM configuration (only for test purposes)

Some critical operations (OP1, OP2, OP16, and OP31) require a key as defined in the FCCU_CTRLK register.

Address: Base + 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	NVML	OPS		0	OPR				
W																
Reset	0	0	0	0	0	0	0	0/1[1]	0/1[1]	0/1[1]	0	0	0	0	0	0

Figure 25-2. FCCU Control Register (FCCU_CTRL)

[1] 1 when NVM interface is used, 0 otherwise.

Table 25-3. FCCU_CTRL field descriptions

Field	Description
NVML	NVM configuration loaded 0 No NVM configuration loaded. 1 NVM configuration loaded. At the end of the reset PHASE3, NVML = 1 and OPS = 11 if the NVM interface of the FCCU is correctly driven by the SSCM interface. This bit can be read and cleared (via OP15 operation) by the software.
OPS	Operation status 00 Idle. 01 In progress. 10 Aborted. 11 Successful. This bit can be read and cleared (via OP15 operation) by the software.

Table 25-3. FCCU_CTRL field descriptions (continued)

Field	Description
OPR	Operation run 00000 No operation [OP0]. 00001 Set the FCCU into the CONFIG state [OP1]. 00010 Set the FCCU into the NORMAL state [OP2]. 00011 Read the FCCU state (refer to the FCCU_STAT register) [OP3]. 00100 Read the FCCU frozen status flags [OP4]. 00101 Read the FCCU frozen status flags [OP5]. 00110 Read the FCCU frozen status flags [OP6]. 00111 Read the FCCU frozen status flags [OP7]. 01000 Read the FCCU frozen status flags [OP8]. 01001 Read the CF status register (refer to the FCCU_CFS register) [OP9]. 01010 Read the NCF status register (refer to the FCCU_NCFS register) [OP10]. 01011 CF status clear operation in progress (refer to the FCCU_CFS register) [OP11]. 01100 NCF status clear operation in progress (refer to the FCCU_NCFS register) [OP12]. 01101 Clear the freeze status registers (refer to the FCCU_SCFS register) [OP13]. 01110 CONFIG to NORMAL FCCU state (configuration timeout) in progress [OP14]. 01111 Clear the operation status (OPS = Idle, NVML = 0) [OP15]. 10000 Lock the FCCU configuration [OP16]. 10001 Read the ALARM timer (refer to the FCCU_XTMR register) [OP17]. 10010 Read the SMRT timer (refer to the FCCU_XTMR register) [OP18]. 10011 Read the CFG timer (refer to the FCCU_XTMR register) [OP19]. 10100–11111 Reserved [OP20–OP30]. 11111 Run the NVM loading operation (only for test purposes) [OP31]. The software must not modify the OPR field until the completion of the operation (any write operation is ignored until then). After the operation has been completed, the OPS field is set and the OPR field is automatically cleared (OPR = 000). Software must not program these opcodes: • OP11 and OP12: These opcodes are automatically selected when the FCCU_CFS or FCCU_NCFS registers are cleared by a write-clear operation into the related register. • OP14: This opcode is automatically selected when the timeout occurs [FCCU_CFG_TO] during the configuration procedure. In this case, the FCCU state is automatically forced to NORMAL mode setting, the default configuration. In this phase, any write operation to the FCCU configuration registers is inhibited. • OP20–OP30: These are reserved; if you attempt to use them, they return an ABORT response without any side effect). The ABORT response occurs in these cases: • wrong access (missing or wrong key) to the FCCU_CFS register (clear operation OP11) • wrong access (missing or wrong key) to the FCCU_NCFS register (clear operation OP12) • wrong access (missing or wrong key) to the FCCU_CTRL register (OP1, OP2, OP16 operation) • OP1 (CONFIG command) execution when FCCU state ≠ NORMAL or configuration locked • OP20–OP30 (RESERVED operations) execution The OP31 opcode executes the NVM configuration loading via the software. It should be used only for test/debug purposes.

25.7.2 FCCU CTRL Key Register (FCCU_CTRLK)

The FCCU CTRL Key Register (FCCU_CTRLK) implements the key access for the operations OP1, OP2, OP16, and OP31 according to this sequence:

1. Write the key into the FCCU_CTRLK register.
2. Write the FCCU_CTRL register (operations OP1, OP2, OP16, or OP31).

The FCCU_CTRLK register is not readable. A read operation always returns 0x0000_0000. The key must be written by a word (32-bit) data write operation.

Address: Base + 0x04 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	CTRLK[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	CTRLK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-3. FCCU CTRL Key Register (FCCU_CTRLK)

Table 25-4. FCCU_CTRLK field descriptions

Field	Description
CTRLK	Control register key: **CTRLK value** / **Function** 0x9137_56AF — Key for operation OP1 0x825A_132B — Key for operation OP2 0x7ACB_32F0 — Key for operation OP16 0x29AF_8752 — Key for operation OP31

25.7.3 FCCU Configuration Register (FCCU_CFG)

The FCCU Configuration Register (FCCU_CFG) defines the global configuration for the FCCU. The reset values of bits 20:31 are determined by the configuration of these bits described in Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO). The FCCU_CFG register is accessible in write mode only in the CONFIG state.

Address: Base + 0x08 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCCE1	RCCE0	SMRT			
W																
Reset	Dependent on the NVM interface; see Section 25.8.13, FCCU_F interface															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	CM	SM	PS	FOM			FOP					
W																
Reset	Dependent on the NVM interface; see Section 25.8.13, FCCU_F interface															

Figure 25-4. FCCU Configuration Register (FCCU_CFG)

[1] Writable only in the CONFIG state.

Table 25-5. FCCU_CFG field descriptions

Field	Description
RCCE1	RCC1 enable 0 RCC1 disabled. 1 RCC1 enabled. **Note:** In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (destructive reset).
RCCE0	RCC0 enable 0 RCC0 disabled. 1 RCC0 enabled. **Note:** In case a single checker (RCC1 or RCC0 unit) is enabled, both the checkers assert a fault condition (destructive reset).
SMRT	Safe Mode Request Timer 0000 Safe mode request delay = 1 period (RC_{16MHz} clock). 0001 Safe mode request delay = 2 periods (RC_{16MHz} clock). 0010 Safe mode request delay = 4 periods (RC_{16MHz} clock). . . . 1111 Safe mode request delay = 32768 periods (RC_{16MHz} clock).
CM	Configuration mode 0 Configuration labelling: a specific FCCU_F configuration is assigned in CONFIG state. 1 Configuration transparency: the FCCU_F protocol is the same in CONFIG and NORMAL states.
SM	Switching mode 0 FCCU_F protocol (dual-rail, time-switching) slow switching mode. 1 FCCU_F protocol (dual-rail, time-switching) fast switching mode. SM has no effect on the bi-stable protocol.
PS	Polarity selection 0 FCCU_F[1] active high, FCCU_F[0] active high. 1 FCCU_F[1] active low, FCCU_F[0] active low.

Table 25-5. FCCU_CFG field descriptions (continued)

Field	Description
FOM	Fault Output Mode selection 000 Dual-Rail (default state; FCCU_F[1:0]= outputs). 001 Time Switching (FCCU_F[1:0]= outputs). 010 Bi-Stable (FCCU_F[1:0]= outputs). 011 Reserved. 100 Reserved. 101 Test0 (FCCU_F[0] = input, FCCU_F[1] = output). 110 Test1 (FCCU_Ff[0] = output, FCCU_F[1] = output). 111 Test2 (FCCU_F[0] = output, FCCU_F[1] = input). The output level of the FCCU_F[0:1] pins in the test modes is controlled in the FCCU I/O Control Register (FCCU_EINOUT). **Note:** In Test*n* mode, a simple double-stage resynchronization stage is used to resynchronize the FCCU_F[0:1] input/outputs on the system/safe clock.
FOP	Fault Output Prescaler FOP defines the prescaler setting used to generate the FCCU_F protocol frequency according to the following equation: 00 0000 Input clock frequency (RC_{16MHz} clock) is divided by 2048. 00 0001 Input clock frequency (RC_{16MHz} clock) is divided by 4096. 00 0010 Input clock frequency (RC_{16MHz} clock) is divided by 6144. ... 00 1000 Input clock frequency (RC_{16MHz} clock) is divided by 18432 (This is the reset value Id est ~ 868 Hz is the out-of- reset frequency of the FCCU_F[0:1] pins). ... 11 1111 Input clock frequency (RC_{16MHz} clock) is divided by 131072. This equation gives the FCCU_F frequency: $$Fccu0F_{freq} = RC_{16MHz} / (1024 \times (FOP + 1) \times 2)$$

25.7.4 FCCU CF Configuration Register *n* (FCCU_CF_CFG*n*)

The FCCU CF Configuration Register n (FCCU_CF_CFGn) contains the configuration for the user critical fault in terms of fault recovery management.

The configuration depends on the type of signaling following a fault event. Hardware recoverable faults are those events where a previous latching stage captures and holds the physical fault, otherwise, the fault can be lost. All the other faults should be configured as software faults. All the critical faults are software recoverable by default.

The FCCU_CF_CFGn registers are accessible in write mode only in the CONFIG state.

Address: Base + 0x0C Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFC31	CFC30	CFC29	CFC28	CFC27	CFC26	CFC25	CFC24	CFC23	CFC22	CFC21	CFC20	CFC19	CFC18	CFC17	CFC16
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFC15	CFC14	CFC13	CFC12	CFC11	CFC10	CFC9	CFC8	CFC7	CFC6	CFC5	CFC4	CFC3	CFC2	CFC1	CFC0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 25-5. FCCU CF Configuration Register *0* (FCCU_CF_CFG*0*)

1 Writable only in the CONFIG state.

Address: Base + 0x10 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CFC39	CFC38	CFC37	CFC36	CF5335	CFC34	CFC33	CFC32
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 25-6. FCCU CF Configuration Register *1* (FCCU_CF_CFG*1)*

1 Writable only in the CONFIG state.

Table 25-6. FCCU_CF_CFG*n* field descriptions

Field	Description
CFC*x*	Critical fault configuration The critical fault configuration defines the fault recovery mode. Hardware-recoverable faults are self-recovered (status flag clearing) if the root cause has been removed. Software-recoverable faults are recovered (status flag clearing) by software clearing the related status flag. 0 Hardware-recoverable fault. 1 Software-recoverable fault.

The critical faults CF[*j*] and their fault sources are assigned in a one-to-one correspondence. This correspondence applies across all registers of the FCCU that refer to the critical fault(s). Critical faults are listed in Table 25-7.

Table 25-7. FCCU critical faults assignment (CF 0 to 39)

Critical Fault CF[*j*]	Fault Source	Module	Source
FCCU_CF_CFG0			
CF 0	RCCU0[0] (RCCU output) [1]	Core	—
CF 1	RCCU1[0] [1]	Core	—
CF 2	RCCU0[1] [1]	DMACHMUX	—
CF 3	RCCU1[1] [1]	DMACHMUX	—
CF 4	RCCU0[2] [1]	PBRIDGE	—
CF 5	RCCU1[2] [1]	PBRIDGE	—
CF 6	RCCU0[3] [1]	XBAR	Master (FlexRay)
CF 7	RCCU1[3] [1]	XBAR	Master (FlexRay)
CF 8	RCCU0[4] [1]	SRAM	—
CF 9	RCCU1[4] [1]	SRAM	—
CF 10	RCCU0[5]	Flash memory	—
CF 11	RCCU1[5]	Flash memory	—
CF 12	RCCU0[6] [1]	XBAR	Slave (EBI/DRAMC)
CF 13	RCCU1[6] [1]	XBAR	Slave (EBI/DRAMC)
CF 14	SWT_0 (software watchdog)	SWT_0	Software watchdog
CF 15	SWT_1	SWT_1	Software watchdog
CF 16	MCM_NCE_0 (ECC Non-correctable error)	ECSM_0	ECC non correctable
CF 17	MCM_NCE_1	ECSM_1	ECC non correctable
CF 18	ADC_CF0 (A/D internal self-test fault)	ADC_0	ADC self test
CF 19	ADC_CF1	ADC_1	ADC self test
CF 20	STCU_CF (Built In Self-Test critical fault)	STCU	BIST (critical fault)
CF 21	Reserved		
CF 22	SSCM_XFER_ERR (SSCM transfer error occurred during STCU configuration loading phase)	SSCM	Config loading of transfer
CF 23	LSM_DPM_ERR0 (Lock Step Mode <—> Decoupled Parallel Mode runtime switch)	—	LSM/DPM change
CF 24	LSM_DPM_ERR1	—	LSM/DPM change
CF 25	RCCU0[7] [1]	XBAR	Master (XBAR_2)
CF 26	RCCU1[7] [1]	XBAR	Master (XBAR_2)

Table 25-7. FCCU critical faults assignment (CF 0 to 39) (continued)

Critical Fault CF[*j*]	Fault Source	Module	Source
CF 27	STCU (Self-Test Control Unit active during run mode as opposed to during start-up BIST phase)	STCU	BIST (wrong config of STCU)
CF 28:31	Reserved for internal test logic monitoring	STCU	Reserved for internal test logic
FCCU_CF_CFG1			
CF 32	CFlash_0 (Code Flash fatal error)	CFlash_0	Fatal error (error during initialization)
CF 33	CFlash_1	CFlash_1	Fatal error (error during initialization)
CF 34	DFlash (Data Flash fatal error)	DFlash	Fatal error (error during initialization)
CF 35	ADC_CF2 (A/D internal self-test fault)	ADC_2	ADC self test
CF 36	ADC_CF3	ADC_3	ADC self test
CF 37	Fault source JTAG_NPC_CF (JTAG Nexus Port Controller Critical signal)	JTAG/Nexus	Port controller
CF 38–63	Reserved		

1 LSM only.

25.7.5 FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0)

The FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0) contains the configuration of each non-critical fault in terms of fault recovery management. The configuration depends on the type of signaling of a fault event. Hardware recoverable faults should be configured only if a previous latching stage captures and holds the physical fault. Otherwise, the fault can be lost. All other faults should be configured as software faults.

The FCCU_NCF_CFG0 register is accessible in write mode only in the CONFIG state.

Address: Base + 0x1C Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFC31	NCFC30	NCFC29	NCFC28	NCFC27	NCFC26	NCFC25	NCFC24	NCFC23	NCFC22	NCFC21	NCFC20	NCFC19	NCFC18	NCFC17	NCFC16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFC15	NCFC14	NCFC13	NCFC12	NCFC11	NCFC10	NCFC9	NCFC8	NCFC7	NCFC6	NCFC5	NCFC4	NCFC3	NCFC2	NCFC1	NCFC0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 25-7. FCCU NCF Configuration Register 0 (FCCU_NCF_CFG0)

[1] Writable only in the CONFIG state.

Table 25-8. FCCU_NCF_CFG0 field descriptions

Field	Description
NCFC*x*	Non-critical fault configuration 0 Hardware recoverable fault. 1 Software recoverable fault. The non-critical fault configuration defines the fault recovery mode. Hardware recoverable faults are self recovered (status flag clearing and related) if the root cause has been removed. Software recoverable faults are recovered (status flag clearing) by software clearing of the related status flag. This register can be read and written by software.

The non-critical faults and their sources are assigned in a one-to-one correspondence that also applies to the FCCU NCF Status Register 0 (FCCU_NCFS0), FCCU NCF Enable Register 0 (FCCU_NCFE0), and FCCU NCF Timeout Enable Register 0 (FCCU_NCF_TOE0) bit fields. The non-critical faults are listed in Table 25-9.

Table 25-9. Non-critical fault assignment for FCCU_NCF_CFG0

Non-critical fault NCF[*j*]	Fault source	Module	Source
NCF 0	Core Watchdog 0	Core_0	Core watchdog
NCF 1	Core Watchdog 1	Core_1	Core watchdog
NCF 2	FM_PLL_0 (loss of lock)	FMPLL_0	Loss of lock
NCF 3	FM_PLL_1 (loss of lock)	FMPLL_1	Loss of lock
NCF 4	CMU_0 (Loss of XOSC clock)	CMU_0	Loss of XOSC
NCF 5	CMU_0 (SYSCLK frequency out of range)	CMU_0	SYSCLK out of range
NCF 6	CMU_1 (MOTC_CLK frequency out of range)	CMU_1	MOTC_CLK out of range
NCF 7	CMU_2 (FRPE_CLK frequency out of range)	CMU_2	FlexRay clock out of range
NCF 8	MCM_ECN_0 (ECC 1-bit error correction notification)	ECSM_0	ECC -bit error
NCF 9	MCM_ECN_1 (ECC 1-bit error correction notification)	ECSM_1	ECC -bit error
NCF 10	ADC_NCF_0 (A/D Internal self test non-critical fault)	ADC_0	ADC self test
NCF 11	ADC_NCF_1 (A/D Internal self test non-critical fault)	ADC_1	ADC self test
NCF 12	STCU_NCF (BIST results non-critical faults)	STCU	BIST (non critical)

Table 25-9. Non-critical fault assignment for FCCU_NCF_CFG0 (continued)

Non-critical fault NCF[*j*]	Fault source	Module	Source
NCF 13:18	Reserved		
NCF 19	FLEXR_ENC (FlexRAY ECC 1-bit error correction notification)	FlexRay	ECC 1-bit error
NCF 20	FLEXR_NCN (FlexRay ECC non correctable error)	FlexRay	ECC non correctable
NCF 21	MC_ME (Software device reset)	Mode Entry	Software device reset (reset request from MC_ME)
NCF 22:24	Reserved		
NCF 25	ADC_NCF_2 (A/D Internal self test non-critical fault)	ADC_2	ADC self test
NCF 26	ADC_NCF_3 (A/D Internal self test non-critical fault)	ADC_3	ADC self test
NCF 27:31	Reserved		

25.7.6 FCCU CFS Configuration Register *n* (FCCU_CFS_CFG*n*)

The FCCU CFS Configuration Register *n* (FCCU_CFS_CFG*n*) contains the configuration of each critical fault in terms of fault response (short or long functional reset) when it is the root cause for the FAULT state transition. Table 25-7 lists the critical faults. The FCCU_CFS_CFG*n* register is accessible in write mode only in the CONFIG state.

Address: Base + 0x2C (FCCU_CFS_CFG0) Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC15		CFSC14		CFSC13		CFSC12		CFSC11		CFSC10		CFSC9		CFSC8	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC7		CFSC6		CFSC5		CFSC4		CFSC3		CFSC2		CFSC1		CFSC0	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Figure 25-8. FCCU CFS Configuration Register 0 (FCCU_CFS_CFG0)

1 Writable only in the CONFIG state.

Address: Base + 0x30 (FCCU_CFS_CFG1) Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC31		CFSC30		CFSC29		CFSC28		CFSC27		CFSC26		CFSC25		CFSC24	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC23		CFSC22		CFSC21		CFSC20		CFSC19		CFSC18		CFSC17		CFSC16	
Reset	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

Figure 25-9. FCCU CFS Configuration Register 1 (FCCU_CFS_CFG1)

[1] Writable only in the CONFIG state.

Address: Base + 0x34 (FCCU_CFS_CFG2) Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CFSC47		CFSC46		CFSC45		CFSC44		CFSC43		CFSC42		CFSC41		CFSC40	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CFSC39		CFSC38		CFSC37		CFSC36		CFSC35		CFSC34		CFSC33		CFSC32	
Reset	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0

Figure 25-10. FCCU CFS Configuration Register 2 (FCCU_CFS_CFG2)

[1] Writable only in the CONFIG state.

Table 25-10. FCCU_CFS_CFG*n* field descriptions

Field	Description
CFSC*x*	Critical fault state configuration 00 No reset response. 01 Short functional reset (FAULT state response). 10 Long functional reset (FAULT state response). 11 No reset response. This register can be read and written by software.

25.7.7 FCCU NCFS Configuration Register *n* (FCCU_NCFS_CFG*n*)

The FCCU NCFS Configuration Register *n* (FCCU_NCFS_CFG*n*) contains the configuration of each non-critical fault in terms of fault response (short or long functional reset) when it is the root cause for the FAULT state transition. Table 25-9 lists the noncritical faults. The FCCU_NCFS_CFG*n* register is accessible in write mode only in the CONFIG state.

Address: Base + 0x4C (FCCU_NCFS_CFG0) Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFSC15		NCFSC14		NCFSC13		NCFSC12		NCFSC11		NCFSC10		NCFSC9		NCFSC8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFSC7		NCFSC6		NCFSC5		NCFSC4		NCFSC3		NCFSC2		NCFSC1		NCFSC0	
Reset	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Figure 25-11. FCCU NCFS Configuration Register 0 (FCCU_NCFS_CFG0)

[1] Writable only in the CONFIG state.

Address: Base + 0x50 (FCCU_NCFS_CFG1) Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFSC31		NCFSC30		NCFSC29		NCFSC28		NCFSC27		NCFSC26		NCFSC25		NCFSC24	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFSC23		NCFSC22		NCFSC21		NCFSC20		NCFSC19		NCFSC18		NCFSC17		NCFSC16	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-12. FCCU NCFS Configuration Register 1 (FCCU_NCFS_CFG1)

[1] Writable only in the CONFIG state.

Table 25-11. FCCU_NCFS_CFG*n* field descriptions

Field	Description
NCFSC*x*	Non-critical fault state configuration 00 No reset response. 01 Short functional reset (FAULT state response). 10 Long functional reset (FAULT state response). 11 No reset response. This register can be read and written by software.

25.7.8 FCCU CF Status Register *n* (FCCU_CFS*n*)

The FCCU CF Status Register *n* (FCCU_CFS*n*) contains the latched fault indication collected from the critical fault sources (FCCU CF[*j*] as in Table 25-7). Faults are latched even if the FCCU is in the CONFIG state and independently from the enabling or responses programmed for the critical fault.

No responses are executed until the FCCU moves into the NORMAL state. The FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if a critical fault is triggered. The status bits of

the FCCU_CFS*n* register, when configured in the corresponding FCCU_CF_CFG*n* register as software recoverable faults, can be cleared by this locked sequence:

1. Write the proper key into the FCCU_CFK register.
2. Clear the FCCU_CFSn[CFSx] field → the opcode OP11 is automatically set into the FCCU_CTRL[OPR] field.
3. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
4. Read the FCCU_CFS*n* register in order to verify the effective deletion and in case of failure to repeat the sequence.

As a result of the above sequence, in addition the FAULT interface provides support to clear a fault latched directly in the FAULT root, asserting an ‘internal’ clear signal as the software clears the corresponding Critical Fault Status bit.

The FCCU moves from the FAULT state into the NORMAL state if the source fault that caused the transition to the FAULT state has been removed (hardware recoverable fault) or cleared via software (software recoverable fault). Concurrently, the FAULT interface provides support to clear the FAULT root. In case of nested faults that are not all recovered, the FCCU remains in the FAULT state or moves into the ALARM state.

The software application executes the FCCU_CFS*n* read operation by this sequence:

1. Program the FCCU_CTRL[OPR] field for OP9 operation.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
3. Read the FCCU_CFS*n* register.

These errors are ignored:

- Writing a wrong key into the FCCU_CFK register.
- Attempting to clear a hardware recoverable error.

Address: Base + 0x6C (FCCU_CFS0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS31	CFS30	CFS29	CFS28	CFS27	CFS26	CFS25	CFS24	CFS23	CFS22	CFS21	CFS20	CFS19	CFS18	CFS17	CFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFS15	CFS14	CFS13	CFS12	CFS11	CFS10	CFS9	CFS8	CFS7	CFS6	CFS5	CFS4	CFS3	CFS2	CFS1	CFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-13. FCCU CF Status Register 0 (FCCU_CFS0)

Address: Base + 0x70 (FCCU_CFS1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CFS39	CFS38	CFS37	CFS36	CFS35	CFS34	CFS33	CFS32
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-14. FCCU CF Status Register 1 (FCCU_CFS1)

Table 25-12. FCCU_CFS*n* field descriptions

Field	Description
CFS*x*	Critical fault status 0 No critical fault latched. 1 Critical fault latched. The status bits related to the critical fault configured as hardware recoverable faults are read-only and the flag is self-cleared when the fault source is removed. The status bits related to the critical fault configured as software recoverable faults are write-clear and the software application can recover from a faulty condition.

25.7.9 FCCU CF Key Register (FCCU_CFK)

The FCCU_CFK register implements the key access to clear the status flags of the FCCU_CFS*n* register.

The status bits of the FCCU_CFS*n* register, when configured in the corresponding FCCU_CF_CFG*n* register as software recoverable faults, can be cleared by this locked sequence:

1. Write the CFK key into the FCCU_CFK register.
2. Clear the FCCU_CFS*n*[CFS*x*] field.

NOTE

The key must be written for each FCCU_CFS*n* clear operation.

The FCCU_CFK register is not readable. A read operation always returns 0x0000_0000.

Address: Base + 0x7C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	CFK[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	CFK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-15. FCCU CF Key Register (FCCU_CFK)

Table 25-13. FCCU_CFK field descriptions

Field	Description
CFK	Critical fault key = 0x618B_7A50

25.7.10 FCCU NCF Status Register 0 (FCCU_NCFS0)

The FCCU NCF Status Register 0 (FCCU_NCFS0) contains the latched fault indication collected from the non-critical fault sources (See Table 25-9 FCCU NCF[*j*]). Faults are latched also in the CONFIG state and independently from the enabling or responses programmed for the NCF.

No responses are executed until the FCCU moves to the NORMAL state.

FCCU reacts and moves from the NORMAL state into the ALARM state only if the respective enable bit for a fault is set in the FCCU_NCFE*x* register and the respective enable bit for the timeout is set in the FCCU_TOE*x* register.

FCCU reacts and moves from the NORMAL or ALARM state into the FAULT state if the respective enable bit for a fault is set in the FCCU_NCFE*x* register and the respective enable bit for the timeout is disabled in the FCCU_TOE*x* register.

FCCU reacts and moves from the ALARM state into the FAULT state if the timeout (FCCU_TO register) is elapsed before the fault is recovered.

The timeout count is stopped only when the FCCU returns to the NORMAL state.

The status bits of the FCCU_NCFS0 register, when configured in the corresponding FCCU_NCFS_CFG*n* register as software recoverable faults, can be cleared by this locked sequence:

1. Write the proper key into the FCCU_NCFK register.
2. Clear the FCCU_NCFS0[NCFS*x*] field → the opcode OP12 is automatically set into the FCCU_CTRL[OPR] field.
3. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
4. Read the FCCU_NCFS0 register in order to verify the effective deletion and in case of failure to repeat the sequence.

As a result of the above sequence, in addition the FAULT interface provides support to clear a fault latched directly in the FAULT root, asserting an ‘internal’ clear signal as the software clears the corresponding Non-critical Fault Status Bit.

The FCCU moves from the FAULT or ALARM state into the NORMAL state if all the source faults that caused the transition into the FAULT state have been removed (hardware recoverable fault) or cleared via software (software recoverable fault). In case of nested faults that are not all recovered, the FCCU remains in the FAULT or ALARM state.

The software application executes the FCCU_NCFS0 read operation by this sequence:

1. Set the OP10 operation into the FCCU_CTRL[OPR] field.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
3. Read the FCCU_NCFS0 register.

In case of re-configuration of the FCCU (CONFIG state), before returning to the NORMAL state the pending status bits in the FCCU_NCFS0 must be cleared in order to avoid a false transition to the ALARM/FAULT state.

These errors are ignored:

- Writing a wrong key into the FCCU_NCFK register.
- Attempting to clear a hardware recoverable error.

Address: Base + 0x80 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCFS31	NCFS30	NCFS29	NCFS28	NCFS27	NCFS26	NCFS25	NCFS24	NCFS23	NCFS22	NCFS21	NCFS20	NCFS19	NCFS18	NCFS17	NCFS16
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NCFS15	NCFS14	NCFS13	NCFS12	NCFS11	NCFS10	NCFS9	NCFS8	NCFS7	NCFS6	NCFS5	NCFS4	NCFS3	NCFS2	NCFS1	NCFS0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-16. FCCU NCF Status Register 0 (FCCU_NCFS0)

Table 25-14. FCCU_NCFS0 field descriptions

Field	Description
NCFS*x*	Non-critical fault status 0 No non-critical fault latched. 1 Non-critical fault is latched. The status bits related to the non-critical fault configured as hardware recoverable faults are read-only and the flag is self-cleared when the fault source (fccu_ncf[*j*] input) is removed. The status bits related to the critical fault configured as software recoverable faults are write-clear and the software application can recover from a faulty condition.

25.7.11 FCCU NCF Key Register (FCCU_NCFK)

The FCCU NCF Key Register (FCCU_NCFK) implements the key access to clear the status flags of the FCCU_NCFS0 register. The status bits of the FCCU_NCFS0 register, when configured in the corresponding FCCU_NCFS_CFG*n* register as software recoverable faults, can be cleared by this locked sequence:

1. Write the key into the FCCU_NCFK register.
2. Clear the FCCU_NCFS0[NCFS*x*] field.

NOTE

The key must be written for each FCCU_NCFS0 clear operation.

The FCCU_NCFK register is not readable. A read operation always returns 0x0000_0000.

Address: Base + 0x90 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	NCFK[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	NCFK[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-17. FCCU NCF Key Register (FCCU_NCFK)

Table 25-15. FCCU_NCFK field descriptions

Field	Description
NCFK	Critical fault key = 0xAB34_98FE.

25.7.12 FCCU NCF Enable Register 0 (FCCU_NCFE0)

The FCCU NCF Enable Register 0 (FCCU_NCFE0) enables the non-critical fault sources (see Table 25-9 FCCU NCF[*j*]) to allow a transition from the NORMAL into the FAULT or ALARM state. In case of fault masking, the respective status bit in the FCCU_NCFS0 register is set (for debugging purposes), but the response is masked.

The FCCU_NCFE0 register is accessible in write mode in the CONFIG state only.

Address: Base + 0x94 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFE31	NCFE30	NCFE29	NCFE28	NCFE27	NCFE26	NCFE25	NCFE24	NCFE23	NCFE22	NCFE21	NCFE20	NCFE19	NCFE18	NCFE17	NCFE16
Reset	0	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFE15	NCFE14	NCFE13	NCFE12	NCFE11	NCFE10	NCFE9	NCFE8	NCFE7	NCFE6	NCFE5	NCFE4	NCFE3	NCFE2	NCFE1	NCFE0
Reset	0	0	0	1	1	1	0	0	1	1	1	1	1	1	1	1

Figure 25-18. FCCU NCF Enable Register 0 (FCCU_NCFE0)

[1] Writable only in the CONFIG state.

Table 25-16. FCCU_NCFE0 field descriptions

Field	Description
NCFE*x*	Non-critical fault enable. 0 No actions following the respective non-critical fault assertion. 1 FCCU moves to ALARM or FAULT state following the respective non-critical fault assertion.

25.7.13 FCCU NCF Timeout Enable Register 0 (FCCU_NCF_TOE0)

The FCCU NCF Timeout Enable Register 0 (FCCU_NCF_TOE0) enables a transition from the NORMAL state into the ALARM state if the respective non-critical fault is enabled (NCFE*x* and NCFTOE*x* are set). In case the respective timeout is disabled (NCFTOEx is cleared) and the non-critical fault is enabled (NCFE*x* is set), the FCCU moves into the FAULT state if the related non-critical fault is asserted. The timer (preset with the timeout value defined by FCCU_TO register) is started when the FCCU moves into the ALARM state. If the fault is not recovered within the timeout period, the FCCU moves from the ALARM state to the FAULT state.

The FCCU_NCFTOE0 register is accessible in write mode in the CONFIG state only.

Address: Base + 0xA4 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	NCFTOE31	NCFTOE30	NCFTOE29	NCFTOE28	NCFTOE27	NCFTOE26	NCFTOE25	NCFTOE24	NCFTOE23	NCFTOE22	NCFTOE21	NCFTOE20	NCFTOE19	NCFTOE18	NCFTOE17	NCFTOE16
Reset	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	NCFTOE15	NCFTOE14	NCFTOE13	NCFTOE12	NCFTOE11	NCFTOE10	NCFTOE9	NCFTOE8	NCFTOE7	NCFTOE6	NCFTOE5	NCFTOE4	NCFTOE3	NCFTOE2	NCFTOE1	NCFTOE0
Reset	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 25-19. FCCU NCF Timeout Enable Register 0 (FCCU_NCF_TOE0)

[1] Writable only in the CONFIG state.

Table 25-17. FCCU_NCF_TOE0 field descriptions

Field	Description
NCFTOE*x*	Non-critical fault timeout enable 0 FCCU moves into the FAULT state if the respective fault is disabled (FCCU_NCFE0[NCFE*x*] is cleared). 1 FCCU moves into the ALARM state if the respective fault is enabled (FCCU_NCFE0[NCFE*x*] is set).

25.7.14 FCCU NCF Timeout Register (FCCU_NCF_TO)

The FCCU NCF Timeout Register (FCCU_NCF_TO) defines the preset value of the timer for the recovery of the non-critical faults (enabled). Once FCCU enters the ALARM state, following the assertion of a non-critical fault enabled (NCFE*x* and NCFTOE*x* are set), the timer starts the countdown.

If the fault is not recovered within the timeout period, the FCCU moves from the ALARM state to the FAULT state. If fault recovery is intended before the FCCU moves to the FAULT state, then software must ensure that the FCCU_CFG[FCCU_NCF_TO] and FCCU_CFG[SMRT] bits are appropriately programmed to give enough time for the core to respond to and clear the fault.

The FCCU_NCF_TO register is accessible in write mode in the CONFIG state only.

Address: Base + 0xB4 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TO[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TO[15:0]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 25-20. FCCU NCF Timeout Register (FCCU_NCF_TO)

1 Writable only in the CONFIG state.

Table 25-18. FCCU_NCF_TO field descriptions

Field	Description
TO	Non-critical fault timeout $\text{Timeout} = (\text{TO}) \times T_{\text{RC16MHz}}$

25.7.15 FCCU CFG Timeout Register (FCCU_CFG_TO)

The FCCU CFG Timeout Register (FCCU_CFG_TO) defines the preset value of the watchdog timer for the recovery from the CONFIG state. Once the FCCU enters the CONFIG state following a software request (OP1 opcode), the watchdog timer is initialized and starts the countdown if the reset is not asserted.

If the configuration is not completed within the timeout period, the FCCU moves automatically from the CONFIG state to the NORMAL state and the default value for the configuration register is restored. The watchdog timeout is clocked with the RC oscillator clock (16 MHz). The default timeout value is 4,096 ms.

The FCCU_CFG_TO register is accessible in write mode, in any state excluded the CONFIG state as follows the execution of the OP1 opcode (NORMAL to CONFIG state) and until the completion of the OP2 opcode (CONFIG to NORMAL state).

In case of watchdog timeout, the FCCU_CFG_TO register is not accessible until the OP14 operation (CONFIG to NORMAL) has been completed.

Address: Base + 0xB8 Access: User read/write[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	TO		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 25-21. FCCU CFG Timeout Register (FCCU_CFG_TO)

[1] Writable only in the CONFIG state.

Table 25-19. FCCU_CFG_TO field descriptions

Field	Description
TO	Configuration timeout $Timeout = T_{RC16MHz} \times 2^{(TO+10)}$ 000 Timeout = 64 µs. . . . 111 Timeout = 8,192 ms.

25.7.16 FCCU I/O Control Register (FCCU_EINOUT)

The FCCU I/O Control Register (FCCU_EINOUT) allows these operations typically in NORMAL state:

- Controlling the FCCU_F[1] pin output level when the FCCU is configured in “Test1” or “Test0” fault output mode (FCCU_CFG[FOM])
- Controlling the FCCU_F[0] pin output level when the FCCU is configured in “Test1” or “Test2” fault output mode (FCCU_CFG[FOM])
- Observing the FCCU_F[1:0] pins input level

NOTE: fccu0 $F_{in/out}[j]$ semantic

The FCCU_F[0:1] system pins are bidirectional. In the register fields of the FCCU I/O Control Register (FCCU_EINOUT), the subscript F_{out} or F_{in} is added to the pin name to distinguish the output from the input path.

Table 25-20. Bi-stable encoding

Mode = FCCU_CFG[FOM]	fccu0 $F_{OUT}[0]$	fccu0 $F_{OUT}[1]$
Test1	output	output
Test2	output	input
Test0	input	output

NOTE

Due to the re-synchronization stage of the FCCU_F interface, there is a latency of a few safe clock (RC 16 MHz clock) cycles following a write/read operation of the FCCU_EINOUT register.

Address: Base + 0xBC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	$F_{in}1$	$F_{IN}0$	0	$F_{out}1$	$F_{out}0$
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-22. FCCU I/O Control Register (FCCU_EINOUT)

Table 25-21. FCCU_EINOUT field descriptions

Field	Description
$F_{in}1$	FCCU Error input 1 0 When fccu0 $F_{in}[1]$ = 0. 1 When fccu0 $F_{in}[1]$ = 1. The $F_{in}1$ Field captures the respective fccu0 $F_{in}[1]$ input signal.
$F_{in}0$	FCCU Error input 0 0 When fccu0 $F_{in}[0]$ = 0. 1 When fccu0 $F_{in}[0]$ = 1. The $F_{in}0$ captures the fccu0 $F_{in}[0]$ input signal.
$F_{out}1$	FCCU Error out 1 (significant only if the FCCU_CFG[FOM] = Test1 or Test0 ≥ fccu_1] configured in output mode) 0 Force fccu0 $F_{out}[1]$ = 0. 1 Force fccu0 $F_{out}[1]$ = 1. The $F_{out}1$ sets/clears the respective fccu0 $F_{out}[1]$ output signal if FCCU_CFG[FOM] = 110 or 101. Otherwise, it is a “don’t care” value.
$F_{out}0$	FCCU Error out 0 (significant only if the FCCU_CFG[FOM] = Test1 or Test2 ≥ fccu_0] configured in output mode) 0 Force fccu0 $F_{out}[0]$ = 0. 1 Force fccu0 $F_{out}[0]$ = 1. The $F_{out}0$ sets/clears the respective fccu0 $F_{out}[0]$ output signal if FCCU_CFG[FOM] = 110 or 111. Otherwise, it is a “don’t care” value.

25.7.17 FCCU Status Register (FCCU_STAT)

The FCCU Status Register (FCCU_STAT) provides the FCCU status for debugging/test purposes. The FCCU finite state machine operates by the RC oscillator clock ($RC16_{MHz}$ safe clock) asynchronous from the system clock (IP Bus clock). Therefore the FCCU status read operation requires a safe mechanism

operated by a hardware/software synchronization sequence. The software application executes an FCCU status read operation by this sequence:

1. Set the OP3 operation into the FCCU_CTRL[OPR] field.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
3. Read the FCCU status (FCCU_STAT register).

Address: Base + 0xC0 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	STATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-23. FCCU Status Register (FCCU_STAT)

Table 25-22. FCCU_STAT field descriptions

Field	Description
STATUS	FCCU Status 000 NORMAL state. 001 CONFIG state. 010 ALARM state. 011 FAULT state. Other UNKNOWN state.

25.7.18 FCCU SC Freeze Status Register (FCCU_SCFS)

The FCCU_SCFS register contains the status of the RCC checkers.

NOTE: RCC and RCCU

The RCC are the two FCCU's internal redundancy control checkers (see Figure 25-1). These should not be confused with the RCCU, which are the are the seven pairs of Redundancy Control Checker units used on the MPC5675K).

This register is for test/debug purposes.

Two interrupt lines (irq_rccs_b[1:0] mapped to IRQ253 and IRQ252, respectively) are asserted by the correspondent status bits (FCCU_SCFS[RCCS*x*]).

The software application executes the FCCU_SCFS read operation by the following sequence:

1. Set the OP8 operation into the FCCU_CTRL[OPR] field.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
3. Read the FCCU_SCFS register.

The software application executes the FCCU_SCFS clear operation by the following sequence:

1. Set the OP13 operation into the FCCU_CTRL[OPR] field.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).

NOTE

All the freeze registers are cleared by this operation.

NOTE

The RCC checkers must be disabled (FCCU_CFG[RCCE*x*] = 00b) during a clear operation. Otherwise, the RCCS*x* flags are asserted.

Address: Base + 0x00D4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RCCS1	RCCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-24. FCCU SC Freeze Status Register (FCCU_SCFS)

Table 25-23. FCCU_SCFS field descriptions

Field	Description
RCCS1	RCC1 Status (valid only if SCF =1) 0 No miscompares. 1 A miscompare has been detected. The interrupt line irq_rccs[1] is asserted. **Note:** This bit can be cleared by writing a 0. Writing a 1 has no effect.
RCCS0	RCCS0: RCC0 Status (valid only if SCF =1) 0 No miscompares. 1 A miscompare has been detected. The interrupt line irq_rccs[0] is asserted. **Note:** This bit can be cleared by writing a 0. Writing a 1 has no effect.

25.7.19 FCCU CF Fake Register (FCCU_CFF)

The FCCU CF Fake Register (FCCU_CFF) contains a unique code to set a “critical fault” in mutually exclusive mode by the external FAULT interface. It allows the software emulation of the critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and response. The fault injection mechanism is optional. The response following a fake critical fault cannot be masked. The FCCU_CFF is a write-only register with a set of codes corresponding to each critical fault injection.

Address: Base + 0xD8 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W										FCFC						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-25. FCCU CF Fake Register (FCCU_CFF)

Table 25-24. FCCU_CFF field descriptions

Field	Description
FCFC	Fake critical fault code 0x00 Fake critical fault injection at critical fault source 0 0x01 Fake critical fault injection at critical fault source 1 0x02 Fake critical fault injection at critical fault source 2 0x03 Fake critical fault injection at critical fault source 3 0x04 Fake critical fault injection at critical fault source 4 0x05 Fake critical fault injection at critical fault source 5 0x06 Fake critical fault injection at critical fault source 6 0x07 Fake critical fault injection at critical fault source 7 0x08 Fake critical fault injection at critical fault source 8 0x09 Fake critical fault injection at critical fault source 9 0x0A Fake critical fault injection at critical fault source 10 0x0B Fake critical fault injection at critical fault source 11 0x0C Fake critical fault injection at critical fault source 12 0x0D Fake critical fault injection at critical fault source 13 0x19 Fake critical fault injection at critical fault source 25 0x1A Fake critical fault injection at critical fault source 26 All other values: No fault injection

25.7.20 FCCU NCF Fake Register (FCCU_NCFF)

The FCCU NCF Fake Register (FCCU_NCFF) contains a unique code to set a “non-critical fault” in mutually exclusive mode by the external FAULT interface. It allows the SW emulation of the non-critical faults, by the injection of the fault directly in the FAULT root, in order to verify the entire path and response. The fault injection mechanism is optional. The response following a fake non-critical fault can be masked. The FCCU_NCFF is a write-only register with a set of codes corresponding to each non-critical fault injection.

Address: Base + 0xDC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W										FCNFC						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-26. FCCU NCF Fake Register (FCCU_NCFF)

Table 25-25. FCCU_NCFF field descriptions

Field	Description
FNCFC	Fake non-critical fault code 0x0C Fake non-critical fault injection at non-critical fault source 12 others No fault injection

25.7.21 FCCU IRQ Status Register (FCCU_IRQ_STAT)

The FCCU IRQ Status Register (FCCU_IRQ_STAT) defines the FCCU interrupt status register related to these events:

- Configuration timeout error
- Alarm interrupt
- NMI interrupt

The external alarm interrupt line is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and the respective enable bit of the FCCU_IRQ_EN register is also set.

The NMI and ALARM interrupts are asserted and cleared according to the FCCU state. The status bits of the FCCU_IRQ_STAT trace the status of the related interrupt lines.

Address: Base + 0xE0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	NMI_STAT	ALRM_STAT	CFG_TO_STAT
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-27. FCCU IRQ Status Register (FCCU_IRQ_STAT)

Table 25-26. FCCU_IRQ_STAT field descriptions

Field	Description
NMI_STAT	NMI Interrupt Status 0 NMI interrupt is OFF. 1 NMI interrupt is ON. NMI_STAT can be only read by software.
ALRM_STAT	Alarm Interrupt Status 0 Alarm interrupt is OFF. 1 Alarm interrupt is ON. ALRM_STAT can be only read by software.
CFG_TO_STAT	Configuration Timeout Status 0 No configuration timeout error. 1 Configuration timeout error. CFG_TO_STAT can be read and cleared by software.

25.7.22 FCCU IRQ Enable Register (FCCU_IRQ_EN)

The FCCU IRQ Enable Register (FCCU_IRQ_EN) defines the FCCU interrupt enable register related to this event:

- Configuration timeout error

The external interrupt alarm line is asserted if any interrupt status bit of the FCCU_IRQ_STAT is set and the respective enable bit of the FCCU_IRQ_EN register is also set.

Address: Base + 0xE4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CFG_TO_IEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-28. FCCU IRQ Enable Register (FCCU_IRQ_EN)

Table 25-27. FCCU_IRQ_EN field descriptions

Field	Description
CFG_TO_IEN	Configuration Timeout Interrupt Enable 0 Configuration timeout interrupt disabled. 1 Configuration timeout interrupt enabled. This bit can be read and written by software.

25.7.23 FCCU XTMR Register (FCCU_XTMR)

The FCCU XTMR Register (FCCU_XTMR) contains the read values of the Alarm, Watchdog, or Safe Mode Request Timer. These timers are clocked on the safe clock.

The software application executes the timer read operation using this sequence:

1. Set the OP17 or OP18 or OP19 operation into the FCCU_CTRL[OPR] field.
2. Wait for the completion of the operation (FCCU_CTRL[OPS] field).
3. Read the FCCU_XTMR register.

Table 25-28. Timer state/value

Timer	CONFIG State	NORMAL State	ALARM State	FAULT State
ALARM	0x0000_0000	Initial value	Running	Idle/End of count
SMRT	0x0000_0001	Initial value	—	Running/End of count
CFG	Running	0x0001_FFFF	0x0001_FFFF	0x0001_FFFF

Address: Base + 0xE8 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	XTMR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	XTMR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-29. FCCU XTMR Register (FCCU_XTMR)

Table 25-29. FCCU_XTMR field descriptions

Field	Description
XTMR	Alarm/Watchdog/Safe request timer. The current timer value is measured in safe clock (RC_{16MHz} cycles. These bits can be read by the software.

25.7.24 FCCU MCS Register (FCCU_MCS)

The FCCU MCS Register (FCCU_MCS) contains a queue of the last four chip modes. MCS0 is the latest one, while MCS3 is the oldest one. In addition, a qualifier indicates if the FCCU is in the FAULT state when the chip mode has been captured. The chip mode is synchronous to the system clock (IP Bus clock) and provided by a different module while the FCCU state is synchronous to the safe clock (RC_{16MHz} safe clock), therefore some uncertainty due to synchronization must be considered regarding the FAULT state indication.

Address: Base + 0xEC Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VL3	FS3	0	0	MCS3				VL2	FS2	0	0	MCS2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VL1	FS1	0	0	MCS1				VL0	FS0	0	0	MCS0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-30. FCCU MCS Register (FCCU_MCS)

Table 25-30. FCCU_MCS field descriptions

Field	Description
VL*x*	Valid. Indicates that the corresponding MCS*x* and FS*x* fields are valid. 0 MCS*x*, FS*x* fields are not significant. 1 MCS*x*, FS*x* fields are significant. These bits can be read by software.
FS*x*	Fault status. Indicates that the corresponding MCS*x* field has been captured when the FCCU is in FAULT state. 0 MCS*x* field captured in any state different from the FAULT state. 1 MCS*x* field captured in FAULT state. These bits can be read by software.
MCS*x*	Chip mode. MCS*x* is the chip mode. The encodings of MCS*x* are the same as those listed in the ME_MCTL[TARGET_MODE] field description (Table 35-5 in Section 35.3.2.2, Mode Control Register (ME_MCTL)). MCS0 = latest state. MCS3 = oldest state. On any chip mode change, the previous chip modes are shifted (MCS3 = MCS2, MCS2= MCS1, MCS1= MCS0) and the latest one is captured in MCS0. These bits can be read by software.

25.8 Functional description

25.8.1 Definitions

In general, these definitions are applicable for fault management:

- Hardware recoverable fault: the fault indication is an edge-triggered and level-sensitive signal that remains asserted until the fault cause is asserted. That is, if 0 on the fault signal indicates fault, then the status flags are valid until the fault line is ‘0’. The status is automatically cleared when the fault signal goes to 1. Typically the fault signal is latched in a external module at the FCCU. The FCCU state transitions are consequently executed on the state changes of the input fault signal (fccu_cf[] or fccu_ncf[]). No software intervention in the FCCU is required to recover the fault condition.

- Software recoverable fault: the fault indication is a signal asserted without a defined time duration. The fault signal is captured in the FCCU. The fault recovery is executed following a software recovery procedure (status/flag register clearing).:

These types of resets are applicable (see Chapter 46, Reset Generation Module (MC_RGM)):

- Destructive reset: any type of reset related to a power failure condition that implies a complete system reinitialization
- Long functional reset: implies initialization of the flash and most digital circuitry (with some exceptions FCCU, STCU)
- Short functional reset: implies initialization of most of the digital circuitry (with some exceptions FCCU, STCU)

25.8.2 Finite State Machine (FSM) description

The FCCU module functionality is depicted by the FSM diagram shown in Figure 25-31.

Four basic states are identified:

- CONFIG: the configuration state is used only to modify the default configuration of the FCCU. A sub-set of the FCCU registers, dedicated to define the FCCU configuration (global configuration, responses to fault, timeout, non-critical fault masking) can be accessed in write mode only in the CONFIG state.

  The CONFIG state is accessible only from NORMAL state and if the configuration is not locked. The configuration lock can be disabled only by a global reset of the FCCU. The configuration shall always be locked when running a safety critical application.

  The CONFIG to NORMAL state transition can be executed by software or else automatically following a timeout condition of the watchdog.

  The incoming faults, occurring during the configuration phase (CONFIG state) are latched in order to process them when the FCCU is moved into the NORMAL state, according to the selected configuration.
- NORMAL: the FCCU operating state when no faults are occurring. It is also the default state on the reset exit. Following one of these events:
  - Critical faults → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the timeout disabled → the FCCU moves to the FAULT state
  - Unmasked non-critical faults with the timeout enabled → the FCCU moves to the ALARM state
  - Masked non-critical faults → the FCCU stays in NORMAL state
- ALARM: the FCCU moves into the ALARM state when an unmasked non-critical fault occurs and the timeout is enabled. The transition to the ALARM state goes along with an interrupt. By definition, this fault may be recovered within a programmable timeout period, before it generates a transition to the FAULT state. The timeout is reinitialized if the FCCU state moves to the NORMAL state. The timeout is stopped if the FCCU state moves to the FAULT state due to a critical fault occurring when the FCCU is in ALARM state. The timeout restarts following the recovery from the FAULT state.

- FAULT: the FCCU moves into the FAULT state when one of these conditions occurs:
  - Critical fault
  - Timeout related to a non-critical fault when the FCCU is in the ALARM state
  - Unmasked non-critical faults with the timeout disabled
- The transition from NORMAL/ALARM state is accompanied by the generation of:
  - NMI interrupt
  - FCCU_F[0:1] signaling
  - Safe mode request after a certain amount of time
  - Software option: Soft response (Short functional reset)
  - Software option: Hard response (Long functional reset)

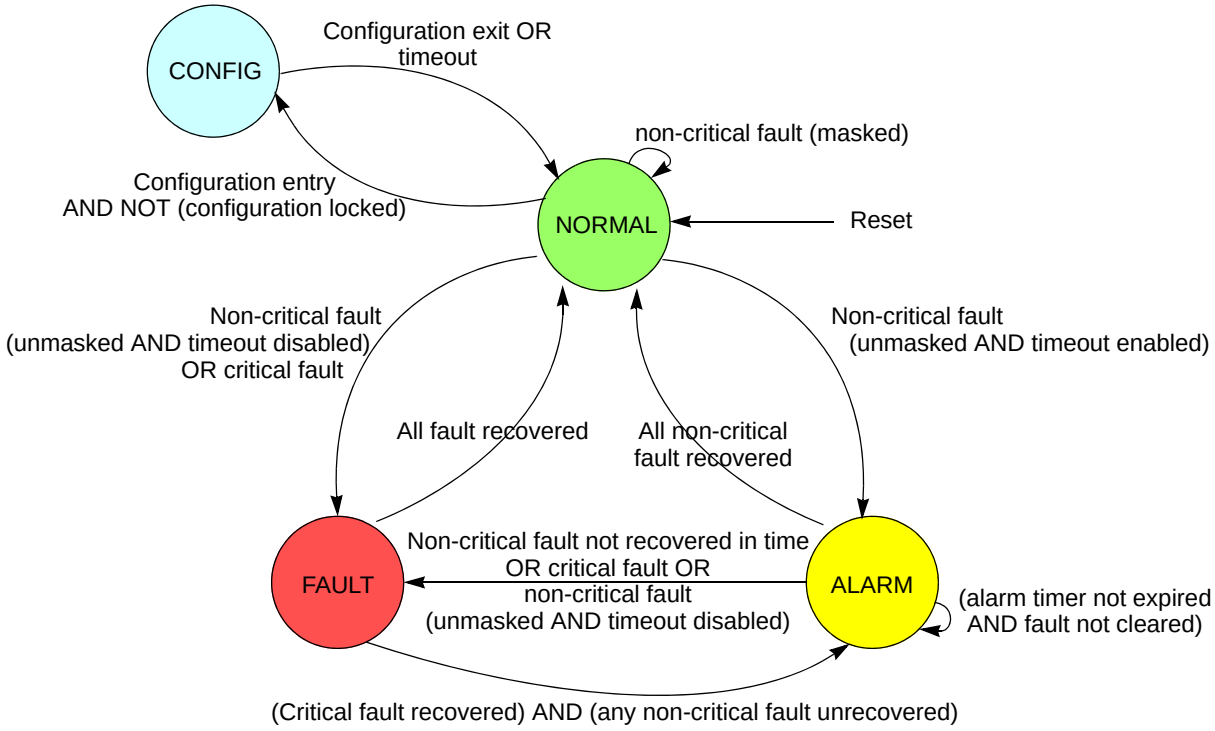


Figure 25-31. FCCU state diagram

25.8.3 Definition of critical signals

The safety concept requires monitoring critical signals from the test control unit (TCU), JTAGC, NPC, SSCM, STCU, and other modules by the FCCU during normal application operation when the system is not being debugged or in any other special mode different from normal application execution mode, such as test mode. In order to identify the right set of signals that need to be monitored, this definition is used for critical signals:

A critical signal that must continuously be monitored meets at least one of these conditions:

- Any signal on the MPC5675K that can disable (in case of a fault) a function of an IP module and its replica in the same way, and detection is not possible by any other method.

- Any signal on the MPC5675K that can disable the Safety Function (without annunciation) and the detection mechanisms at the same time.

Other requirements:

- There must not be any signal on the MPC5675K that can disable the Safety Function (without annunciation) and all fault annunciation mechanisms at the same time.

25.8.4 Self-checking capabilities

The FCCU includes some features to support self-checking capabilities of the main FSM in running mode. The FSM unit is duplicated and its state (internal state and relevant outputs) is checked by two RCC*x* units (cycle accurate). These checks (per IRCOSC clock cycle) are provided by the RCCU*x* units:

- All the FSM outputs and its internal state for the redundant FSM instances are checked to detect runtime faults on the FSM outputs and its internal state.
- Parity bits (computed at byte level) on the configuration registers are checked to detect runtime faults on the configuration inputs of the FSM.
- Parity bits (computed at byte level) on the interface are used to clear the FCCU_CFS*x* and FCCU_NCFS*x* status registers,
- FCCU_F protocol state in dual-rail, time-switching and bi-stable mode.
- Common mode signals (handshake interface signals activated only in CONFIG state) congruence with the FSM state.

In case of failure, each RCCx unit provides an interrupt request.

The RCCx state is frozen in the FCCU_SCFS register.

To enable the self checking capabilities through the FAULT interface, each critical fault source must be duplicated on a couple of CF inputs.

Two separate IRCOSC clock inputs are routed to the redundant sub-modules (FSM*x*, RCC*x*, FCCU_F*x*, FAULT-if). An external (to the FCCU) monitor of the IRCOSC clocks and an external response (‘functional’ reset to the MC_RGM) is required to cover any potential fault on the IRCOSC clocks.

Self checking capabilities cover the HW response of the FCCU due to the assertion of an external fault. The register interface and the handshake modules are not covered by self-checking capabilities. Any operation executed by SW (configuration, fault recovery) must be cross-verified by redundancy checks via SW (registers read following a write/clear operation, internal FCCU state, and so on).

A typical sequence with self checking failure and related response is given in Figure 25-32.

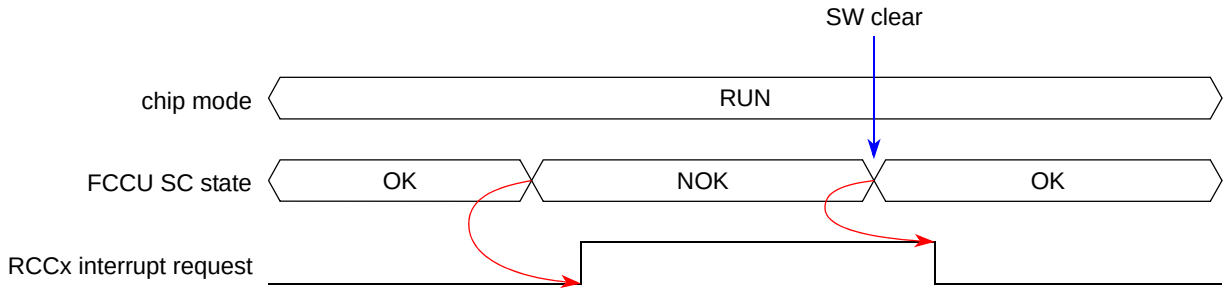


Figure 25-32. Self-checking response

25.8.5 Reset interface

The FCCU has two input resets and two output resets related to the FAULT state.

Table 25-31. Reset sources

Reset source	Support
External reset	enabled
POR	enabled
STCU	enabled
'Functional' reset	disabled
'Destructive' reset	enabled

25.8.6 Fault priority scheme and nesting

The FAULT state has a higher priority than the ALARM state in case of concurrent fault events (critical and non-critical) that occur in the NORMAL state. In case of concurrent critical faults, the fault response corresponds to the worst case (that is, a long functional reset is asserted if it has been programmed).

The ALARM to FAULT state transition occurs if a critical fault or a non-critical fault (unmasked and with timeout disabled) is asserted in the ALARM state.

Any critical fault (programmed to react with a hard or soft response) that occurs when the FCCU is already in the FAULT state causes an immediate hard or soft response (long or short functional reset).

The ALARM to NORMAL state transition occurs only if all the non-critical faults (including the faults that have been collected after the entry in the ALARM state) have been cleared (software or hardware recovery). Otherwise, the FCCU remains in the ALARM state.

The FAULT to NORMAL state transition occurs only if all the critical and non-critical faults (including the faults that have been collected after the entry in the FAULT/ALARM state) have been cleared (software or hardware recovery). Otherwise, the FCCU remains in the FAULT state (if any critical fault is still pending) or returns to the ALARM state (if any non-critical fault is still pending and the timeout has not elapsed).

In general, no fault nesting is supported except for the non-critical versus critical faults that cause an ALARM to FAULT state transition. In this case, the NCT is stopped until the FAULT state is recovered.

25.8.7 Fault recovery

The timing diagrams below describe the main use cases of the FCCU in terms of fault events and related recovery.

A typical sequence related to a critical FAULT management, given in Figure 25-33 or Figure 25-34, is described here:

- Critical fault assertion
- FCCU state transition (automatic): NORMAL → FAULT
  - Short ‘functional’ reset
  - NMI assertion
  - SAFE mode request (delayed)
- Chip mode transition: RUN → RESET → SAFE
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - Chip mode transition: SAFE → RUN

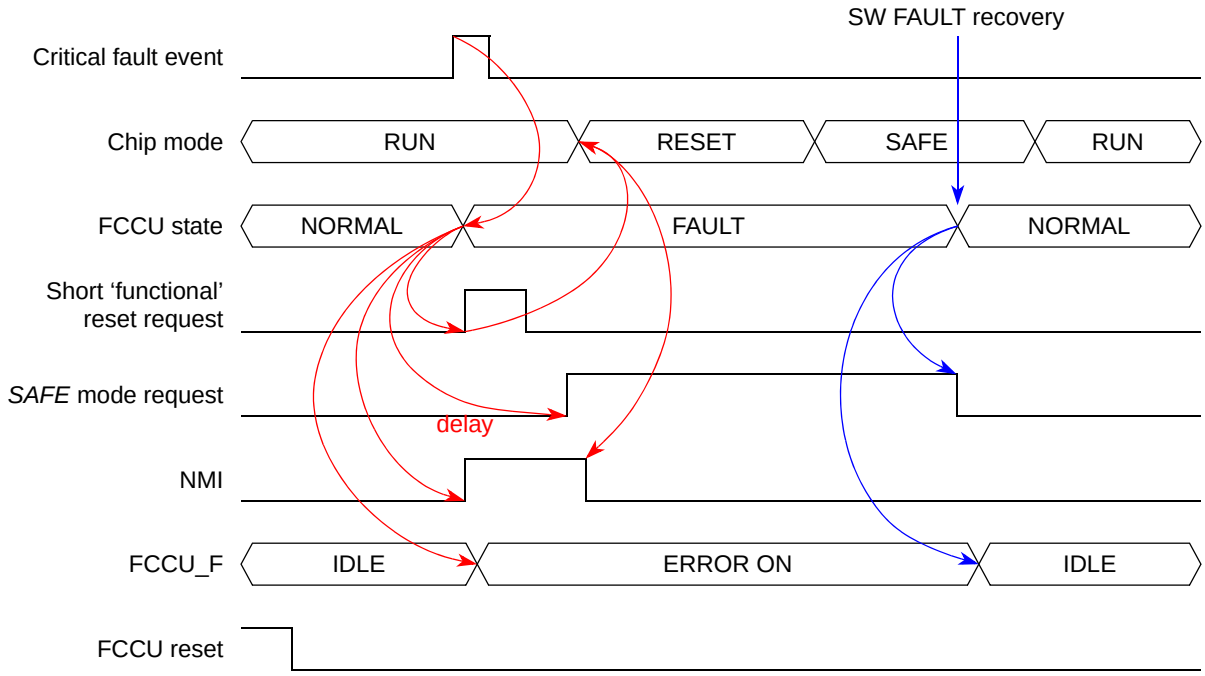


Figure 25-33. Critical FAULT recovery (a)

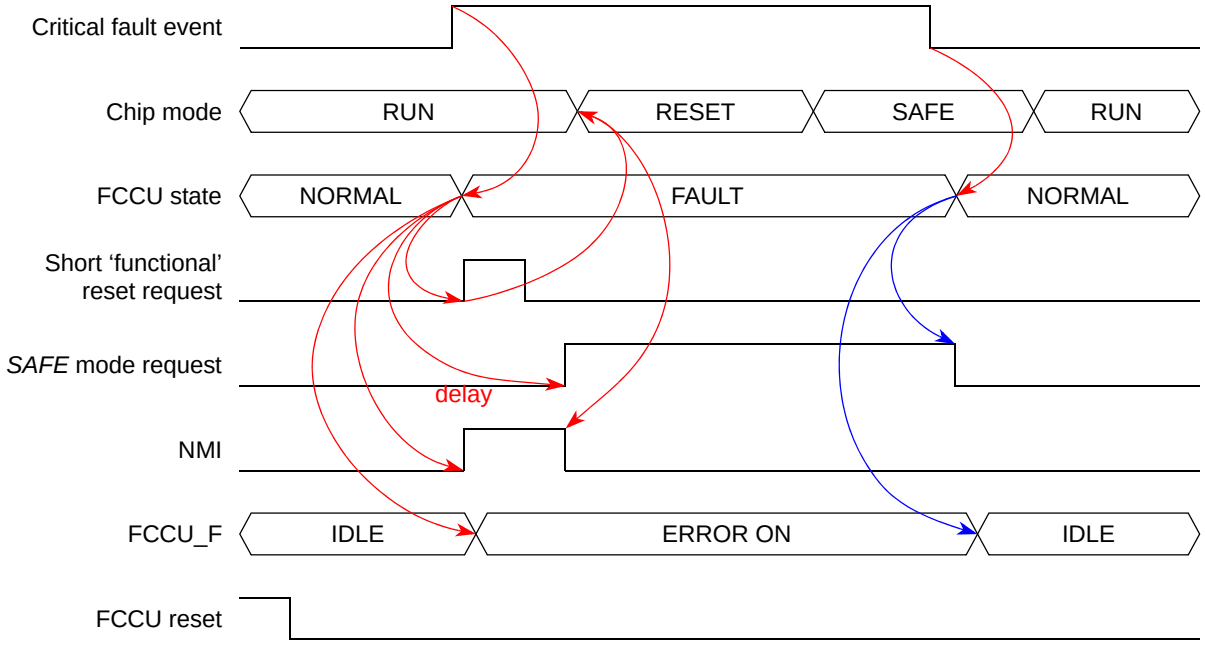


Figure 25-34. Critical FAULT recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM state), given in Figure 25-35 and Figure 25-36, is described here:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Timeout running
- Chip mode: RUN
- Alarm interrupt management
  - FAULT recovery (by SW): FCCU state transition ALARM → NORMAL

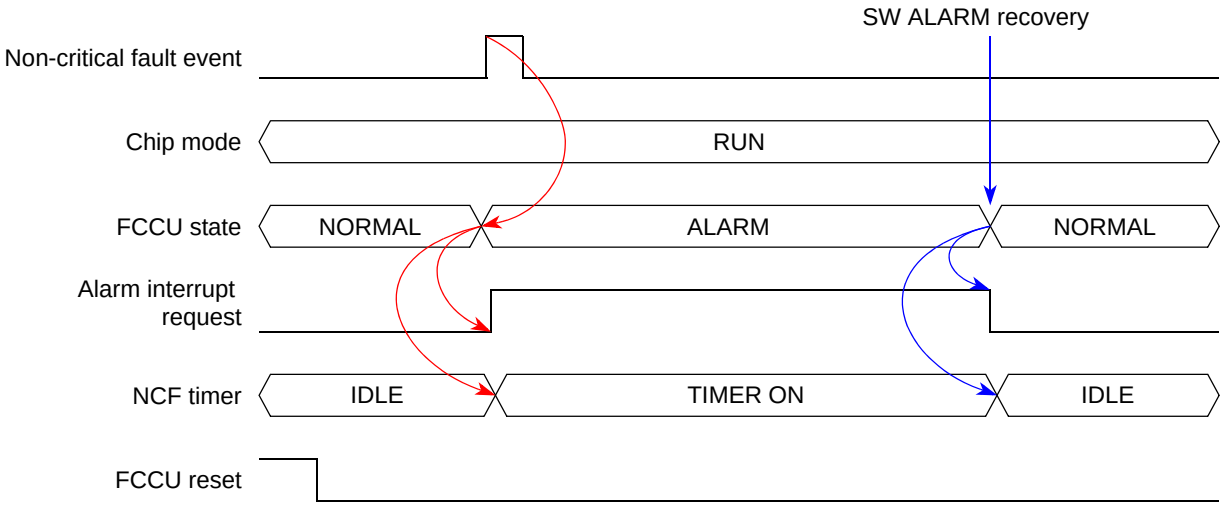


Figure 25-35. Non-critical FAULT (ALARM state) recovery (a)

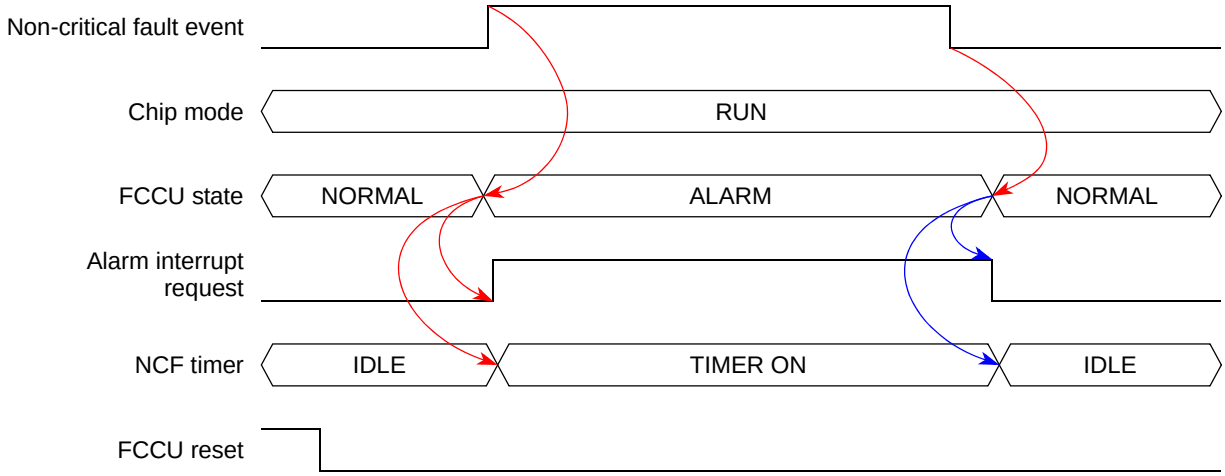


Figure 25-36. Non-critical FAULT (ALARM state) recovery (b)

A typical sequence related to a non-critical FAULT management (ALARM → FAULT state), given in Figure 25-37, is described here:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Timeout running
- FCCU state transition (following the timeout trigger): ALARM → FAULT
  - NMI assertion
  - SAFE mode request (delayed)
- Chip mode transition: RUN → SAFE
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - System state transition: SAFE → RUN

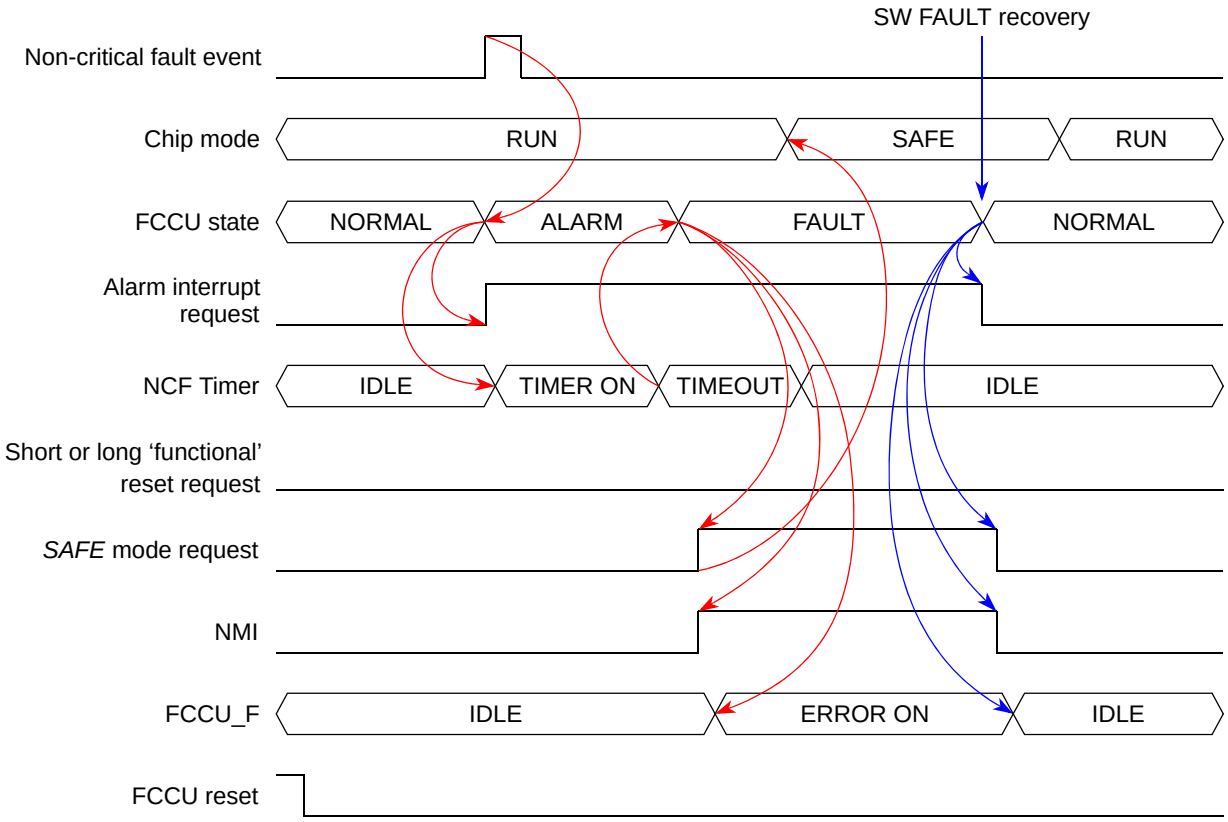


Figure 25-37. Non-critical FAULT (ALARM -> FAULT state) recovery

A typical sequence related to a critical FAULT (with nesting) management, given in Figure 25-38, is described here:

- Critical fault assertion (no short or long 'functional' reset)
- FCCU state transition (automatic): NORMAL → FAULT
  - NMI assertion
  - SAFE mode request (delayed)
- Chip mode state transition: RUN → SAFE
- Critical fault assertion
  - Short 'functional' reset
- Chip mode transition: SAFE → RESET
- NMI interrupt management
  - FAULT recovery (by SW): FCCU state transition FAULT → NORMAL
  - System state transition: SAFE → RUN

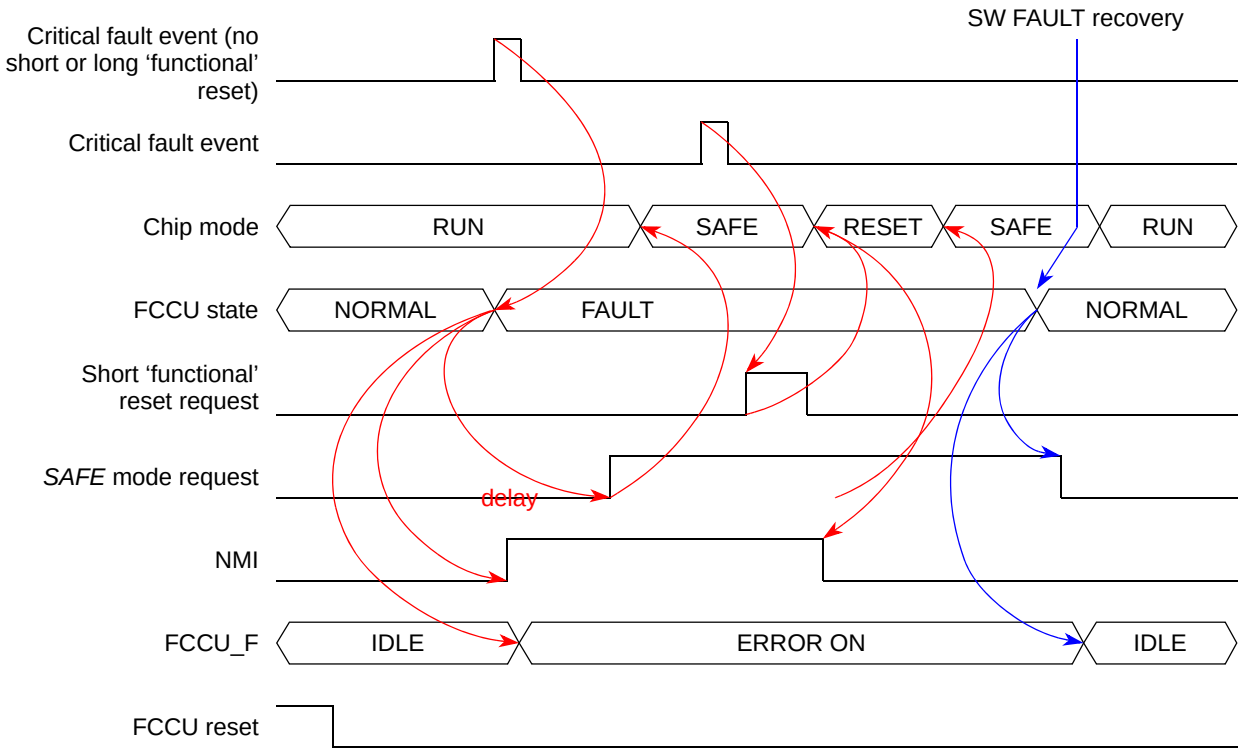


Figure 25-38. Critical FAULT (nesting) recovery

A typical sequence related to a critical FAULT (with non-critical fault nesting) management (ALARM → FAULT → ALARM state), given in Figure 25-39, where the faults are recovered sequentially, is described here:

- Non-critical fault assertion
- FCCU state transition (automatic): NORMAL → ALARM
  - Alarm interrupt request
  - Timeout running
- Critical fault assertion
- FCCU state transition (automatic): ALARM → FAULT
  - NMI assertion
  - SAFE mode request
- Chip mode transition: RUN → SAFE
- NMI interrupt management
  - FAULT (CF) recovery (by SW): FCCU state transition FAULT → ALARM, because only the critical fault has been recovered
  - Chip mode transition: SAFE → RUN
  - Timeout is still running
- Alarm interrupt management
  - FAULT (NCF) recovery (by SW): FCCU state transition ALARM → NORMAL

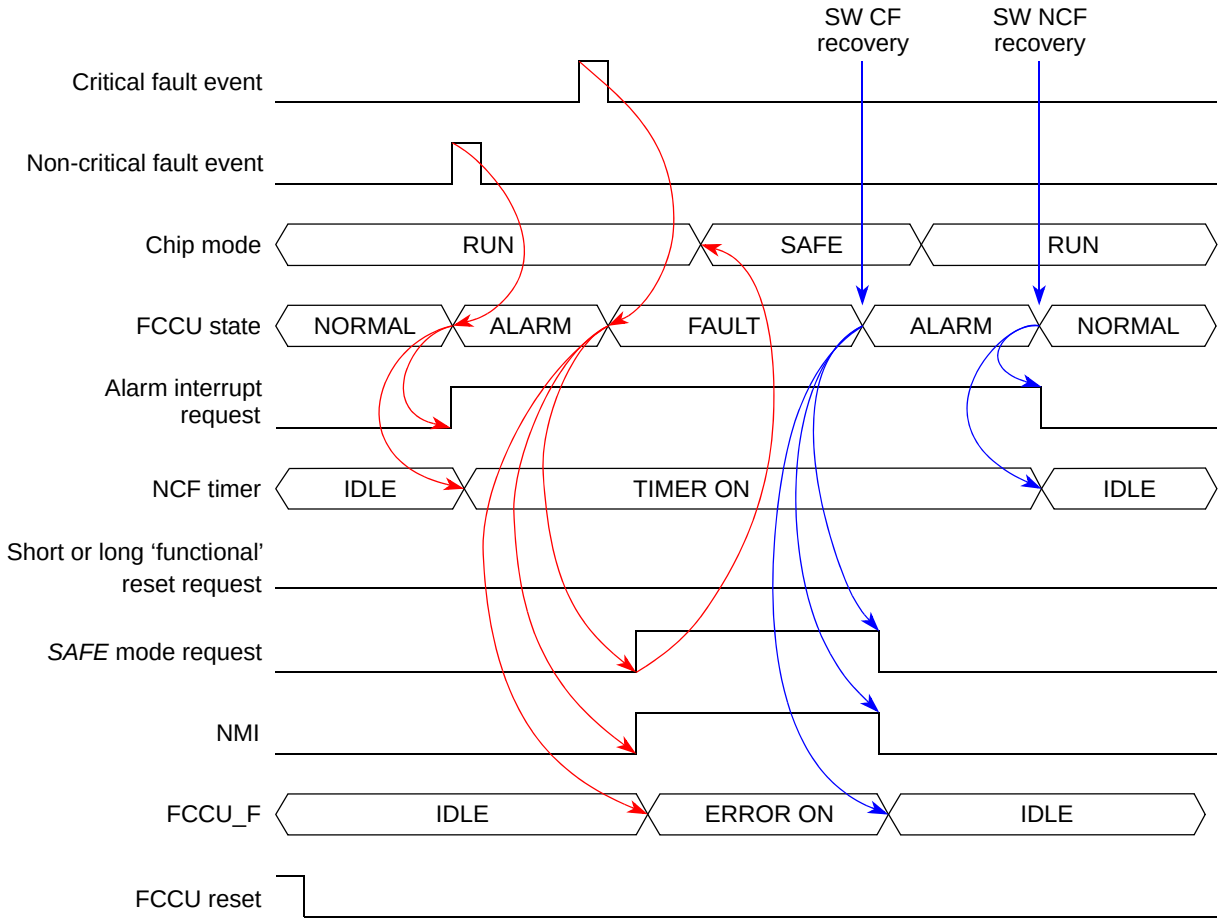


Figure 25-39. Critical FAULT (and non-critical FAULT nesting) recovery

25.8.8 FCCU fault inputs

25.8.8.1 Monitoring JTAGC and NPC

The FCCU monitors all JTAGC and NPC signals routed to both parts of the MPC5675K (JTAGC and NPC output signals routed to both parts of the SoR) that could influence the normal operation of the device. The state of the JCOMP pin determines whether the system is being debugged (JTAGC not in reset) or whether it is operating in normal application mode (JTAGC in reset). As soon as the state of one of the signals being monitored deviates from its correct state in the normal application mode, the FCCU signals a fault condition.

Asserting JCOMP to the device while Nexus is working may trigger the monitoring circuit. Therefore the user should disable Nexus tracing to safe state NPC signals, before JCOMP is asserted.

Table 25-32 defines the debug-related signals that are monitored by the FCCU.

Table 25-32. Debug signals monitored by the FCCU

Module	Signal Name	Inactive State
JTAGC	jtag_test_ctrl[7]	0
JTAGC	jtag_rst_b	1
NPC	ipp_port_en_fpm	0
NPC	ipp_port_en_rpm	0
NPC	ipp_port_en_mseo_b	0
NPC	ipp_obe_evt	0

25.8.9 Module outputs

The FCCU controls two dedicated error out pins: F[0] and F[1]. These dedicated pins are not controlled by the SIUL. The state of the output pins is readable by application software via the IPS interface of the FCCU. During power-on reset, these pins are set to high impedance.

An internal functional or destructive reset does not affect the state of the FCCU[0] and [1] pads.

The FCCU generates IRQ signals listed in Table 32-10, and reset signals listed in Chapter 46, Reset Generation Module (MC_RGM).

25.8.10 WKUP/NMI interface

The NMI signal internally generated by FCCU is masked when:

- FCCU asynchronous reset
- Chip mode = RESET

and un-masked when:

- A SW change request of the chip mode is triggered

The logical scheme is given in Figure 25-40.

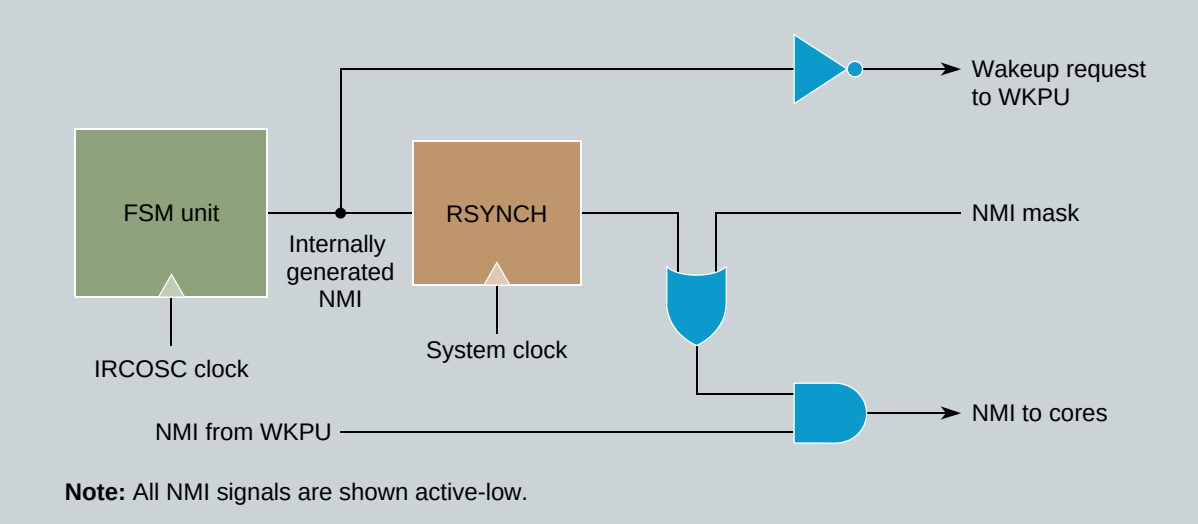


Figure 25-40. NMI/WKUP scheme

25.8.11 STCU interface

The STCU interface includes:

- A set of signals resulting from the self-checking procedure connected externally at the FCCU critical/non-critical faults. The STCU fault signals are processed by the FCCU when the chip is re-booted following the self-testing procedure. The STCU also includes a status register that stores the self-testing results (flags).
- A mask that inhibits the FCCU_F dummy signaling until the STCU self-checking procedure has been completed.
- During the self-testing procedure, depending on the STCU results, three cases are applicable:
  - The STCU completes the self-testing procedure successfully. The chip re-boots and the FCCU is responsible for providing a response. (See Case A, Figure 25-41.)
  - The STCU completes the self-testing procedure with low severity failures. The FCCU is responsible for providing the proper responses according to the fault occurred. (See Case B, Figure 25-42.)
  - The STCU completes the self-testing procedure with serious failures. The FCCU or other critical parts of the chip were not able to provide the proper response. STCU should permanently keep the chip in a RESET state. This feature is optionally programmable inside the STCU. (See Case C, Figure 25-43.)

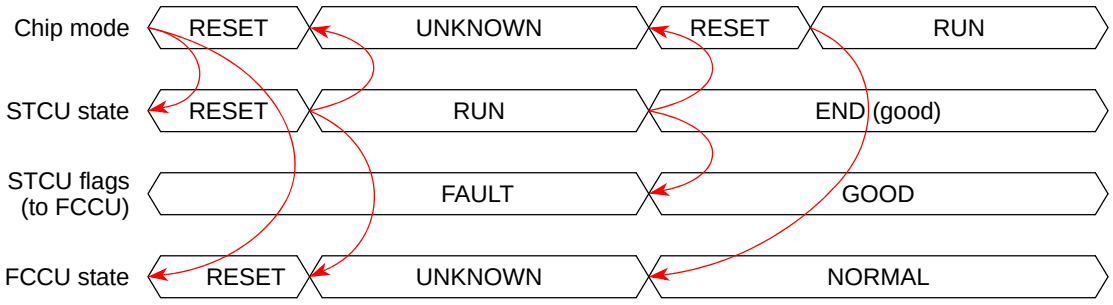


Figure 25-41. STCU-FCCU case A

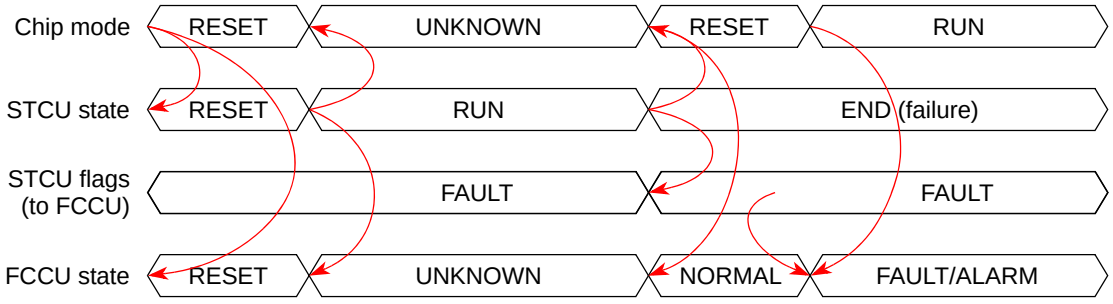


Figure 25-42. STCU-FCCU case B

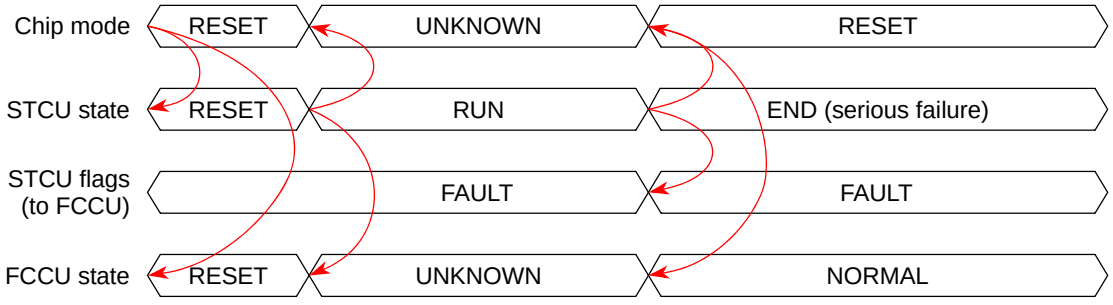


Figure 25-43. STCU-FCCU Case C

For details related to the STCU, see Chapter 50, Self-Test Control Unit (STCU).

25.8.12 NVM interface

The NVM provides the FCCU with the initial configuration information shown in Table 25-33.

Table 25-33. NVM configuration

Flash memory option bit locations	Description
BIU4[20]	Initial FCCU_CFG.CM value
BIU4[21]	Initial FCCU_CFG.SM value
BIU4[22]	Initial FCCU_CFG.PS value
BIU4[23:25]	Initial FCCU_CFG.FOM value
BIU4[26:31]	Initial FCCU_CFG.FOP value

Figure 25-44 shows the FCCU configuration sequence after a power-on, ‘destructive’, or external reset.

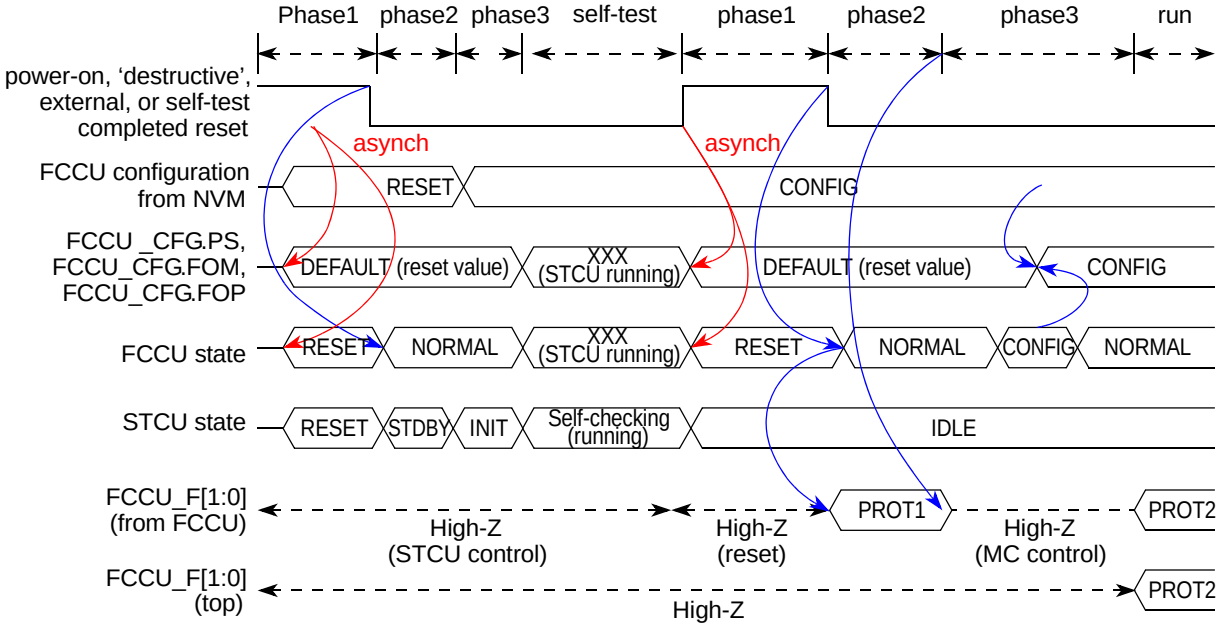


Figure 25-44. NVM interface

25.8.13 FCCU_F interface

The FCCU provides two external bidirectional signals (FCCU_F[0:1] interface). Different protocols for the FCCU_F[0:1] interface are supported, selecting the FCCU_CFG[FOM] bitfield:

- Dual rail protocol
- Time switching protocol
- Bi-stable protocol
- Test mode

The signal polarity and the frequency can be programmed by setting the FCCU_CFG[PS] and FCCU_CFG[FOP] bitfields. All the diagrams and tables are related to the default configuration selection (FCCU_CFG[FOP] = 0), switching mode (FCCU_CFG[SM] = 0), and configuration mode (FCCU_CFG[CM] = 0). In case of inverted polarity (FCCU_CFG[FOP] = 1) all the values on the FCCU_F[0:1] output pins are inverted.

Two modes can be programmed to define the FCCU_F[0:1] protocol transitions in dual-rail or time-switching mode:

- Slow switching mode: no FCCU_F[0:1] frequency violation during the FCCU state transition (NORMAL to FAULT or vice versa and CONFIG to NORMAL). The FCCU_F[0:1] protocol transition occurs after a max delay equal to the duration of the semi-period of the FCCU_F[0:1] frequency.
- Fast switching mode: The FCCU_F[0:1] protocol transition (NORMAL to FAULT or vice versa and CONFIG to NORMAL) occurs immediately. A pulse with the minimum duration

corresponding to 16 MHz / 1024 (safe clock) period can occur in fast switching mode. It implies a frequency violation of the FCCU_F[0:1] protocol.

Two modes, depending on the FCCU_CFG[CM] bit setting, can be programmed to define the FCCU_F[0:1] protocol in CONFIG state:

- Configuration labelling: the CONFIG state is marked by a specific FCCU_F[0:1] setting.
- Configuration transparency: the CONFIG and NORMAL state are equivalent.

The FCCU_F[0:1] frequency is programmable based on the RC oscillator frequency divided by a fixed prescaler (1024).

The external monitor of the FCCU_F[0:1] protocol should oversample the FCCU_F[0:1] signals in order to synchronize periodically the external clock (used by the monitor) and the safe clock detecting the edge transition of the FCCU_F[0:1] protocol in dual-rail or time-switching mode.

NOTE

After the reset phase, the initial values of the FCCU_CFG[CM], FCCU_CFG[SM], FCCU_CFG[PS], FCCU_CFG[FOP], FCCU_CFG[FOM] bitfields are set by the NVM interface (nvm_cfg[11:0]).

25.8.13.1 Dual-rail protocol

Dual-rail encoding is an alternate method for encoding bits. In contrast with classical encoding, where each signal carries a single-bit value, dual-rail encoded circuits use two wires to carry each bit. The encoding scheme is shown in Table 25-34 and the related timing diagram is shown in Figure 25-45 and Figure 25-46.

Table 25-34. Dual-rail encoding

Logical state	Dual-rail encoding (output pins fccu_1:0])	Note
Non-faulty	10	Toggling
Non-faulty	01	
Faulty	00	Toggling
Faulty	11	
Reset	High-impedance	No Toggling
Configuration	High-impedance	When FCCU_CFG[CM] = 0
	= Non-faulty	When FCCU_CFG[CM] = 1

As long as FCCU is in NORMAL or ALARM state, the output shows a “non-faulty” signal. Output pins FCCU_F[0] and FCCU_F[1] toggle between 01 and 10 with a given frequency. By default, the frequency is the RC oscillator frequency divided by 18 × 1024.

In the RESET phase, the output pins are set as high impedance.

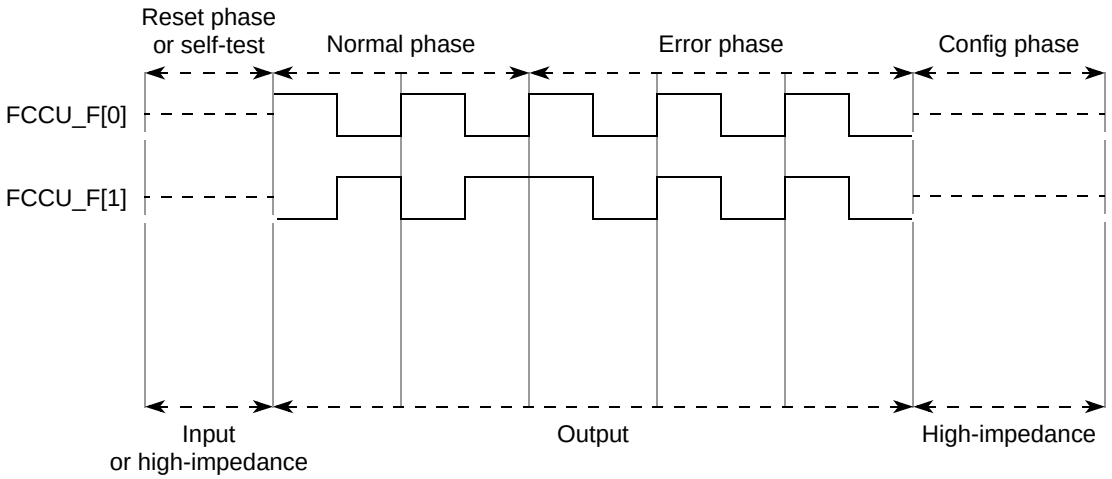


Figure 25-45. Dual-rail protocol (slow switching mode)

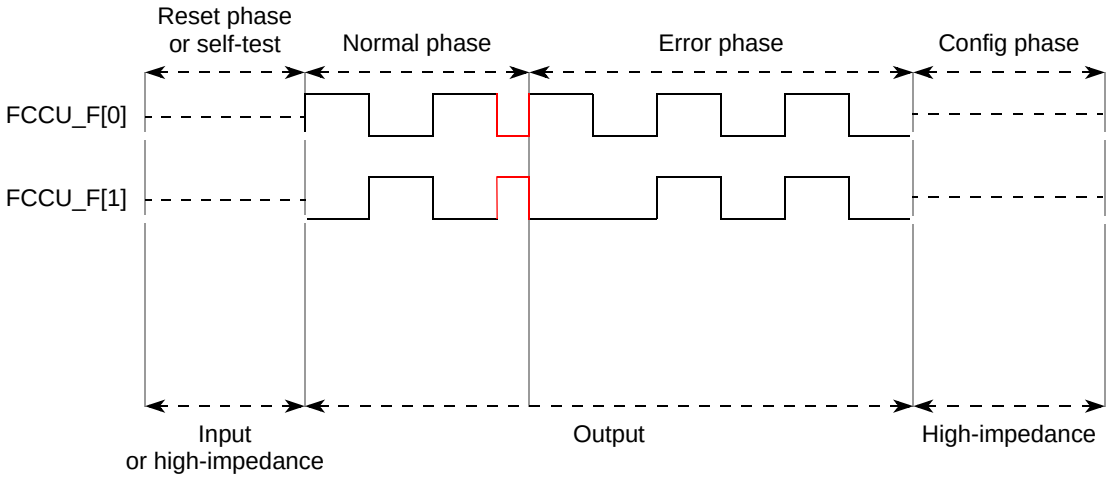


Figure 25-46. Dual-rail protocol (fast switching mode)

25.8.13.2 Time switching protocol

The encoding scheme is shown in Table 25-35 and the related timing diagram is shown in Figure 25-47.

Table 25-35. Time switching encoding

Logical state	Time switching encoding (output pins fccu_1:0])	Note
Non-faulty	10	Toggling
Non-faulty	01	
Faulty	10	No Toggling
Reset	High-impedance	No Toggling

Table 25-35. Time switching encoding

Logical state	Time switching encoding (output pins fccu_1:0])	Note
Configuration	01	When FCCU_CFG[CM] = 0
	= Non-faulty	When FCCU_CFG[CM] = 1

As long as FCCU is in NORMAL or ALARM state, outputs show a “non-faulty” signal. Output pins #0 and #1 toggle between 01 and 10 with a given frequency. By default, the frequency is the RC oscillator frequency divided by 18 × 1024.

In the FAULT state, the output pin FCCU_F[0] is set as low.

In time switching mode, the second output (FCCU_F[1]) is the inverted signal of the first output (fccu_0]). Values of 00 on the outputs indicate a fault in the error out protocol itself. This state must be considered as a critical fault, because a reliable error out indication is no longer available.

In the RESET phase the output pins are set as “high impedance”.

NOTE

Figure 25-47 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

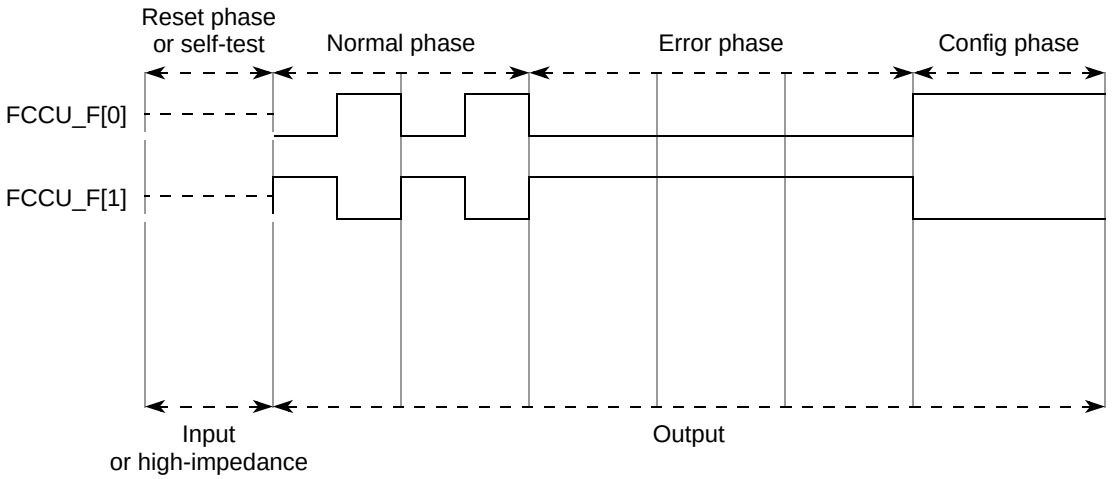


Figure 25-47. Time-switching protocol

25.8.13.3 Bi-stable protocol

The encoding scheme is shown in Table 25-36 and the related timing diagram is shown in Figure 25-48.

Table 25-36. Bi-stable encoding

Logical state	Bi-stable encoding (output pins FCCU_F[1:0])	Note
Non-faulty	01	No Toggling
Faulty	10	No Toggling
Reset	High-impedance	No Toggling
Configuration	10	When FCCU_CFG[CM] = 0
	= Non-faulty	When FCCU_CFG[CM] = 1

In the FAULT state, the faulty logical state is indicated. In NORMAL or ALARM state, “no-fault” state is indicated. In Bi-stable mode the second output (FCCU_F[1]) is the inverted signal of the first output (FCCU_F[0]).

In the RESET phase, the output pins are set as “high impedance”.

NOTE

Figure 25-48 is formatted to display the behavior in all four phases (reset, normal, error, and config), not to imply transitions between one phase to another. In particular, transition from error phase to config phase is not possible.

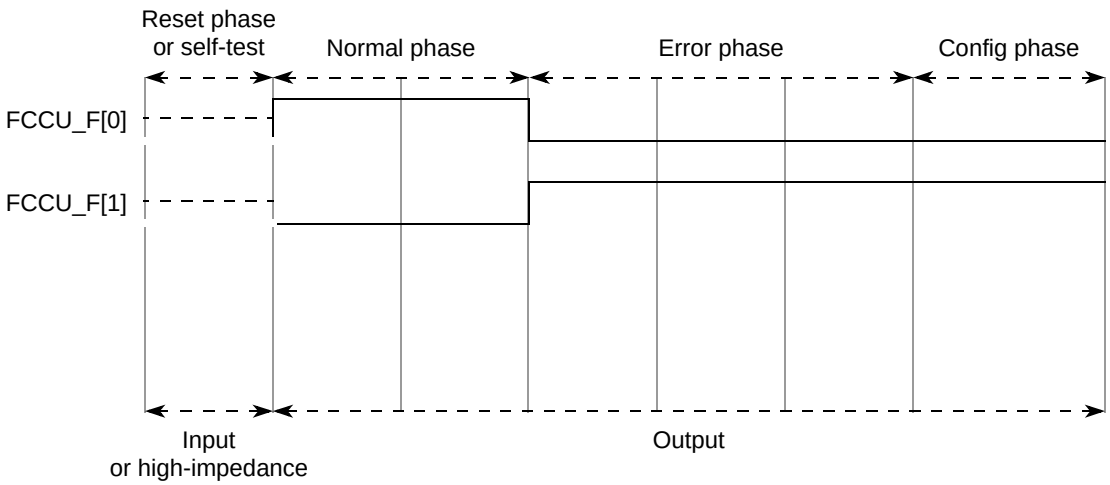


Figure 25-48. Bi-stable protocol

This page is intentionally left blank.

Chapter 26
Fast Ethernet Controller (FEC)

26.1 Introduction

26.1.1 Overview

The Fast Ethernet Controller (FEC) is an Ethernet media access controller (MAC) designed to support both 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface, which uses a subset of the MII signals.

The descriptor controller is a RISC-based controller that provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High-level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine is for the transfer of FEC data only, and is not related to the eDMA controller described in Chapter 19, Enhanced Direct Memory Access (eDMA).

The RAM FIFO is the focal point of all data flow in the fast Ethernet controller and is divided into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

MPC5675K uses a 36 x 128 array for the FIFO RAM (about 576 bytes). Of that, the default size of the Tx FIFO (as determined by the FRSR[R_FSTART] field) is 64 of the available 128 entries or 288 bytes, but this is programmable via the FRSR register.

The user controls the FEC by writing, through the slave interface module, into control registers located in each block. The CSR (control and status register) block provides global control (for example, Ethernet reset and enable) and interrupt handling registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the MDC (management data clock) and MDIO (management data input/output) lines of the MII interface.

The DMA block provides multiple channels allowing transmit data, transmit descriptor, receive data, and receive descriptor accesses to all run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The message information block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters.

26.1.2 Block diagram

The block diagram of the FEC is shown below. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE® 802.3 standards.

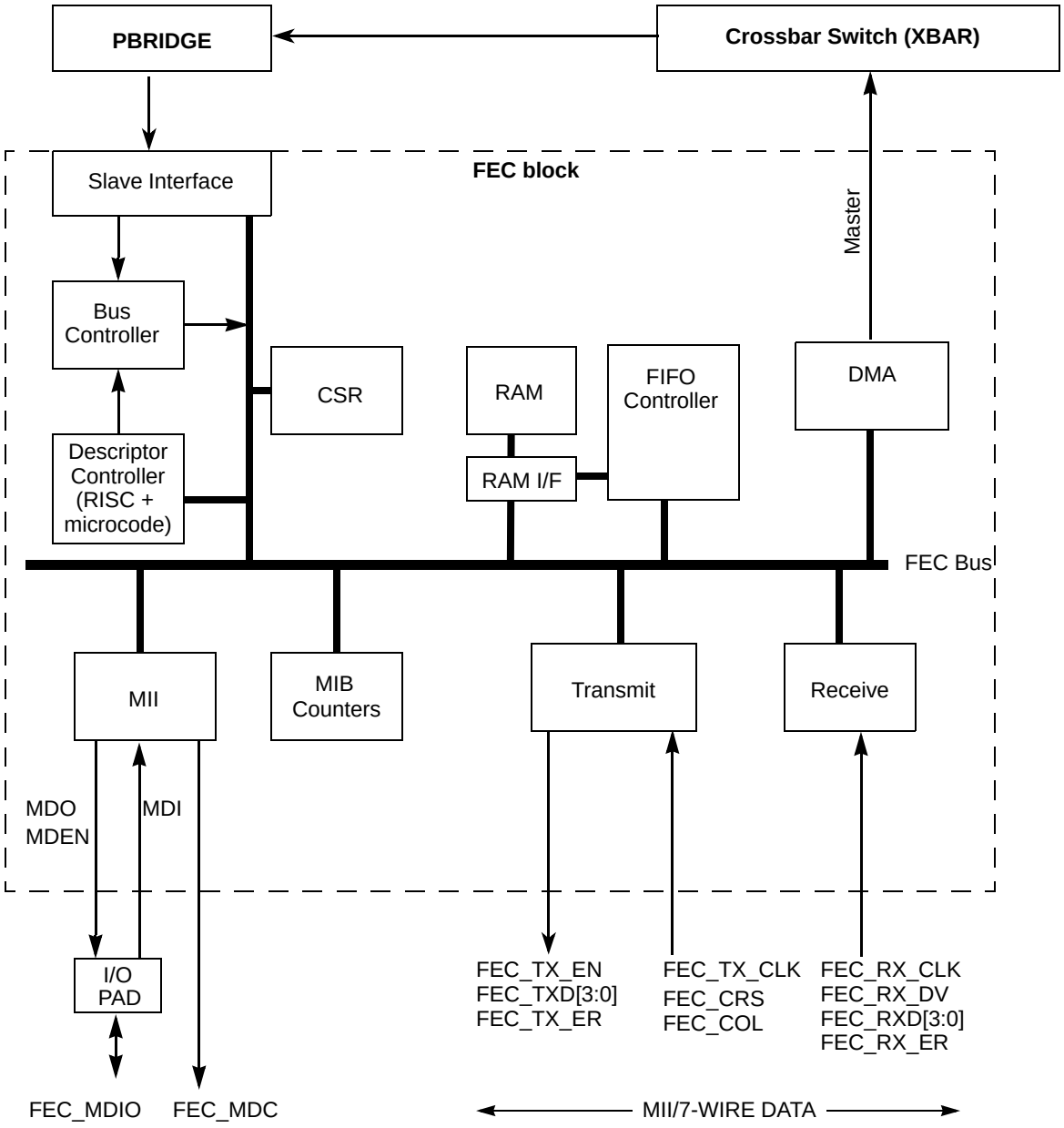


Figure 26-1. FEC block diagram

26.1.3 Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 10-Mbps 7-wire interface (industry standard)
- Built-in FIFO and DMA controller

  - 36 x 128 array FIFO RAM (about 576 bytes)
- IEEE 802.3 MAC (compliant with IEEE 802.3 1998 edition)
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- IEEE 802.3 full duplex flow control
- Support for full-duplex operation (200 Mbps throughput) with a system clock rate of 100 MHz using the external FEC_TX_CLK or FEC_RX_CLK
- Support for half-duplex operation (100 Mbps throughput) with a system clock rate of 50 MHz using the external FEC_TX_CLK or FEC_RX_CLK
- Retransmission from transmit FIFO following a collision (no system bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no system bus utilization)
- Address recognition
  - Frames with broadcast address may be always accepted or always rejected
  - Exact match for single 48-bit individual (unicast) address
  - Hash (64-bit hash) check of individual (unicast) addresses
  - Hash (64-bit hash) check of group (multicast) addresses
  - Promiscuous mode
- RMON and IEEE statistics
- Interrupts for network activity and error conditions

26.2 Modes of operation

The primary operational modes are described in this section.

26.2.1 Full and half duplex operation

Full duplex mode is intended for use on point-to-point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Selection of the duplex mode is controlled by TCR[FDEN].

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE] and TCR[TFC_PAUSE] bits, the RCR[FCE] bit, and Section 26.4.10, Full duplex flow control, for more details.

Throughputs of 200 Mbps in full duplex operations and 100 Mbps in half-duplex operations can be attained.

26.2.2 Interface options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 26.4.5, Network interface options.

26.2.2.1 10 Mbps and 100 Mbps MII interface

MII is the media-independent interface defined by the IEEE 802.3 standard for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by asserting RCR[MII_MODE].

The speed of operation is determined by the FEC_TX_CLK and FEC_RX_CLK signals, which are driven by the external transceiver. The transceiver can auto-negotiate the speed, or it can be controlled by software via the serial management interface (FEC_MDC/FEC_MDIO signals) to the transceiver. Refer to the MMFR and MSCR register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.

26.2.2.2 10 Mbps 7-wire interface operation

The FEC supports a 7-wire interface as used by many 10 Mbps ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is deasserted, the MII mode is disabled and the 10 Mbps, 7-wire mode is enabled.

26.2.3 Address recognition options

The address options supported are:

- Promiscuous
- Broadcast reject
- Individual address (hash or exact match)
- Multicast hash match

Address recognition options are discussed in detail in Section 26.4.8, Ethernet address recognition.

26.2.4 Internal loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in Section 26.4.13, Internal and external loopback.

26.2.5 Debug mode

When the MCU is in debug mode, the FEC behavior is unaffected and remains dictated by the mode of the FEC.

26.3 Programming model

This section gives an overview of the registers, followed by a description of the buffers.

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control and to extract global status information. The descriptors are used to pass data buffers and related buffer information between the hardware and software.

26.3.1 Top level module memory map

The FEC implementation requires a 1 KB memory map space. This is divided into two sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. Table 26-1 defines the top level memory map. All accesses to and from the FEC memory map must be via 32-bit accesses. Accesses other than 32-bit are not supported.

Table 26-1. FEC module memory map

Address	Function
FFF4_C000 (Base Address) – FFF4_C1FF	Control/status registers
FFF4_C200 – FFF4_C3FF	MIB block counters

26.3.2 Detailed memory map (control/status registers)

Table 26-2 shows the FEC register memory map with each register address, name, and a brief description. The base address of the FEC registers is 0xFFF4_C000.

NOTE

Some memory locations are not documented. The actual FEC memory map is from 0xFFF4_C000 to 0xFFF4_C5FF. Also, some bits in otherwise documented registers are not documented. These memory locations and bits are not needed for the FEC software driver. They are used mainly by the FEC subblocks for the FEC operation and happen to be visible through the slave interface.

Errant writes to these locations can corrupt FEC operation. Because the FEC is a system bus master, errant writes also can result in the corruption of any memory-mapped location in the system. However, even errant writes to documented FEC memory locations can cause the same corruption.

Table 26-2. FEC register memory map

Offset from FEC_BASE (0xFFF4_C000)	Register	Access[1, 2]	Reset Value[3]	Location
0x0000	FEC_ID—FEC Identification Register	R	0x0000_0600	on page 795
0x0004	EIR—Interrupt Event Register	R/W	0x0000_0000	on page 795
0x0008	EIMR—Interrupt Mask Register	R/W	0x0000_0000	on page 797
0x000C–0x000F	Reserved			
0x0010	RDAR—Receive Descriptor Active Register	R/W	0x0000_0000	on page 799
0x0014	TDAR—Transmit Descriptor Active Register	R/W	0x0000_0000	on page 799
0x0018–0x0023	Reserved			
0x0024	ECR—Ethernet Control Register	R/W	0xF000_0000	on page 800

Table 26-2. FEC register memory map (continued)

Offset from FEC_BASE (0xFFF4_C000)	Register	Access[1, 2]	Reset Value[3]	Location
0x0028–0x003F	Reserved			
0x0040	MMFR—MII Management Frame Register	R/W	0xUUUU_UUUU[4]	on page 801
0x0044	MSCR—MII Speed Control Register	R/W	0x0000_0000	on page 802
0x0048–0x0063	Reserved			
0x0064	MIBC—MIB Control/Status Register	R/W	0xC000_0000	on page 804
0x0068–0x0083	Reserved			
0x0084	RCR—Receive Control Register	R/W	0x05EE_0001	on page 804
0x0088	R_HASH—Receive Hash Register	R/W	0x0000_0000	on page 805
0x008C–0x00C3	Reserved			
0x00C4	TCR—Transmit Control Register	R/W	0x0000_0000	on page 806
0x00C8–0x00E3	Reserved			
0x00E4	PALR—MAC Address Low Register	R/W	0xUUUU_UUUU[4]	on page 807
0x00E8	PAUR—MAC Address Upper Register + Type Field Register	R/W	0xUUUU_8808[4]	on page 808
0x00EC	OPD—Opcode + Pause Duration Fields Register	R/W	0x0001_UUUU[4]	on page 808
0x00F0–0x0117	Reserved			
0x0118	IAUR—Descriptor Individual Upper Address Register	R/W	0xUUUU_UUUU[4]	on page 809
0x011C	IALR—Descriptor Individual Lower Address Register	R/W	0xUUUU_UUUU[4]	on page 810
0x0120	GAUR—Descriptor Group Upper Address Register	R/W	0xUUUU_UUUU[4]	on page 810
0x0124	GALR—Descriptor Group Lower Address Register	R/W	0xUUUU_UUUU[4]	on page 811
0x0128–0x013F	Reserved			
0x0140	FIFO_ID— FIFO Identification Register	R	0x0000_0400	on page 812
0x0144	TFWR—Transmit FIFO Watermark Register	R/W	0x0000_0000	on page 812
0x0148–0x014B	Reserved			
0x014C	FRBR—FIFO Receive Bound Register	R	0x0000_0600	on page 813
0x0150	FRSR—FIFO Receive FIFO Start Registers	R	0x0000_0500	on page 814

Table 26-2. FEC register memory map (continued)

Offset from FEC_BASE (0xFFF4_C000)	Register	Access[1, 2]	Reset Value[3]	Location
0x0154–0x017F	Reserved			
0x0180	ERDSR—Pointer to Receive Descriptor Ring Register	R/W	0xUUUU_UUUU[4]	on page 814
0x0184	ETDSR—Transmit Buffer Descriptor Ring Start Register	R/W	0xUUUU_UUUU[4]	on page 815
0x0188	EMRBR—Receive Buffer Size Register	R/W	0xUUUU_UUUU[4]	on page 816
0x018C–0x01FF	Reserved			
0x0200–0x02E0	MIB Block Counters Memory Map (see Table 26-3)			
0x02E4–0x3FFF	Reserved			

1 All accesses to and from the FEC memory map must be via 32-bit accesses. Accesses other than 32-bit are not supported.

2 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

3 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

4 One or more bits (indicated by “U”) are of indeterminate value at Reset. See register for more information.

26.3.3 MIB Block Counters Memory Map

Table 26-3 defines the MIB Counters memory map, which defines the locations in the MIB RAM space where hardware-maintained counters reside. These fall in the 0xFFF4_C200 – 0xFFF4_C3FF address offset range. The counters are divided into two groups.

- RMON counters are included, which cover the Ethernet Statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet Statistics group, a counter is included to count truncated frames as the FEC only supports frame lengths up to a maximum of 2047 bytes. The RMON counters are implemented independently for transmit and receive to ensure accurate network statistics when operating in full duplex mode.
- IEEE counters are included, which support the Mandatory and Recommended counter packages defined in section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE Basic Package objects are supported by the FEC but do not require counters in the MIB block. In addition, some of the recommended package objects that are supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 26-3. MIB Counters Memory Map

Offset from FEC_BASE (0xFFF4_C000)[1]	Mnemonic	Description
0x0200	RMON_T_DROP	Count of frames not counted correctly
0x0204	RMON_T_PACKETS	RMON Tx packet count
0x0208	RMON_T_BC_PKT	RMON Tx Broadcast Packets
0x020C	RMON_T_MC_PKT	RMON Tx Multicast Packets
0x0210	RMON_T_CRC_ALIGN	RMON Tx Packets w CRC/Align error
0x0214	RMON_T_UNDERSIZE	RMON Tx Packets < 64 bytes, good CRC
0x0218	RMON_T_OVERSIZE	RMON Tx Packets > MAX_FL bytes, good CRC
0x021C	RMON_T_FRAG	RMON Tx Packets < 64 bytes, bad CRC
0x0220	RMON_T_JAB	RMON Tx Packets > MAX_FL bytes, bad CRC
0x0224	RMON_T_COL	RMON Tx collision count
0x0228	RMON_T_P64	RMON Tx 64 byte packets
0x022C	RMON_T_P65TO127	RMON Tx 65 to 127 byte packets
0x0230	RMON_T_P128TO255	RMON Tx 128 to 255 byte packets
0x0234	RMON_T_P256TO511	RMON Tx 256 to 511 byte packets
0x0238	RMON_T_P512TO1023	RMON Tx 512 to 1023 byte packets
0x023C	RMON_T_P1024TO2047	RMON Tx 1024 to 2047 byte packets
0x0240	RMON_T_P_GTE2048	RMON Tx packets w > 2048 bytes
0x0244	RMON_T_OCTETS	RMON Tx Octets
0x0248	IEEE_T_DROP	Count of frames not counted correctly
0x024C	IEEE_T_FRAME_OK	Frames Transmitted OK
0x0250	IEEE_T_1COL	Frames Transmitted with Single Collision
0x0254	IEEE_T_MCOL	Frames Transmitted with Multiple Collisions
0x0258	IEEE_T_DEF	Frames Transmitted after Deferral Delay
0x025C	IEEE_T_LCOL	Frames Transmitted with Late Collision
0x0260	IEEE_T_EXCOL	Frames Transmitted with Excessive Collisions
0x0264	IEEE_T_MACERR	Frames Transmitted with Tx FIFO Underrun
0x0268	IEEE_T_CSERR	Frames Transmitted with Carrier Sense Error
0x026C	IEEE_T_SQE	Frames Transmitted with SQE Error
0x0270	IEEE_T_FDXFC	Flow Control Pause frames transmitted
0x0274	IEEE_T_OCTETS_OK	Octet count for Frames Transmitted w/o Error
0x0278–0x027F	—	Reserved

Table 26-3. MIB Counters Memory Map (continued)

Offset from FEC_BASE (0xFFF4_C000)[1]	Mnemonic	Description
0x0280	RMON_R_DROP	Count of frames not counted correctly
0x0284	RMON_R_PACKETS	RMON Rx packet count
0x0288	RMON_R_BC_PKT	RMON Rx Broadcast Packets
0x028C	RMON_R_MC_PKT	RMON Rx Multicast Packets
0x0290	RMON_R_CRC_ALIGN	RMON Rx Packets w CRC/Align error
0x0294	RMON_R_UNDERSIZE	RMON Rx Packets < 64 bytes, good CRC
0x0298	RMON_R_OVERSIZE	RMON Rx Packets > MAX_FL bytes, good CRC
0x029C	RMON_R_FRAG	RMON Rx Packets < 64 bytes, bad CRC
0x02A0	RMON_R_JAB	RMON Rx Packets > MAX_FL bytes, bad CRC
0x02A4–0x02A7	—	Reserved
0x02A8	RMON_R_P64	RMON Rx 64 byte packets
0x02AC	RMON_R_P65TO127	RMON Rx 65 to 127 byte packets
0x02B0	RMON_R_P128TO255	RMON Rx 128 to 255 byte packets
0x02B4	RMON_R_P256TO511	RMON Rx 256 to 511 byte packets
0x02B8	RMON_R_P512TO1023	RMON Rx 512 to 1023 byte packets
0x02BC	RMON_R_P1024TO2047	RMON Rx 1024 to 2047 byte packets
0x02C0	RMON_R_P_GTE2048	RMON Rx packets w > 2048 bytes
0x02C4	RMON_R_OCTETS	RMON Rx Octets
0x02C8	IEEE_R_DROP	Count of frames not counted correctly
0x02CC	IEEE_R_FRAME_OK	Frames Received OK
0x02D0	IEEE_R_CRC	Frames Received with CRC Error
0x02D4	IEEE_R_ALIGN	Frames Received with Alignment Error
0x02D8	IEEE_R_MACERR	Receive Fifo Overflow count
0x02DC	IEEE_R_FDXFC	Flow Control Pause frames received
0x02E0	IEEE_R_OCTETS_OK	Octet count for Frames Received w/o Error
0x02E4–0x3FFF	—	Reserved

1 All accesses to and from the FEC memory map must be via 32-bit accesses. Accesses other than 32-bit are not supported.

26.3.4 Registers

FEC registers are described in Section 26.3.4.23, FIFO Receive Start Register (FRSR), through Section 26.3.4.26, Receive Buffer Size Register (EMRBR).

26.3.4.1 Ethernet Identification Register (FEC_ID)

The Ethernet Identification Register (FEC_ID) identifies the FEC block and revision.

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEC_ID															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	DMA	FIFO	SMII	FEC_REV							
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 26-2. Ethernet Identification Register (FEC_ID)

Table 26-4. FEC_ID field descriptions

Field	Description
FEC_ID	Value identifying the FEC (10/100 Ethernet MAC).
DMA	DMA function included in the FEC. 0 FEC does not include DMA. 1 FEC includes DMA (DMA_CONTROL register contains DMA revision).
FIFO	FIFO function included in the FEC. 0 FEC does not include a FIFO. 1 FEC does include a FIFO (FIFO_ID register contains the FIFO revision).
SMII	The Ethernet PHY interface configuration. 0 MII (18 pins) or 7-wire (select MII or 7-Wire via the R_CNTRL.MII_MODE bit). 1 SMII (Serial MII) interface. This 6-pin serial MII option requires that the R_CNTRL.MII_MODE bit be set = 1.
FEC_REV	Value identifies the revision of the EMAC_10_100.

26.3.4.2 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the EIR, an interrupt is generated if the corresponding bit in the interrupt mask register (EIMR) is also set. The bit in the EIR is cleared if a one is written to that bit position; writing zero has no effect. This register is cleared on hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts that may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and RL. Interrupts resulting from internal errors are EBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts, since these errors are visible to network management via the MIB counters:

- HBERR – IEEE_T_SQE
- BABR – RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT – RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL – IEEE_T_LCOL
- COL_RETRY_LIM – IEEE_T_EXCOL
- XFIFO_UN – IEEET_MACERR

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HB ERR	BAB R	BAB T	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	w1c[1]	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. Ethernet Interrupt Event Register (EIR)

1 "w1c" signifies the bit is cleared by writing 1 to it.

Table 26-5. EIR field descriptions

Field	Description
HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR register and that the COL input was not asserted within the Heartbeat window following a transmission.
BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR[MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffers. Truncation does not occur.
GRA	Graceful stop complete. This interrupt is asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. • A graceful stop, which was initiated by setting the TCR[GTS] bit, is now complete. • A graceful stop, which was initiated by setting the TCR[TFC_PAUSE] bit, is now complete. • A graceful stop, which was initiated by the reception of a valid full duplex flow control "pause" frame, is now complete. Please see Section 26.4.10, Full duplex flow control.
TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.
RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.

Table 26-5. EIR field descriptions (continued)

Field	Description
RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
EBERR	Ethernet bus error. This bit indicates that a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, ECR[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software must ensure that the FIFO controller and DMA are also soft reset.
LC	Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
RL	Collision retry limit. This bit indicates that a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted, and transmission of the next frame begins. Can only occur in half duplex mode.
UN	Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.

26.3.4.3 Ethernet Interrupt Mask Register (EIMR)

The Ethernet Interrupt Mask Register (EIMR) controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared on a hardware reset. If the corresponding bits in both the EIR and EIMR registers are set, the interrupt is signaled to the CPU. The interrupt signal remains asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HB ERR	BAB R	BAB T	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. Interrupt Mask Register (EIMR)

Table 26-6. EIMR field descriptions

Field	Description
HBERR	Heartbeat error interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
BABR	Babbling receiver interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
BABT	Babbling transmitter interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
GRA	Graceful stop interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
TXF	Transmit frame interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
TXB	Transmit buffer interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
RXF	Receive frame interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
RXB	Receive buffer interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
MII	MII interrupt enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
EBERR	Ethernet controller bus error enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
LC	Late collision enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
RL	Collision retry limit enable 0 The interrupt source is masked. 1 The interrupt source is enabled.
UN	Transmit FIFO underrun enable 0 The interrupt source is masked. 1 The interrupt source is enabled.

26.3.4.4 Receive Descriptor Active Register (RDAR)

The Receive Descriptor Active Register (RDAR) is a command register, written by the user, that indicates that the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the empty bit set).

Whenever the register is written, the R_DES_ACTIVE bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames (if ECR[ETHER_EN] is also set). Once the FEC polls a receive descriptor whose empty bit is not set, the FEC clears R_DES_ACTIVE and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors have been placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER_EN] is cleared.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	R_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. Receive Descriptor Active Register (RDAR)

Table 26-7. RDAR field descriptions

Field	Description
R_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “empty” descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.

26.3.4.5 Transmit Descriptor Active Register (TDAR)

The Transmit Descriptor Active Register (TDAR) is a command register that should be written by the user to indicate that the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

Whenever the register is written, the X_DES_ACTIVE bit is set. This value is independent of the data actually written by the user. When set, the FEC polls the transmit descriptor ring and processes transmit frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a transmit descriptor whose ready bit is not set, the FEC clears X_DES_ACTIVE and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	X_DES_ACTIVE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. Transmit Descriptor Active Register (TDAR)

Table 26-8. TDAR field descriptions

Field	Description
X_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “ready” descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.

26.3.4.6 Ethernet Control Register (ECR)

The Ethernet Control Register (ECR) enables and disables the FEC.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ETHER_EN	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. Ethernet Control Register (ECR)

Table 26-9. ECR field descriptions

Bits	Description
ETHER_EN	When this bit is set, the FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any currently transmitted frame. The buffer descriptors for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is deasserted, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions: • ECR[RESET] is set by software, in which case ETHER_EN is cleared. • An error condition causes the EIR[EBERR] bit to set, in which case ETHER_EN is cleared.
RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately 8 system clock cycles after RESET is written with a 1.

26.3.4.7 MII Management Frame Register (MMFR)

The MII Management Frame Register (MMFR) communicates with the attached MII compatible PHY devices, providing read/write access to their MII registers. Performing a write to the MMFR causes a management frame to be sourced unless the MSCR has been programmed to 0. In the case of writing to MMFR when MSCR = 0, if the MSCR register is then written to a non-zero value, an MII frame is generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ST		OP		PA				RA				TA			
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-8. MII Management Frame Register (MMFR)

Table 26-10. MMFR field descriptions

Field	Description
ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
OP	Operation code. This field must be programmed to 10 (read) or 01 (write) to generate a valid MII management frame. A value of 11 produces “read” frame operation. A value of 00 produces “write” frame operation, but these frames are not MII compliant.

Table 26-10. MMFR field descriptions (continued)

Field	Description
PA	PHY address. This field specifies one of as many as 32 attached PHY devices.
RA	Register address. This field specifies one of as many as 32 registers within the specified PHY device.
TA	Turn around. This field must be programmed to 0b10 to generate a valid MII management frame.
DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

To perform a read or write operation on the MII management interface, the MMFR register must be written by the user. To generate a valid read or write management frame, the ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated that does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII management interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. Once the write management frame operation has completed, the MII interrupt is generated. At this time the contents of the MMFR register match the original value written.

To generate an MII management interface read frame (read a PHY register) the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a “don’t care”). Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted, and are unpredictable if read by the user. Once the read management frame operation has completed, the MII interrupt is generated. At this time, the contents of the MMFR register match the original value written, except for the DATA field, whose contents have been replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents are altered. Software should poll the EIR[MII] bit or use the EIR[MII] bit to generate an interrupt to avoid writing to the MMFR register while frame generation is in progress.

26.3.4.8 MII Speed Control Register (MSCR)

The MSCR:

- Provides control of the MII clock (FEC_MDC signal) frequency
- Allows a preamble drop on the MII management frame
- Provides observability (intended for manufacturing test) of an internal counter used in generating the FEC_MDC clock signal

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	DIS_PREAMBLE	MII_SPEED						0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. MII Speed Control Register (MSCR)

Table 26-11. MSCR field descriptions

Field	Description
DIS_PREAMBLE	Asserting this bit causes preamble (32 1's) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY devices do not require it.
MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (FEC_MDC) relative to the system clock. A value of 0 in this field "turns off" the MDC and leaves it in low voltage state. Any non-zero value results in the MDC frequency of 1/(MII_SPEED × 2) of the system clock frequency.

The MII_SPEED field must be programmed with a value to provide an MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value in order to source a read or write management frame. After the management frame is complete the MSCR register may optionally be set to zero to turn off the MDC. The MDC generated has a 50% duty cycle except when MII_SPEED is changed during operation (change takes effect following either a rising or falling edge of MDC).

If the system clock is 50 MHz, programming this register to 0x0000_0014 results in an MDC frequency of 50 MHz × 1/20 = 2.5 MHz. A table showing optimum values for MII_SPEED as a function of system clock frequency is provided in Table 26-12.

Table 26-12. Programming Examples for MSCR

System Clock Frequency	MII_SPEED bitfield	MDC frequency
50 MHz	0xA	2.5 MHz
66 MHz	0xE	2.36 MHz
80 MHz	0x10	2.5 MHz

26.3.4.9 MIB Control Register (MIBC)

The MIB Control Register (MIBC) provides control and status of the state of the MIB block. This register is accessed by user software if there is a need to disable the MIB block operation. For example, in order to clear all MIB counters in RAM the user should disable the MIB block, then clear all the MIB RAM locations, then enable the MIB block. The MIB_DISABLE bit is reset to 1. See Table 26-3 for the locations of the MIB counters.

Address: Base + 0x0064 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIB_DISABLE	MIB_IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. MIB Control Register (MIBC)

Table 26-13. MIBC field descriptions

Field	Description
MIB_DISABLE	A read/write control bit. If set, the MIB logic halts and does not update any MIB counters.
MIB_IDLE	A read-only status bit. If set, the MIB block is not currently updating any MIB counters.

26.3.4.10 Receive Control Register (RCR)

The Receive Control Register (RCR) controls the operational mode of the receive block and should be written only when ECR[ETHER_EN] = 0 (initialization time).

Address: Base + 0x0084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 26-11. Receive Control Register (RCR)

Table 26-14. RCR field descriptions

Field	Description
MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL cause the BABT interrupt to occur. Receive frames longer than MAX_FL cause a BABR interrupt and set the LG bit in the end-of-frame receive buffer descriptor. You can program the default value to 1518 or 1522.
FCE	Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = 0xFFFF_FFFF_FFFF are rejected unless the PROM bit is set. If both BC_REJ and PROM = 1, then frames with broadcast DA are accepted and the M (MISS) bit in the receive buffer descriptor is set. See Section 26.5.2, Ethernet receive buffer descriptor (RxBD).
PROM	Promiscuous mode. All frames are accepted, regardless of address matching.
MII_MODE	Media independent interface mode. Selects external interface mode. Setting this bit to 1 selects MII mode, clearing this bit selects 7-wire mode (used only for serial 10 Mbps). This bit controls the interface mode for both transmit and receive blocks.
DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the FEC_TX_CLK when LOOP is asserted. DRT must be set to 0 when asserting LOOP.

26.3.4.11 Receive Hash Register (R_HASH)

The Receive Hash Register (R_HASH) provides address recognition information from the receive block about the frame currently being received. This field is read by the device. These bits provide the device with information used in the address recognition subroutine.

Address: Base + 0x0088 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FCE_DC	MULT CAST	HASH						0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-12. Receive Hash Register (R_HASH)

Table 26-15. R_HASH field descriptions

Field	Description
FCE_DC	This is a read-only view of the FCE bit in the RCR register.
MULTCAST	Multicast. This bit is set if the current receive frame contains a multi-cast destination address (the least significant bit of the DA was set). This bit is cleared if the current receive frame does not correspond to a multi-cast address.
HASH	Corresponds to the hash value of the current receive frame's destination address. The hash value is a six-bit field extracted from the least significant portion of the CRC register.

26.3.4.12 Transmit Control Register (TCR)

The Transmit Control Register (TCR) configures the transmit block. Bits 29 and 30 should be modified only when ECR[ETHER_EN] = 0.

Address: Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	RFC_ PAUSE	TFC_ PAUSE	FD EN	HBC	GTS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-13. Transmit Control Register (TCR)

Table 26-16. TCR field descriptions

Field	Description
RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit is automatically cleared when the pause duration is complete.
TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmitting data frames after the current transmission is complete. At this time, the GRA interrupt in the EIR register is asserted. With transmission of data frames stopped, the MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. Note that if the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may still transmit a MAC Control PAUSE frame.
FDEN	Full duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should only be modified when ETHER_EN is deasserted.
HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ETHER_EN is deasserted.
GTS	Graceful transmit stop. When this bit is set, the MAC stops transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the EIR register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. Once transmission has completed, a “restart” can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS = 1, transmission stops after the collision. The frame is transmitted again once GTS is cleared. Note that there may be old frames in the transmit FIFO that are transmitted when GTS is reasserted. To avoid this situation, deassert ECR[ETHER_EN] following the GRA interrupt.

26.3.4.13 Physical Address Low Register (PALR)

The Physical Address Low Register (PALR) contains the lower 32 bits (bytes 0, 1, 2, 3) of the 48-bit MAC address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte source address field when transmitting PAUSE frames. This register is not reset and must be initialized by the user.

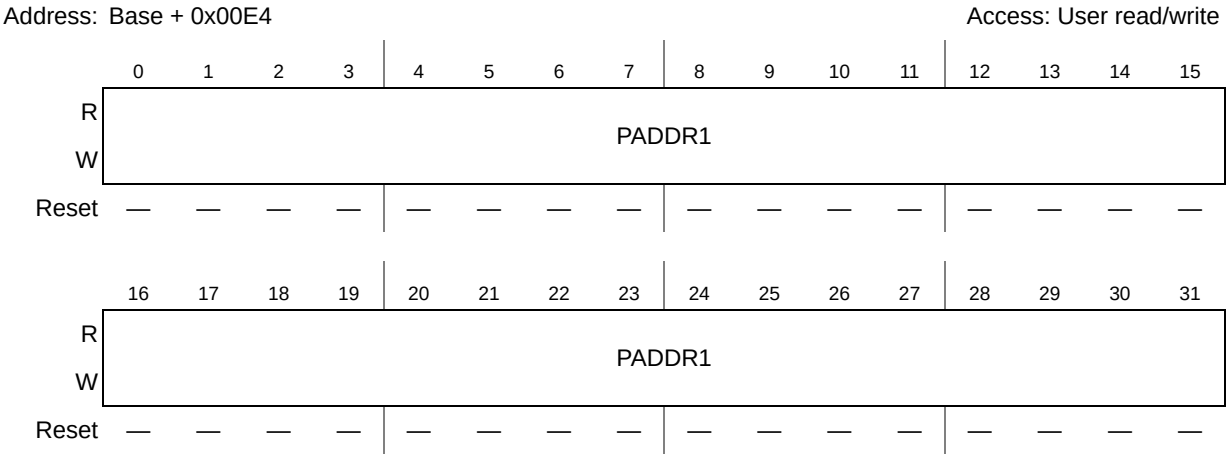


Figure 26-14. Physical Address Low Register (PALR)

Table 26-17. PALR field descriptions

Field	Description
PADDR1	Bytes 0 (bits 0:7), 1 (bits 8:15), 2 (bits 16:23) and 3 (bits 24:31) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.

26.3.4.14 Physical Address Upper Register (PAUR)

The Physical Address Upper Register (PAUR) contains the upper 16 bits (bytes 4 and 5) of the 48-bit MAC address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 16:31 of PAUR contain a constant TYPE field (0x8808) used for transmission of PAUSE frames.This register is not reset, and bits 0:15 must be initialized by the user. Refer to Section 26.4.10, Full duplex flow control, for information on using the TYPE field.

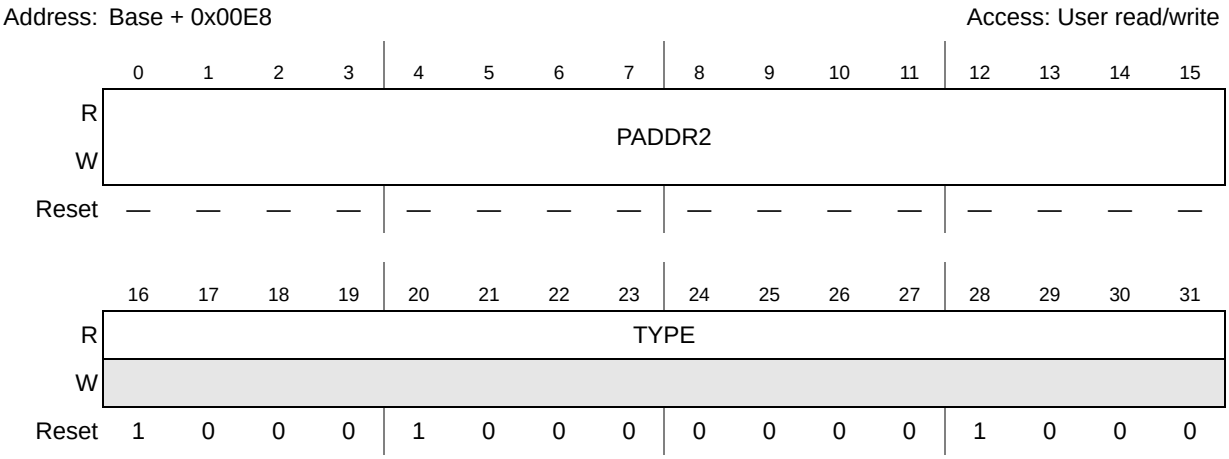


Figure 26-15. Physical Address Upper Register (PAUR)

Table 26-18. PAUR field descriptions

Field	Description
PADDR2	Bytes 4 (bits 0:7) and 5 (bits 8:15) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
TYPE	The type field is used in PAUSE frames. These bits are a constant, 0x8808.

26.3.4.15 Opcode/Pause Duration Register (OPD)

The Opcode/Pause Duration Register (OPD) is read/write accessible. This register contains the 16-bit OPCODE and 16-bit pause duration (PAUSE_DUR) fields used in transmission of a PAUSE frame. The OPCODE field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. This register is not reset and must be initialized by the user. Refer to Section 26.4.10, Full duplex flow control, for information on using the OPD register.

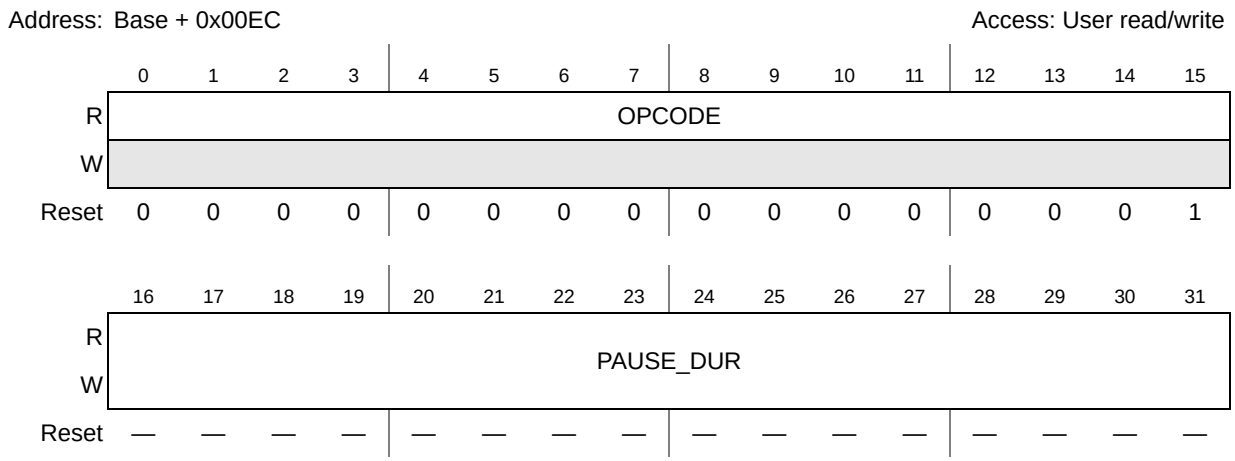


Figure 26-16. Opcode/Pause Duration Register (OPD)

Table 26-19. OPD field descriptions

Field	Description
OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
PAUSE_DUR	Pause duration field used in PAUSE frames.

26.3.4.16 Descriptor Individual Upper Address Register (IAUR)

The Descriptor Individual Upper Address Register (IAUR) contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

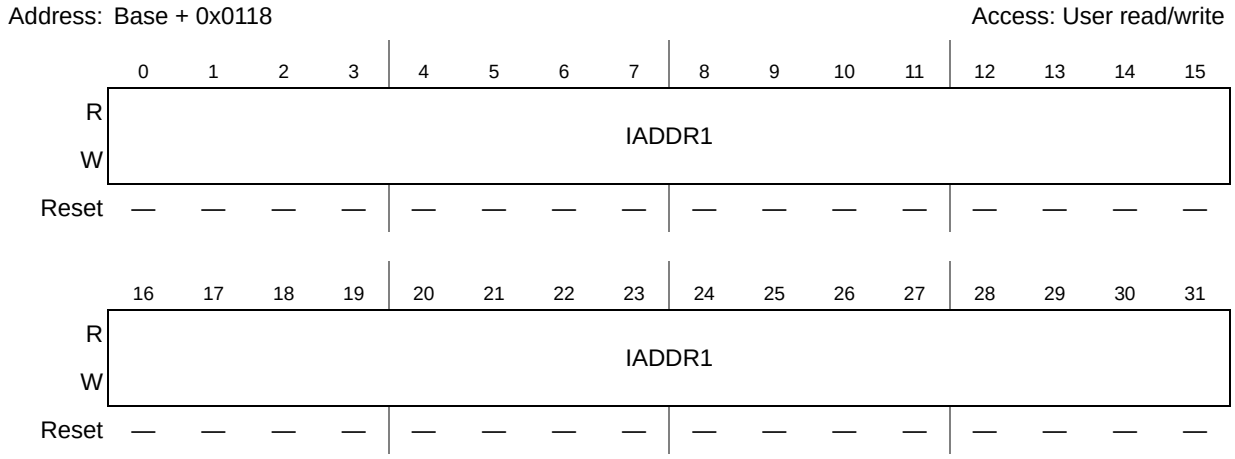


Figure 26-17. Descriptor Individual Upper Address Register (IAUR)

Table 26-20. IAUR field descriptions

Field	Descriptions
IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 0 of IADDR1 contains hash index bit 63. Bit 31 of IADDR1 contains hash index bit 32.

26.3.4.17 Descriptor Individual Lower Address Register (IALR)

The Descriptor Individual Lower Address Register (IALR) contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

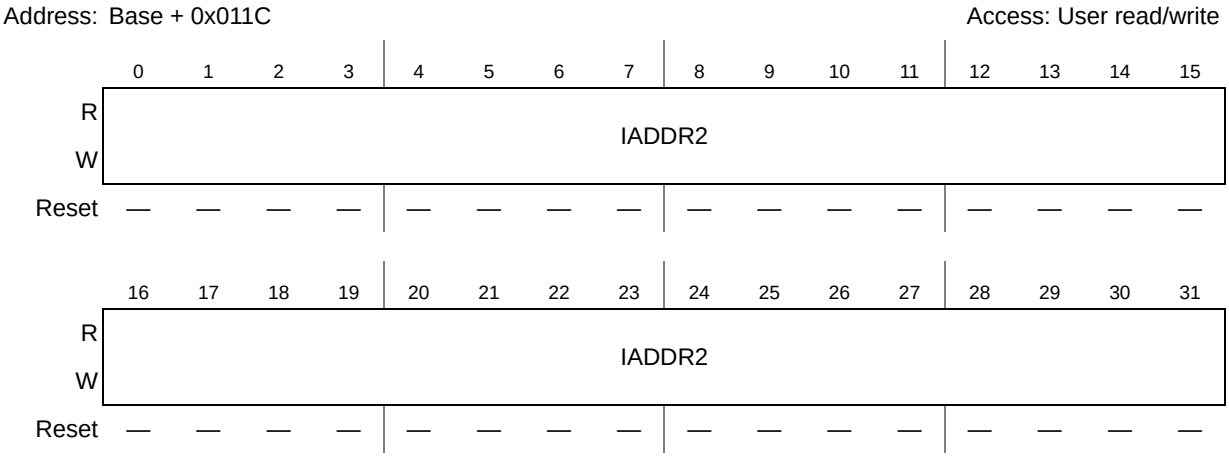


Figure 26-18. Descriptor Individual Lower Address (IALR)

Table 26-21. IALR field descriptions

Field	Description
IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 0 of IADDR2 contains hash index bit 31. Bit 31 of IADDR2 contains hash index bit 0.

26.3.4.18 Descriptor Group Upper Address Register (GAUR)

The Descriptor Group Upper Address Register (GAUR) contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

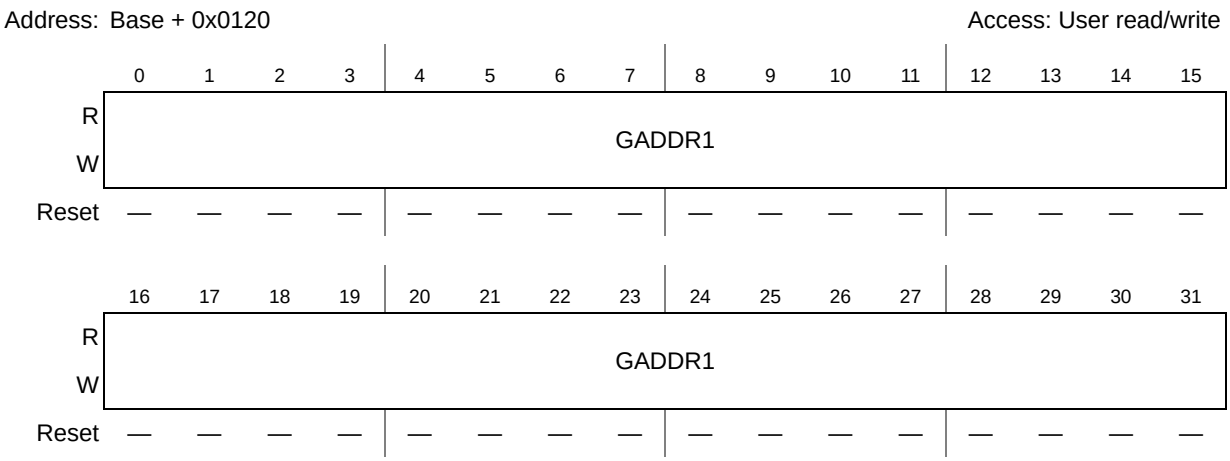


Figure 26-19. Descriptor Group Upper Address Register (GAUR)

Table 26-22. GAUR field descriptions

Field	Description
GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 0 of GADDR1 contains hash index bit 63. Bit 31 of GADDR1 contains hash index bit 32.

26.3.4.19 Descriptor Group Lower Address Register (GALR)

The Descriptor Group Lower Address Register (GALR) contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

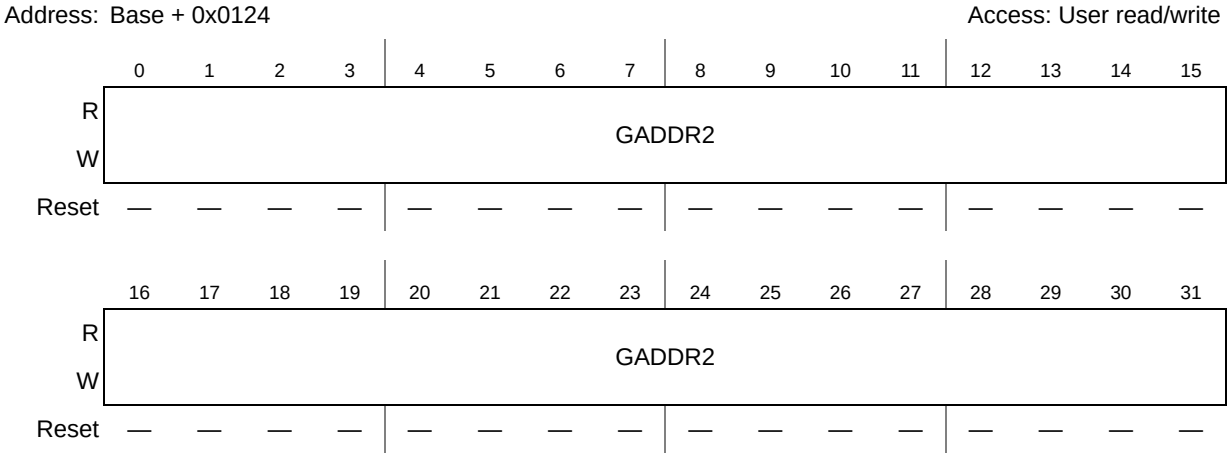


Figure 26-20. Descriptor Group Lower Address Register (GALR)

Table 26-23. GALR field descriptions

Field	Description
GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 0 of GADDR2 contains hash index bit 31. Bit 31 of GADDR2 contains hash index bit 0.

26.3.4.20 FIFO Identification Register (FIFO_ID)

The FIFO Identification Register (FIFO_ID) provides identification information about the FIFO.

Address: Base + 0x0140 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FIFO_REV								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FIFO_SIZE														0	0
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-21. FIFO Identification Register (FIFO_ID)

Table 26-24. FIFO_ID field descriptions

Field	Description
FIFO_REV	FIFO Revision
FIFO_SIZE	Size of RAM block used to implement registers, Transmit FIFO and Receive FIFO. The FIFO RAM is 36 bits wide. This field indicates the number of 36-bit words in the RAM.

26.3.4.21 Transmit FIFO Watermark Register (TFWR)

The Transmit FIFO Watermark Register (TFWR) controls the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user to minimize transmit latency (TFWR = 0x) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Address: Base + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X_WMRK	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-22. FIFO Transmit FIFO Watermark Register (TFWR)

Table 26-25. TFWR field descriptions

Field	Descriptions
0–29	Reserved, should be cleared.
X_WMRK	Number of bytes written to transmit FIFO before transmission of a frame begins. 0x 64 bytes written 10 128 bytes written 11 192 bytes written

26.3.4.22 FIFO Receive Bound Register (FRBR)

The FIFO Receive Bound Register (FRBR) allows the user to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR register, to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Address: Base + 0x014C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	1	R_BOUND								0	0
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 26-23. FIFO Receive Bound Register (FRBR)

Table 26-26. FRBR field descriptions

Field	Descriptions
R_BOUND	Read-only. Highest valid FIFO RAM address.

26.3.4.23 FIFO Receive Start Register (FRSR)

The FIFO Receive Start Register (FRSR) indicates the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR. The receive FIFO uses addresses from FRSR to FRBR inclusive.

The FRSR register is initialized by hardware at reset. FRSR only needs to be written to change the default value.

Address: Base + 0x0150 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	1	R_FSTART								0	0
W																
Reset	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Figure 26-24. FIFO Receive Start Register (FRSR)

Table 26-27. FRSR field descriptions

Field	Descriptions
R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs.

26.3.4.24 Receive Descriptor Ring Start (ERDSR)

The Receive Descriptor Ring Start (ERDSR) provides a pointer to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized by the user prior to operation.

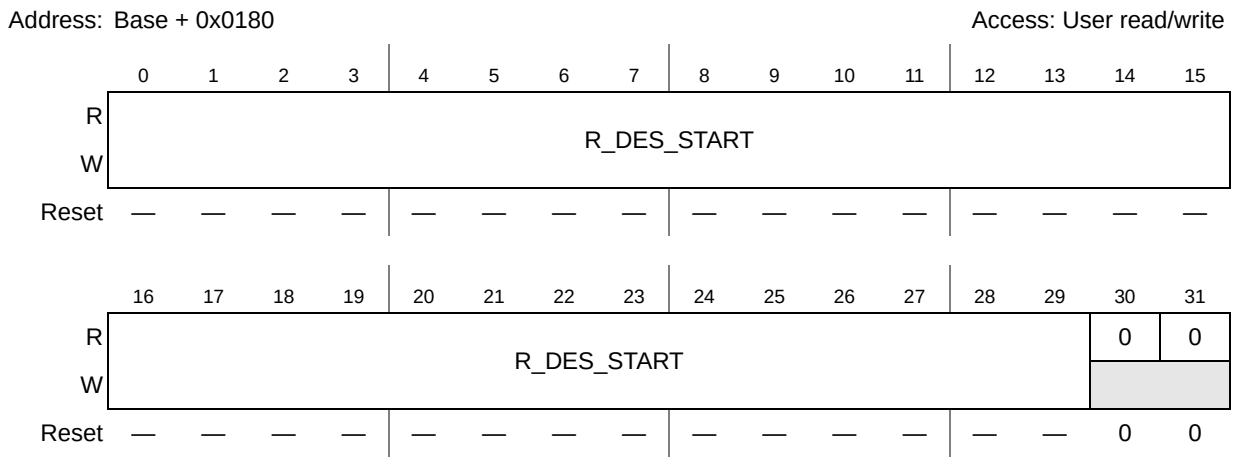


Figure 26-25. Receive Descriptor Ring Start Register (ERDSR)

Table 26-28. ERDSR field descriptions

Field	Descriptions
R_DES_START	Pointer to start of receive buffer descriptor queue.

26.3.4.25 Transmit Buffer Descriptor Ring Start Register (ETDSR)

The Transmit Buffer Descriptor Ring Start Register (ETDSR) provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). Bits 30 and 31 should be written to 0 by the user. Non-zero values in these two bit positions are ignored by the hardware.

This register is not reset and must be initialized by the user prior to operation.

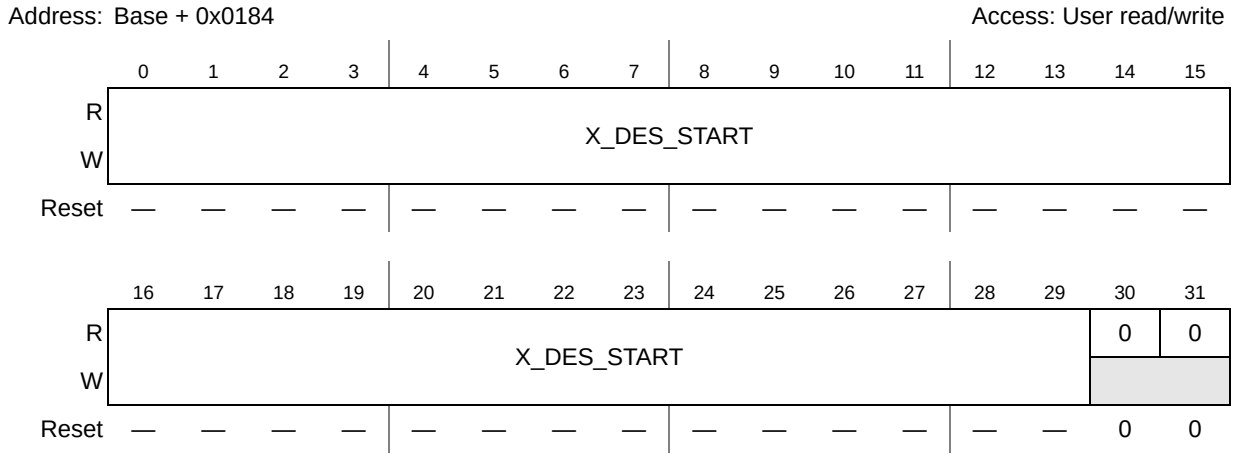


Figure 26-26. Transmit Buffer Descriptor Ring Start Register (ETDSR)

Table 26-29. ETDSR field descriptions

Field	Descriptions
X_DES_START	Pointer to start of transmit buffer descriptor queue.

26.3.4.26 Receive Buffer Size Register (EMRBR)

The Receive Buffer Size Register (EMRBR) dictates the maximum size of all receive buffers. Note that because receive frames are truncated at 2 KB – 1 bytes, only bits 21–27 are used. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR must be set to RCR[MAX_FL] or larger. The EMRBR must be evenly divisible by 16. To ensure this, bits 28–31 are forced low. To minimize bus utilization (descriptor fetches) it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register does not reset, and must be initialized by the user.

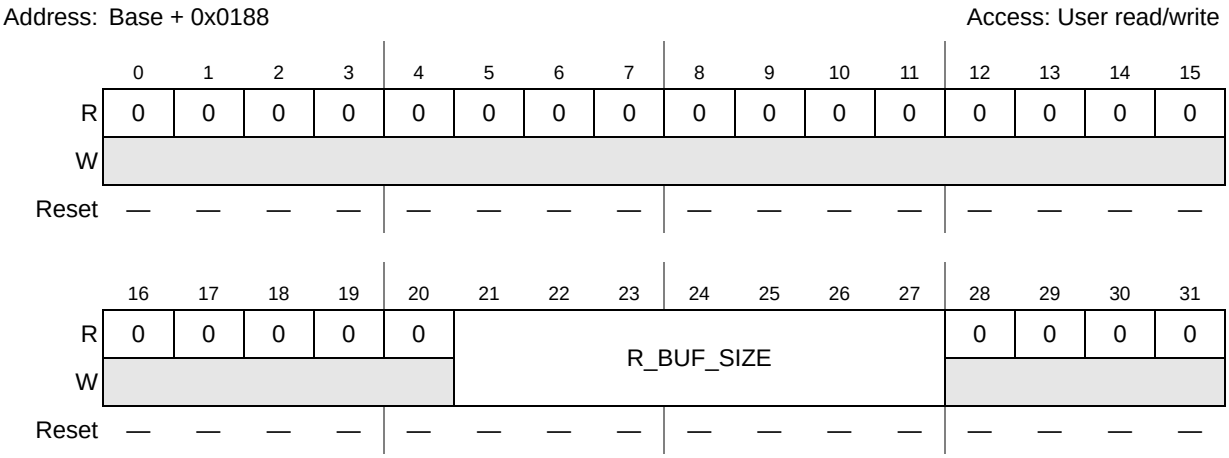


Figure 26-27. Receive Buffer Size Register (EMRBR)

Table 26-30. EMRBR field descriptions

Field	Descriptions
R_BUF_SIZE	Receive buffer size.

26.4 Functional description

This section describes the operation of the FEC, beginning with the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

After the software initialization and operation sections there are sections providing a detailed description of the functions of the FEC.

26.4.1 Initialization sequence

This section describes which registers are reset due to hardware reset, which registers are reset by the FEC RISC, and the locations that the user must initialize prior to enabling the FEC.

26.4.1.1 Hardware controlled initialization

In the FEC, registers and control logic that generate interrupts are reset by hardware. A hardware reset deasserts output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER_EN] bit is cleared. ECR[ETHER_EN] is deasserted by a hard reset or may be deasserted by software to halt operation. By deasserting ECR[ETHER_EN], the configuration control registers such as the TCR and RCR are not reset, but the entire data path is reset.

Table 26-31. ECR[ETHER_EN] de-assertion effect on FEC

Register/Machine	Reset value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

26.4.2 User initialization (prior to asserting ECR[ETHER_EN])

The user needs to initialize portions of the FEC prior to setting the ECR[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in Table 26-32.

Table 26-32. User initialization (before ECR[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM (FEC Base + 0x02E0)

FEC FIFO/DMA registers that require initialization are defined in Table 26-33.

Table 26-33. FEC user initialization (before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

26.4.3 Microcontroller initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

Table 26-34 shows microcontroller initialization operations.

Table 26-34. Microcontroller initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

26.4.4 User initialization (after asserting ECR[ETHER_EN])

After asserting ECR[ETHER_EN], the user can set up the buffer/frame descriptors and write to the TDAR and RDAR. Refer to Section 26.5, Buffer descriptors, for more details.

26.4.5 Network interface options

The FEC supports both an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The interface mode is selected by the RCR[MII_MODE] bit. In MII mode (RCR[MII_MODE] = 1), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. These signals are shown in Table 26-35 below.

Table 26-35. MII mode

Signal description	EMAC signal
Transmit Clock	FEC_TX_CLK
Transmit Enable	FEC_TX_EN
Transmit Data	FEC_TXD[3:0]
Transmit Error	FEC_TX_ER
Collision	FEC_COL
Carrier Sense	FEC_CRS
Receive Clock	FEC_RX_CLK
Receive Data Valid	FEC_RX_DV
Receive Data	FEC_RXD[3:0]
Receive Error	FEC_RX_ER
Management Data Clock	FEC_MDC
Management Data Input/Output	FEC_MDIO

The 7-wire serial mode interface (RCR[MII_MODE] = 0) operates in what is generally referred to as the “AMD” mode. 7-wire mode connections to the external transceiver are shown in Table 26-36.

Table 26-36. 7-wire mode configuration

Signal Description	FEC signal
Transmit Clock	FEC_TX_CLK
Transmit Enable	FEC_TX_EN
Transmit Data	FEC_TXD0
Collision	FEC_COL
Receive Clock	FEC_RX_CLK
Receive Data Valid	FEC_RX_DV
Receive Data	FEC_RXD0

26.4.6 FEC frame transmission

The Ethernet transmitter is designed to work with almost no intervention from software. Once ECR[ETHER_EN] is asserted and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the TFWR), the MAC transmit logic asserts FEC_TX_EN and starts transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (FEC_CRS asserts). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, the transmission begins after waiting

an additional 36 bit times (96 bit times after carrier sense originally became inactive). See Section 26.4.14.1, Transmission errors, for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (frame check sequence or 32-bit cyclic redundancy check, CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end of frame buffer = 1).

Both buffer (TXB) and frame (TFINT) interrupts may be generated as determined by the settings in the EIMR.

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes, the BABT interrupt is asserted but the entire frame is transmitted (no truncation). All these interrupts are mapped to a single interrupt vector (336) on the MPC5675K. Software must determine which interrupt has occurred in the interrupt sub-routine.

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR register. When the TCR[GTS] is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame either finishes or terminates with a collision. After the transmitter has stopped, the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

26.4.7 FEC frame reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by asserting ECR[ETHER_EN], it starts processing receive frames immediately. When FEC_RX_DV asserts, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the frame processed by the receiver. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of FEC_RXD0 following assertion of FEC_RX_DV are ignored. Following the first 16 bit times the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame have been received, the FEC performs address recognition on the frame.

Once a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, the receive FIFO is signaled that the frame is “accepted” and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to “reject” the frame. Thus, no collision fragments are presented to the user except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and once the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV and TR status bits, and the frame length. See Section 26.4.14.2, Reception errors, for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt occurs and the LG bit in the receive buffer descriptor (RxBD) is set. See Section 26.5.2, Ethernet receive buffer descriptor (RxBD), for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data LSB first.

26.4.8 Ethernet address recognition

The FEC filters the received frames based on the type of destination address (DA)—individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit (bit 40) in the destination address field. A flowchart for address recognition on received frames is illustrated in Figure 26-28 and Figure 26-29.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in Figure 26-28 illustrates the address recognition decisions made by the receive block, while Figure 26-29 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is deasserted, then the frame is accepted unconditionally, as shown in Figure 26-28. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 26-29.

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

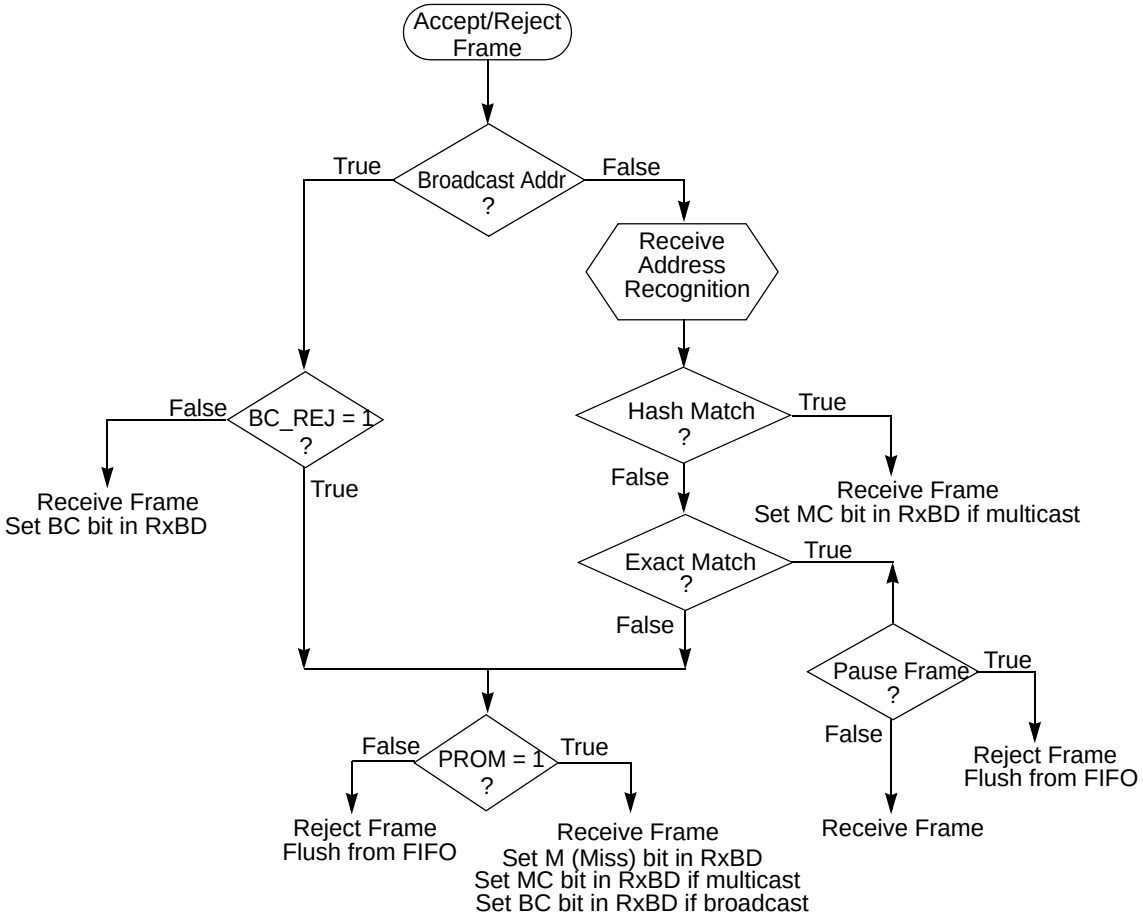
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines that the received frame is a valid PAUSE frame, then the frame is rejected. Note the receiver detects a PAUSE frame with the DA field set to either the designated PAUSE DA or to the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or rejects the frame based on PAUSE frame detection, as shown in Figure 26-28.

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then if promiscuous mode is enabled (RCR[PROM] = 1), the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

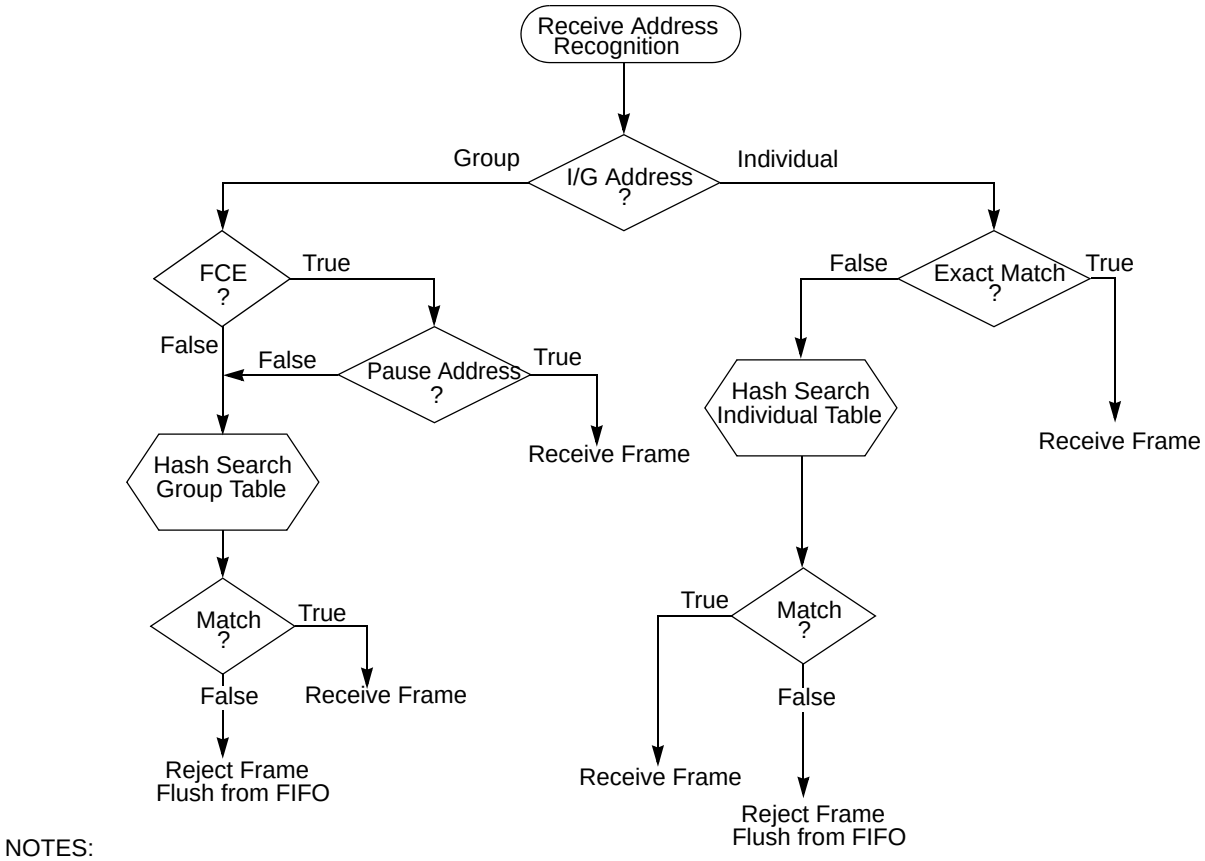
Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC_REJ]) is asserted, and promiscuous mode is enabled, then the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected and the MISS bit is cleared.

In general, when a frame is rejected, it is flushed from the FIFO.



NOTES:
BC_REJ — field in RCR register (BroadCast REJect)
PROM — field in RCR register (PROMiscuous mode)
Pause Frame — valid Pause frame received

Figure 26-28. Ethernet address recognition—receive block decisions



NOTES:
FCE — field in RCR register (Flow Control Enable)
I/G — Individual/Group bit in Destination Address (least significant bit in first byte received in MAC frame)

Figure 26-29. Ethernet address recognition—microcode decisions

26.4.9 Hash algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GAUR, GALR (group address hash match) or IAUR, IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GAUR (MSB = 1) or GALR (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The hash table registers must be initialized by the user. The CRC32 polynomial to use in computing the hash is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

A table of example Destination Addresses and corresponding hash values is included below for reference.

Table 26-37. Destination address to 6-bit hash

48-bit DA	6-bit Hash (in hex)	Hash decimal value
65:FF:FF:FF:FF:FF	0x0	0
55:FF:FF:FF:FF:FF	0x1	1
15:FF:FF:FF:FF:FF	0x2	2
35:FF:FF:FF:FF:FF	0x3	3
B5:FF:FF:FF:FF:FF	0x4	4
95:FF:FF:FF:FF:FF	0x5	5
D5:FF:FF:FF:FF:FF	0x6	6
F5:FF:FF:FF:FF:FF	0x7	7
DB:FF:FF:FF:FF:FF	0x8	8
FB:FF:FF:FF:FF:FF	0x9	9
BB:FF:FF:FF:FF:FF	0xA	10
8B:FF:FF:FF:FF:FF	0xB	11
0B:FF:FF:FF:FF:FF	0xC	12
3B:FF:FF:FF:FF:FF	0xD	13
7B:FF:FF:FF:FF:FF	0xE	14
5B:FF:FF:FF:FF:FF	0xF	15
27:FF:FF:FF:FF:FF	0x10	16
07:FF:FF:FF:FF:FF	0x11	17
57:FF:FF:FF:FF:FF	0x12	18
77:FF:FF:FF:FF:FF	0x13	19
F7:FF:FF:FF:FF:FF	0x14	20
C7:FF:FF:FF:FF:FF	0x15	21
97:FF:FF:FF:FF:FF	0x16	22
A7:FF:FF:FF:FF:FF	0x17	23
99:FF:FF:FF:FF:FF	0x18	24
B9:FF:FF:FF:FF:FF	0x19	25
F9:FF:FF:FF:FF:FF	0x1A	26

Table 26-37. Destination address to 6-bit hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash decimal value
C9:FF:FF:FF:FF:FF	0x1B	27
59:FF:FF:FF:FF:FF	0x1C	28
79:ff:ff:ff:ff:ff	0x1D	29
29:ff:ff:ff:ff:ff	0x1E	30
19:ff:ff:ff:ff:ff	0x1F	31
D1:FF:FF:FF:FF:FF	0x20	32
F1:FF:FF:FF:FF:FF	0x21	33
B1:FF:FF:FF:FF:FF	0x22	34
91:FF:FF:FF:FF:FF	0x23	35
11:FF:FF:FF:FF:FF	0x24	36
31:FF:FF:FF:FF:FF	0x25	37
71:FF:FF:FF:FF:FF	0x26	38
51:FF:FF:FF:FF:FF	0x27	39
7F:FF:FF:FF:FF:FF	0x28	40
4F:FF:FF:FF:FF:FF	0x29	41
1F:FF:FF:FF:FF:FF	0x2A	42
3F:FF:FF:FF:FF:FF	0x2B	43
BF:FF:FF:FF:FF:FF	0x2C	44
9F:FF:FF:FF:FF:FF	0x2D	45
DF:FF:FF:FF:FF:FF	0x2E	46
EF:FF:FF:FF:FF:FF	0x2F	47
93:FF:FF:FF:FF:FF	0x30	48
B3:FF:FF:FF:FF:FF	0x31	49
F3:FF:FF:FF:FF:FF	0x32	50
D3:FF:FF:FF:FF:FF	0x33	51
53:FF:FF:FF:FF:FF	0x34	52
73:FF:FF:FF:FF:FF	0x35	53
23:FF:FF:FF:FF:FF	0x36	54
13:FF:FF:FF:FF:FF	0x37	55
3D:FF:FF:FF:FF:FF	0x38	56
0D:FF:FF:FF:FF:FF	0x39	57
5D:FF:FF:FF:FF:FF	0x3A	58
7D:FF:FF:FF:FF:FF	0x3B	59

Table 26-37. Destination address to 6-bit hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash decimal value
FD:FF:FF:FF:FF:FF	0x3C	60
DD:FF:FF:FF:FF:FF	0x3D	61
9D:FF:FF:FF:FF:FF	0x3E	62
BD:FF:FF:FF:FF:FF	0x3F	63

26.4.10 Full duplex flow control

Full-duplex flow control allows the user to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] asserted) and flow control enable (RCR[FCE]) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in the table below. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 26-38. PAUSE frame field specification

48-bit destination address	0x0180_C200_0001 or physical address
48-bit Source Address	Any
16-bit TYPE	0x8808
16-bit OPCODE	0x0001
16-bit PAUSE_DUR	0x0000 to 0xFFFF

Pause frame detection is performed by the receiver and microcontroller modules. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the TYPE and OPCODE pause frame fields. On detection of a pause frame, TCR[GTS] is asserted by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. Note that the pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is deasserted allowing MAC data frame transmission to resume. Note that the receive flow control pause (TCR[RFC_PAUSE]) status bit is asserted while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and the user must assert flow control pause (TCR[TFC_PAUSE]). On assertion of transmit flow control pause (TCR[TFC_PAUSE]), the transmitter asserts TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts. Following EIR[GRA] assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause (TCR[TFC_PAUSE]) and TCR[GTS] are deasserted internally.

Table 26-39. Transmit pause frame registers

Pause frame fields	FEC register	Register contents
48-bit source address	{PAUR.PADDR1[0:31], PAUR.PADDR2[0:15]}	physical address
16-bit type	PAUR.PADDR2[16:31]	0x8808
16-bit opcode	PAUR.OP_PAUSE[0:15]	0x0001
16-bit pause duration	PAUR.OP_PAUSE[16:31]	0x0000 to 0xFFFF

The user must specify the desired pause duration in the OPD register.

Note that when the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause (TCR[TFC_PAUSE]) may still be asserted and causes the transmission of a single pause frame. In this case, the EIR[GRA] interrupt is not asserted.

26.4.11 Inter-Packet Gap (IPG) time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96-bit-time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the following frame may be discarded by the receiver.

26.4.12 Collision handling

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern is sent after the end of the preamble sequence.

If a collision occurs within 512 bit times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a late collision (LC) error indication.

26.4.13 Internal and external loopback

Both internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of the LOOP and DRT bits in the RCR register and the FDEN bit in the TCR register.

For both internal and external loopback set FDEN = 1.

For internal loopback, set RCR[LOOP] = 1 and RCR[DRT] = 0. FEC_TX_EN and FEC_TX_ER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in

normal operation because the internal system clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMAed to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback set RCR[LOOP] = 0, RCR[DRT] = 0, and configure the external transceiver for loopback.

26.4.14 Ethernet error-handling procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the EIR register, and the MIB block counters.

26.4.14.1 Transmission errors

26.4.14.1.1 Transmitter underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The UN interrupt is asserted if enabled in the EIMR register. This is mapped to interrupt vector 336.

26.4.14.1.2 Retransmission attempts limit expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the RL bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The RL interrupt is asserted if enabled in the EIMR register. This is mapped to interrupt vector 336.

26.4.14.1.3 Late collision

When a collision occurs after the slot time (512 bits starting at the preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the LC bit is set in the EIR register. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame.

The LC interrupt is asserted if enabled in the EIMR register. This is mapped to interrupt vector 336.

26.4.14.1.4 Heartbeat

Some transceivers have a self-test feature called “heartbeat” or “signal quality error.” To signify a good self-test, the transceiver indicates a collision to the FEC within four microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the TCR register and the heartbeat condition is not detected by the FEC after a frame transmission, then a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets the HB bit in the EIR register, and generates the HBERR interrupt if it is enabled. This is mapped to interrupt vector 336.

26.4.14.2 Reception errors

26.4.14.2.1 Overrun error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the RxBD. All subsequent data in the frame is discarded. Subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.

26.4.14.2.2 Non-octet error (dribbling bits)

The Ethernet controller handles as many as seven dribbling bits when the receive frame terminates past a non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If a CRC error occurs, then the frame non-octet aligned (NO) error is reported in the RxBD. If no CRC error is detected, no error is reported.

26.4.14.2.3 CRC error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

26.4.14.2.4 Frame length violation

When the receive frame length exceeds MAX_FL bytes, the BABR interrupt is generated, and the LG bit in the end of frame RxBD is set. This is mapped to interrupt vector 336. The frame is not truncated unless the frame length exceeds 2047 bytes.

26.4.14.2.5 Truncation

When the receive frame length exceeds 2047 bytes the frame is truncated, and the TR bit is set in the RxBD.

26.5 Buffer descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

26.5.1 Driver/DMA operation with buffer descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. Associated with each buffer is a buffer descriptor (BD) that contains a starting address (pointer), data length, and status/control information (which contains the current state for the buffer). To

permit maximum user flexibility, the BDs are also located in external memory and are read in by the FEC DMA engine.

Software “produces” buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit “produces” the buffer. Software writing to either the TDAR or RDAR tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and “consumes” the buffers after they have been produced. After the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the RxBD[E] or TxBD[R] bit is cleared by hardware to signal that the buffer has been “consumed.” Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ECR[ETHER_EN] signal operates as a reset to the BD/DMA logic. When ECR[ETHER_EN] is deasserted the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before the ECR[ETHER_EN] bit is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they be 128-bit aligned.

26.5.1.1 Driver/DMA operation with transmit BDs

Typically a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a second buffer, IP header in a third buffer, and Ethernet/IEEE 802.3 header in a fourth buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type fields), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD, which must be set by the driver.

The driver (TxBD software producer) should set up Tx BDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the TxBD[R] bits should be set = 1 in reverse order (3rd, 2nd, 1st BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the second was made available, potentially causing a transmit FIFO underrun.

In the FEC, the DMA is notified by the driver that new transmit frames are available by writing to the TDAR register. When this register is written to (data value is not significant), the FEC RISC tells the DMA to read the next transmit BD in the ring. Once started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers, until a transmit BD is encountered with the R bit = 0. At this point, the FEC polls this BD one more time. If the R bit = 0 the second time, then the RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer. A second driver task (Tx BD software consumer) processes the transmit descriptor ring and returns buffers consumed by the hardware to the free list.

26.5.1.2 Driver/DMA operation with receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of “empty” buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L (1 indicates last buffer in frame) bit, the frame status bits (if L = 1), and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end of frame buffers, the receive BD is written with L = 1 and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators that, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame, not just the length of the last buffer.

For simplicity the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out of spec implementation could result in giant frames. Frames of 2KB (2048) bytes or larger are truncated by the FEC at 2047 bytes so software does not have to take care of receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received, the FEC fills receive buffers and updates the associated BDs, then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit = 0, it polls this BD once more. If the BD = 0 a second time, the FEC stops reading receive BDs until the driver writes to RDAR.

26.5.2 Ethernet receive buffer descriptor (RxBD)

In the RxBD, the user initializes the E and W bits in the first word and the pointer in the second word. When the buffer has been DMAed, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first word. The M, BC, MC, LG, NO, CR, OV and TR bits in the first word of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data Length															
Offset + 4	Rx Data Buffer Pointer - A [0:15]															
Offset + 6	Rx Data Buffer Pointer - A [16:31]															

Figure 26-30. Receive buffer descriptor (RxBD)

Table 26-40. Receive buffer descriptor field definitions

Halfword	Location	Field Name	Description
Offset + 0	Bit 0	E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data, or data reception has been aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	Bit 1	RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 2	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in ERDSR.
Offset + 0	Bit 3	RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 4	L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	Bits 5-6	—	Reserved.
Offset + 0	Bit 7	M	Miss. Written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition. Thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	Bit 8	BC	Set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	Bit 9	MC	Set if the DA is multicast and not BC.
Offset + 0	Bit 10	LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	Bit 11	NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. When this bit is set, the CR bit is not set.

Table 26-40. Receive buffer descriptor field definitions (continued)

Halfword	Location	Field Name	Description
Offset + 0	Bit 12	—	Reserved.
Offset + 0	Bit 13	CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	Bit 14	OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and are zero. This bit is valid only if the L bit is set.
Offset + 0	Bit 15	TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set the frame should be discarded and the other error bits should be ignored as they may be incorrect.
Offset + 2	Bits [0:15]	Data Length	Data length. Written by the FEC. Data length is the number of 8-bit data groups (octets) written by the FEC into this BD's data buffer if L = 0 (the value is equal to EMRBR), or the length of the frame including CRC if L = 1. It is written by the FEC once as the BD is closed.
0ffset + 4	Bits [0:15]	A[0:15]	RX data buffer pointer, bits [0:15][1]
Offset + 6	Bits [0:15]	A[16:31]	RX data buffer pointer, bits [16:31]

[1] The receive buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Whenever the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDAR.

26.5.3 Ethernet transmit buffer descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (R bit) when DMA of the buffer is complete. In the TxBD the user initializes the R, W, L, and TC bits and the length (in bytes) in the first word, and the buffer pointer in the second word.

The FEC sets the R bit = 0 in the first word of the BD when the buffer has been DMAed. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See Section 26.3.3, MIB Block Counters Memory Map, for more details.

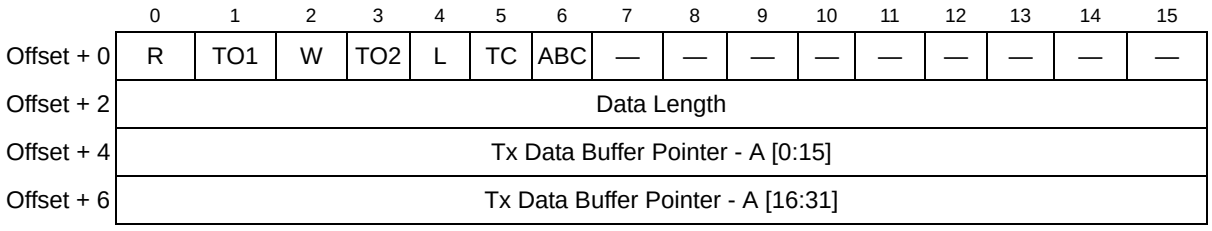


Figure 26-31. Transmit buffer descriptor (TxBD)

Table 26-41. Transmit buffer descriptor field definitions

Halfword	Location	Field Name	Description
Offset + 0	Bit 0	R	Ready. Written by the FEC and the user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set.
Offset + 0	Bit 1	TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 2	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in ETDSR.
Offset + 0	Bit 3	TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, and its value does not affect hardware.
Offset + 0	Bit 4	L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.
Offset + 0	Bit 5	TC	Tx CRC. Written by user (only valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
Offset + 0	Bit 6	ABC	Append bad CRC. Written by user (only valid if L = 1). 0 No effect. 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value).
Offset + 0	Bits [7:15]	—	Reserved.
Offset + 2	Bits [0:15]	Data Length	Data length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [0:10] are used by the DMA engine, bits[11:15] are ignored.
Offset + 4	Bits [0:15]	A[0:15]	Tx data buffer pointer, bits [0:15][1]
Offset + 6	Bits [0:15]	A[16:31]	Tx data buffer pointer, bits [16:31].

[1] The transmit buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by four. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Once the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the R bit in the first BD for the frame. The driver should follow that with a write to TDAR, which triggers the FEC to poll the next BD in the ring.

Chapter 27
FlexCAN Module

27.1 Introduction

The FlexCAN module is a communication controller implementing the FlexCAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in Figure 27-1, which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing message buffers (MB) and another one for storing Rx individual mask registers. The functions of the sub-modules are described in subsequent sections.

The MPC5675K device includes four FlexCAN modules, referred to as FlexCAN_0 through FlexCAN_3. Each module supports 32 message buffers.

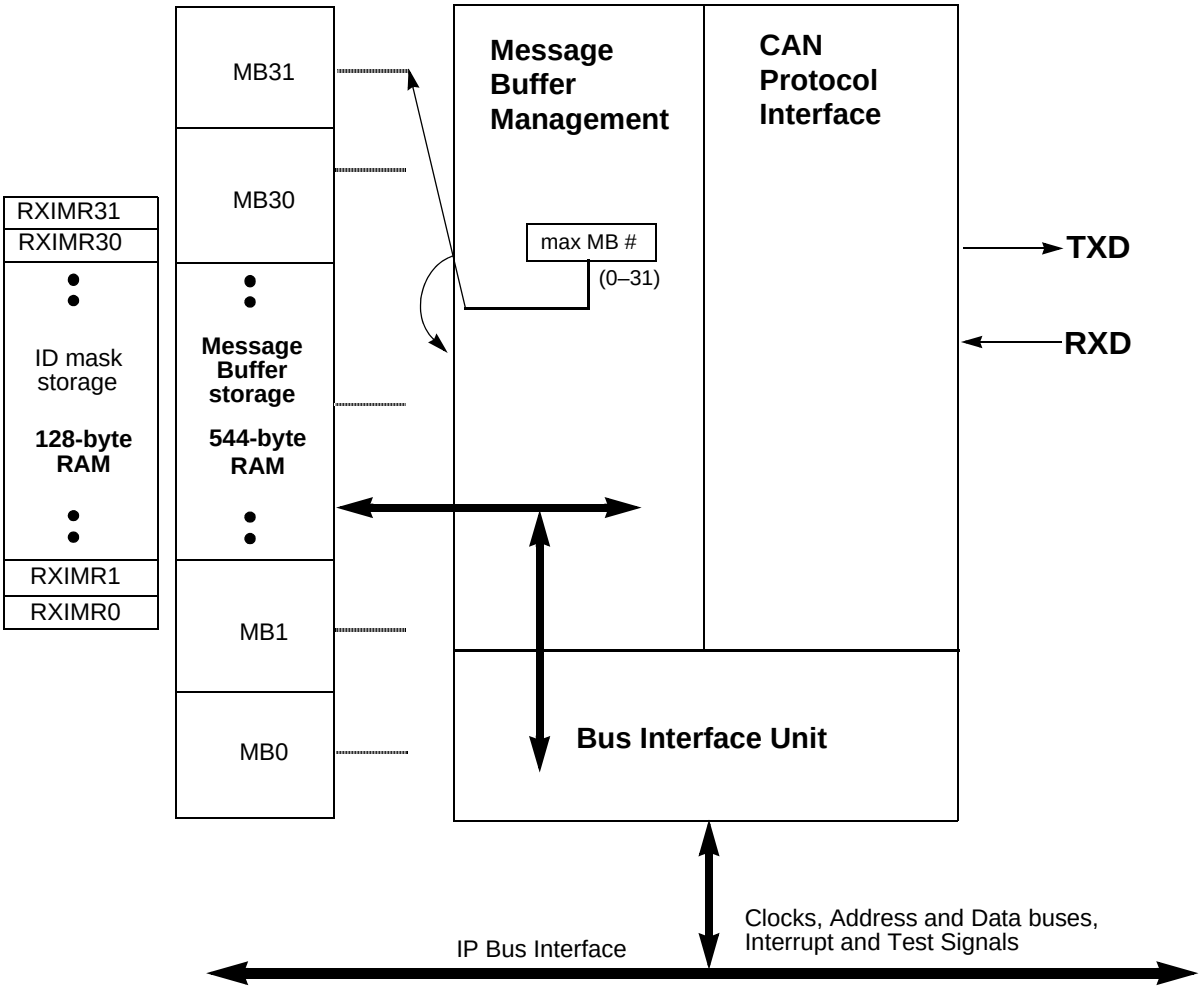


Figure 27-1. FlexCAN block diagram

27.1.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) sub-module handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) sub-module controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the BIU.

27.1.2 FlexCAN module features

The FlexCAN module includes these features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0 to 8 bytes data length
  - Programmable bit rate up to 1 Mbit/s
  - Content-related addressing
- Compliant with the ISO 11898-1 standard
- As many as 32 flexible message buffers (MB) of 0 to 8 bytes data length each
- Each MB configurable as receive (Rx) or transmit (Tx), all supporting standard and extended messages
- Individual Rx mask registers per MB
- Includes 544 bytes of RAM used for MB storage
- Includes 128 bytes of RAM used for individual Rx mask registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard, or 32 partial (8-bit) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN versions
- Programmable clock source to the CAN Protocol Interface, either FlexCAN clock or XOSC clock
- Unused MB and Rx mask register space can be used as general purpose RAM space
- Listen-only mode capability
- Programmable loopback mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number, or highest priority

- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes

27.1.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only mode, and Loopback mode. Additionally, these low power modes are implemented: Disable mode and Stop mode.

- Normal mode (User or Supervisor):

  In Normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally, and all the CAN protocol functions are enabled. User and Supervisor modes differ in the access to some restricted control registers.
- Freeze mode:

  Freeze mode is enabled when the MCR[FRZ] bit is set. If enabled, Freeze mode is entered when the HALT bit in MCR is set or when Debug mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See Section 27.4.8.6, Freeze mode, for more information.
- Listen-Only mode:

  The module enters this mode when the CTRL[LOM] bit is set. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station are received. If the FlexCAN module detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- Loopback mode:

  The module enters this mode when the CTRL[LPB] bit is set. In this mode, the FlexCAN module performs an internal loopback that can be used for self-test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The RXD pin is ignored and the TXD pin goes to the recessive state (logic 1). The FlexCAN module behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, the FlexCAN module ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.
- Module Disable mode:

  This low power mode is entered when the MDIS bit in the MCR Register is asserted by the CPU. When disabled, the module requests to disable the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by clearing the MCR[MDIS] bit. See Section 27.4.8.7, Module disable mode, for more information.
- Stop mode:

This low power mode is entered when Stop mode is requested at the MCU level. When in Stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop mode request is removed or when activity is detected on the CAN bus and the self wake-up mechanism is enabled. See Section 27.4.8.8, Stop mode, for more information.

27.2 External signal description

27.2.1 Overview

Each FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in Table 27-1 and described in more detail in the next subsections.

Table 27-1. FlexCAN signals

Signal Name	Direction	Description
RXD	Input	CAN Receive Pin
TXD	Output	CAN Transmit Pin

27.2.2 Signal descriptions

27.2.2.1 RXD

The RXD pin is the receive pin from the CAN bus transceiver. The dominant state is represented by logic level 0. The recessive state is represented by logic level 1.

27.2.2.2 TXD

The TXD pin is the transmit pin to the CAN bus transceiver. The dominant state is represented by logic level 0. The recessive state is represented by logic level 1.

27.3 Memory map and register description

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by each FlexCAN module has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID mask storage space in a separate embedded RAM starting at address 0x0880.

27.3.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 32 MBs capability is shown in Table 27-3. Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). These registers are identified as S/U in the Access type column of Table 27-4.

The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK), and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the Backwards Compatibility Configuration (BCC) bit in MCR is set.

If the MCR[BCC] bit is cleared, then the whole Rx individual mask registers address range (0x0880–0x097F) is considered reserved space.

Table 27-2 shows the base addresses for the ADC modules. Addresses are the same for Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). Table 27-3 shows the memory map for the ADC program-visible registers.

Table 27-2. FlexCAN module base addresses

Mode	Module	Module base address
LSM and DPM	FlexCAN_0	0xFFFC_0000
	FlexCAN_1	0xFFFC_4000
	FlexCAN_2	0xFFFC_8000
	FlexCAN_3	0xFFFC_C000

Table 27-3. FlexCAN memory map

Offset from FlexCAN_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Module Configuration (MCR)	R/W	0xD890_000F	on page 849
0x0004	Control Register (CTRL)	R/W	0x0000_0000	on page 853
0x0008	Free Running Timer (TIMER)	R/W	0x0000_0000	on page 856
0x000C	Reserved			
0x0010	Rx Global Mask (RXGMASK)	R/W	0xFFFF_FFFF	on page 857
0x0014	Rx Buffer 14 Mask (RX14MASK)	R/W	0xFFFF_FFFF	on page 858
0x0018	Rx Buffer 15 Mask (RX15MASK)	R/W	0xFFFF_FFFF	on page 859
0x001C	Error Counter Register (ECR)	R/W	0x0000_0000	on page 860
0x0020	Error and Status Register (ESR)	R/W	0x0000_0000	on page 861
0x0024	Reserved			
0x0028	Interrupt Masks 1 (IMASK1)	R/W	0x0000_0000	on page 864
0x002C	Reserved			
0x0030	Interrupt Flags 1 (IFLAG1)	R/W	0x0000_0000	on page 864
0x0034–0x007F	Reserved			
0x0080–0x017F	Message Buffers MB0–MB15	R/W	0x0000_0000	on page 843
0x0180–0x027F	Message Buffers MB16–MB31	R/W	0x0000_0000	on page 843
0x0280–0x087F	Reserved			
0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	R/W	0x0000_0000	on page 866

Table 27-3. FlexCAN memory map (continued)

Offset from FlexCAN_BASE	Register	Access[1]	Reset Value[2]	Location
0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	R/W	0x0000_0000	on page 866
0x0900–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

Table 27-4 shows the effect of reset on the FlexCAN registers.

Table 27-4. Reset effect on registers

Use	Access type	Affected by hard reset	Affected by soft reset
Module Configuration (MCR)	S	Yes	Yes
Control Register (CTRL)	S/U	Yes	No
Free Running Timer (TIMER)	S/U	Yes	Yes
Rx Global Mask (RXGMASK)	S/U	Yes	No
Rx Buffer 14 Mask (RX14MASK)	S/U	Yes	No
Rx Buffer 15 Mask (RX15MASK)	S/U	Yes	No
Error Counter Register (ECR)	S/U	Yes	Yes
Error and Status Register (ESR)	S/U	Yes	Yes
Interrupt Masks 1 (IMASK1)	S/U	Yes	Yes
Interrupt Flags 1 (IFLAG1)	S/U	Yes	Yes
Message Buffers MB0–MB15	S/U	No	No
Message Buffers MB16–MB31	S/U	No	No
Rx Individual Mask Registers RXIMR0-RXIMR15	S/U	No	No
Rx Individual Mask Registers RXIMR16-RXIMR31	S/U	No	No

The FlexCAN module stores CAN messages for transmission and reception using a message buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in Table 27-5. Table 27-5 shows a standard/extended message buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 27-5. Message buffer MB0 memory mapping

Address Offset	MB Field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

27.3.2 Message buffer structure

The message buffer structure used by the FlexCAN module is represented in Figure 27-2. Both extended and standard frames (29-bit identifier and 11-bit identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

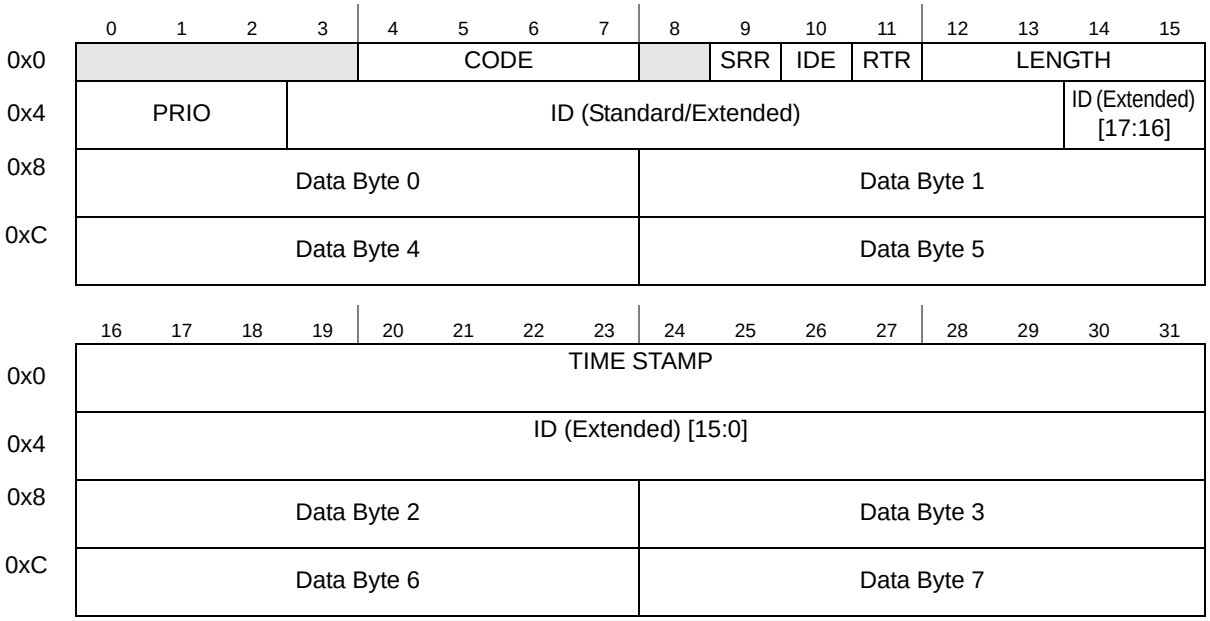


Figure 27-2. Message buffer structure

Table 27-6. Message buffer structure field description

Field	Description
CODE	Message Buffer Code This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 27-7 and Table 27-8 for Rx and Tx buffers, respectively. See Section 27.4, Functional description, for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to 1 by the user for transmission (Tx buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If the FlexCAN module receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in extended format frames. 1 Recessive value is compulsory for transmission in extended format frames.
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 0 Frame format is standard. 1 Frame format is extended.

Table 27-6. Message buffer structure field description (continued)

Field	Description
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If the FlexCAN module transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as a bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted. 1 Indicates the current MB has a remote rrame to be transmitted.
LENGTH	Length of Data in Bytes This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see Figure 27-2). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the frame to be transmitted is a remote frame and does not include the data field, regardless of the Length field.
TIME STAMP	Free-Running Counter Time Stamp This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the identifier field appears on the CAN bus.
PRIO	Local priority This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See Section 27.4.3, Arbitration process.
ID	Frame identifier In Standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In extended frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data field As many as 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 27-7. Message buffer code for Rx buffers

Rx code BEFORE Rx new frame	Description	Rx code AFTER Rx new frame	Comment
0000	INACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.

Table 27-7. Message buffer code for Rx buffers (continued)

Rx code BEFORE Rx new frame	Description	Rx code AFTER Rx new frame	Comment
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 27.4.5, Matching process, for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN. Refer to Section 27.4.5, Matching process, for details about overrun behavior.
0XY1[1]	BUSY: FlexCAN is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1 For Tx MBs (see Table 27-8), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register.

Table 27-8. Message buffer code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when the MCR[AEN] bit is set. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to start the process again.

Table 27-8. Message buffer code for Tx buffers (continued)

RTR	Initial Tx code	Code after successful transmission	Description
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame is transmitted unconditionally once and then the code automatically returns to '1010'. The CPU can also write this code with the same effect.

NOTE

FlexCAN does not transmit an expected message when the same node detects an incoming Remote Request message asking for any remote answer.

The issue happens when two specific conditions occur:

1) The Message Buffer (MB) configured for remote answer (with code "a") is the last MB. The last MB is specified by Maximum MB field in the Module Configuration Register (MCR[MAXMB]).

2) The incoming Remote Request message does not match its ID against the last MB ID.

While an incoming Remote Request message is being received, the FlexCAN also scans the transmit (Tx) MBs to select the one with the higher priority for the next bus arbitration. It is expected that by the Intermission field it ends up with a selected candidate (winner). The coincidence of conditions (1) and (2) above creates an internal corner case that cancels the Tx winner and therefore no message will be selected for transmission in the next frame. This gives the appearance that the FlexCAN transmitter is stalled or "stops transmitting".

The problem can be detectable only if the message traffic ceases and the CAN bus enters into Idle state after the described sequence of events.

There is NO ISSUE if any of the conditions below holds:

a) The incoming message matches the remote answer MB with code "a".

b) The MB configured as remote answer with code "a" is not the last one.

c) Any MB (despite of being Tx or Rx) is reconfigured (by writing its CS field) just after the Intermission field.

d) A new incoming message sent by any external node starts just after the Intermission field.

To manage this behavior, do not configure the last MB as a Remote Answer (with code "a").

Table 27-9. MB0–MB31 addresses

Address	Register	Address	Register
Base + 0x0080	MB0	Base + 0x0180	MB16
Base + 0x0090	MB1	Base + 0x0190	MB17
Base + 0x00A0	MB2	Base + 0x01A0	MB18
Base + 0x00B0	MB3	Base + 0x01B0	MB19
Base + 0x00C0	MB4	Base + 0x01C0	MB20
Base + 0x00D0	MB5	Base + 0x01D0	MB21
Base + 0x00E0	MB6	Base + 0x01E0	MB22
Base + 0x00F0	MB7	Base + 0x01F0	MB23
Base + 0x0100	MB8	Base + 0x0200	MB24
Base + 0x0110	MB9	Base + 0x0210	MB25
Base + 0x0120	MB10	Base + 0x0220	MB26
Base + 0x0130	MB11	Base + 0x0230	MB27
Base + 0x0140	MB12	Base + 0x0240	MB28
Base + 0x0150	MB13	Base + 0x0250	MB29
Base + 0x0160	MB14	Base + 0x0260	MB30
Base + 0x0170	MB15	Base + 0x0270	MB31

27.3.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. Figure 27-3 shows the Rx FIFO data structure. The region 0x80–0x8C contains an MB structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. Figure 27-4 shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See Section 27.4.7, Rx FIFO, for more information.

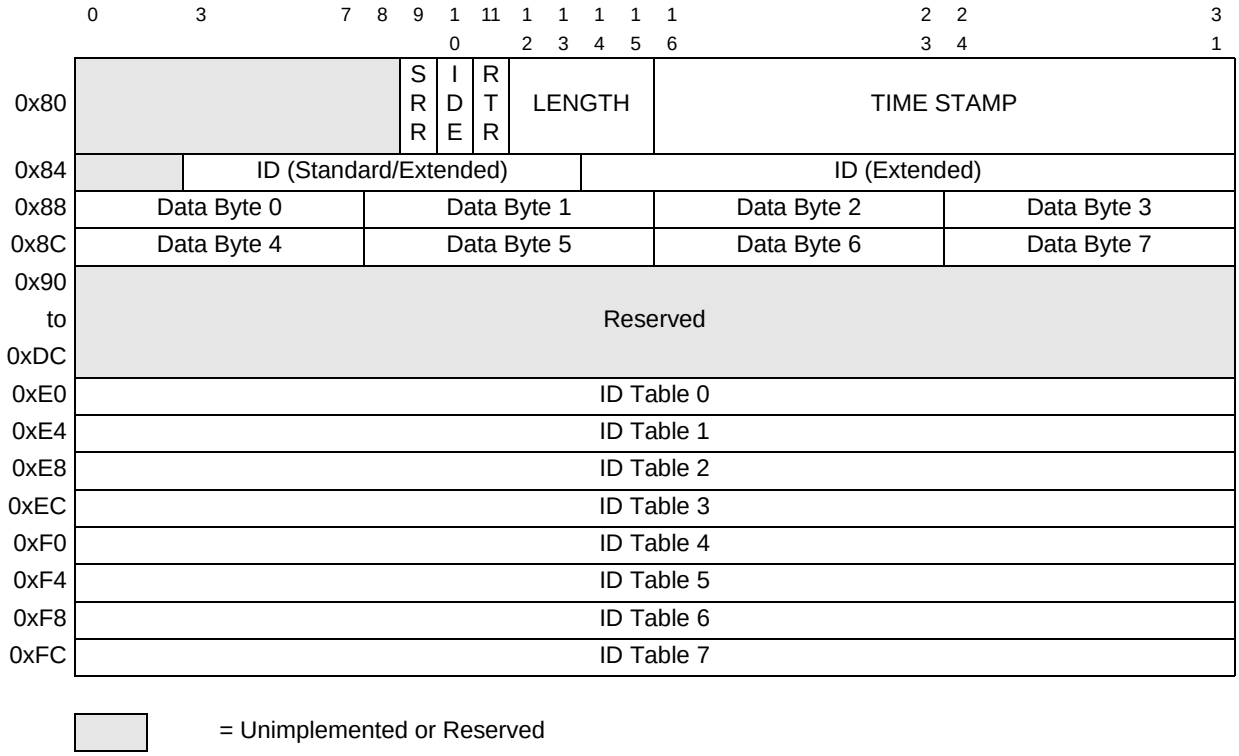


Figure 27-3. Rx FIFO structure

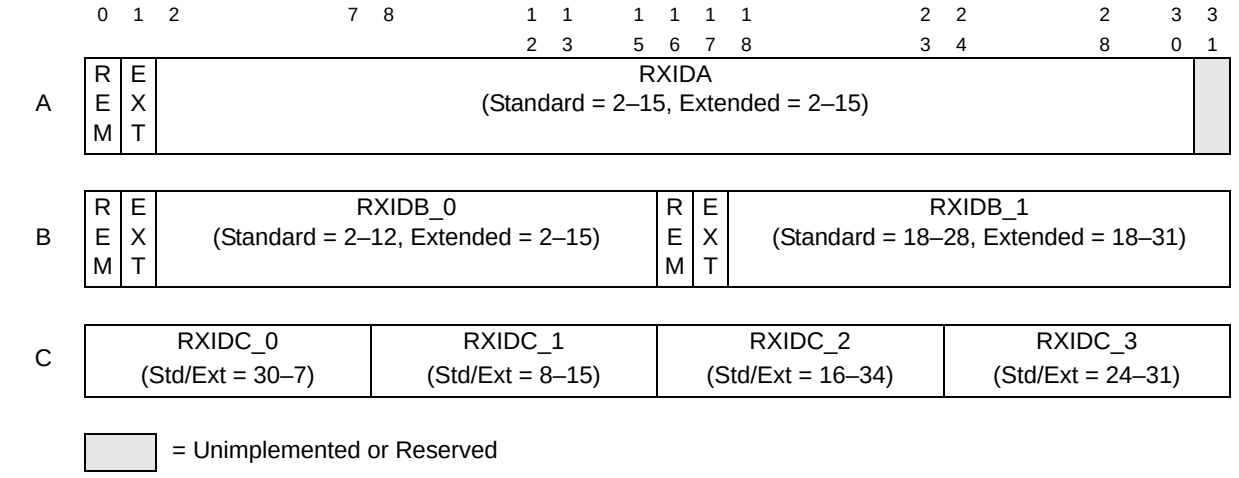


Figure 27-4. ID Table 0–7

Table 27-10. ID Table field descriptions

Field	Description
REM	Remote frame. This bit specifies if remote frames are accepted into the FIFO if they match the target ID. 0 Remote frames are rejected and data frames can be accepted. 1 Remote frames can be accepted and data frames are rejected.
EXT	Extended frame. Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted. 1 Extended frames can be accepted and standard frames are rejected.
RXIDA	Rx frame identifier (Format A). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (2 to 12) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_[0:1]	Rx frame identifier (Format B). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (2 to 12 and 18 to 28) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_[0:3]	Rx frame identifier (Format C). Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

27.3.4 Register descriptions

The FlexCAN registers are described in this section in ascending address order.

27.3.4.1 Module Configuration Register (MCR)

The Module Configuration Register (MCR) defines global system configurations, such as the module operation mode (for example, low power) and maximum message buffer configuration. This register can be accessed at any time; however, some fields must be changed only during Freeze mode.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	WAK_MSK	SOFT_RST	FRZ_ACK	SUPV	SLF_WAK	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
Reset	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 27-5. Module Configuration Register (MCR)

Table 27-11. MCR field descriptions

Field	Description
MDIS	Module Disable This bit controls whether the FlexCAN module is enabled or not. When disabled, the FlexCAN module shuts down the clocks to the FlexCAN protocol interface and message buffer management submodules. This is the only bit in MCR not affected by soft reset. See Section 27.4.8.7, Module disable mode, for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
FRZ	Freeze Enable This bit specifies the FlexCAN behavior when the HALT bit in the MCR is set or when Debug mode is requested at MCU level. When FRZ is set, the FlexCAN module is able to enter Freeze mode. Negation of this bit field causes the FlexCAN module to exit from Freeze mode. 0 Not able to enter Freeze mode. 1 Able to enter Freeze mode.
FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See Section 27.3.3, Rx FIFO structure, and Section 27.4.7, Rx FIFO, for more information. 0 FIFO disabled. 1 FIFO enabled. **Note:** This bit must be written in Freeze mode only.
HALT	Halt FlexCAN Setting this bit puts the FlexCAN module into Freeze mode. The CPU should clear it after initializing the message buffers and Control Register. No reception or transmission is performed by the FlexCAN module before this bit is cleared. While in Freeze mode, the CPU has write access to the ECR register that is otherwise read-only. Freeze mode cannot be entered while the FlexCAN module is in any of the low power modes. See Section 27.4.8.6, Freeze mode, for more information. 0 No Freeze mode request. 1 Enters Freeze mode if the FRZ bit is set.
NOT_RDY	FlexCAN Not Ready This bit indicates that the FlexCAN module is either in Disable mode, Stop mode, or Freeze mode. It is cleared once the FlexCAN module has exited these modes. 0 FlexCAN module is either in Normal mode, Listen-Only mode, or Loopback mode. 1 FlexCAN module is either in Disable mode, Stop mode, or Freeze mode.
WAK_MSK	Wake-up Interrupt Mask This bit enables the wake-up interrupt generation. 0 Wake-up interrupt is disabled. 1 Wake-up interrupt is enabled.

Table 27-11. MCR field descriptions (continued)

Field	Description
SOFT_RST	Soft Reset When this bit is set, the FlexCAN module resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IFLAG1. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: • CTRL • RXIMR0–RXIMR31 • RXGMASK, RX14MASK, RX15MASK • All message buffers The SOFT_RST bit can be set directly by the CPU when it writes to the MCR register, but it is also set when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains set while reset is pending, and is automatically cleared when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 0 No reset request. 1 Resets the registers marked as “affected by soft reset” in Table 27-4.
FRZ_ACK	Freeze mode Acknowledge This read-only bit indicates that the FlexCAN module is in Freeze mode and its prescaler is stopped. The Freeze mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when the FlexCAN module has actually entered Freeze mode. If the Freeze mode request is negated, then this bit is cleared once the FlexCAN prescaler is running again. If Freeze mode is requested while the FlexCAN module is in any of the low power modes, then the FRZ_ACK bit is set only when the low power mode is exited. See Section 27.4.8.6, Freeze mode, for more information. 0 FlexCAN not in Freeze mode, prescaler running. 1 FlexCAN in Freeze mode, prescaler stopped.
SUPV	Supervisor mode This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of Table 27-4. The reset value of this bit is 1, so the affected registers start with Supervisor access restrictions. 0 Affected registers are in Unrestricted memory space. 1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location. **Note:** This bit must be written in Freeze mode only.
SLF_WAK	Self Wake-up This bit enables the self wake-up feature when the FlexCAN module is in Stop mode. If this bit had been set by the time the FlexCAN module entered Stop mode, then the FlexCAN module looks for a recessive to dominant transition on the bus during these modes. If a transition from recessive to dominant is detected, the FlexCAN module resumes its clocks and, if enabled to do so, generates a wake-up interrupt to the CPU. If a transition from recessive to dominant is detected during Stop mode, then the FlexCAN module generates, if enabled to do so, a wake-up interrupt to the CPU so that it can resume the clocks globally. This bit cannot be written while the module is in Stop mode. 0 FlexCAN self wake-up feature disabled. 1 FlexCAN self wake-up feature enabled.

Table 27-11. MCR field descriptions (continued)

Field	Description
WRN_EN	Warning Interrupt Enable When set, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the ESR register. If WRN_EN is cleared, the TWRN_INT and RWRN_INT flags always are zero, independent of the values of the error counters, and no warning interrupt is ever generated. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transitions from < 96 to ≥ 96. **Note:** This bit must be written in Freeze mode only.
LPM_ACK	Low Power mode Acknowledge This bit indicates that the FlexCAN module is either in Disable mode or Stop mode. Either of these low power modes cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when the FlexCAN module has actually entered low power mode. See Section 27.4.8.7, Module disable mode, and Section 27.4.8.8, Stop mode, for more information. 0 FlexCAN not in any of the low power modes. 1 FlexCAN is either in Disable mode or Stop mode.
SRX_DIS	Self Reception Disable This bit defines whether the FlexCAN module is allowed to receive frames transmitted by itself. If this bit is set, frames transmitted by the module are not stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal is generated due to the frame reception. 0 Self reception enabled. 1 Self reception disabled. **Note:** This bit must be written in Freeze mode only.
BCC	Backwards Compatibility Configuration This bit is provided to support backwards compatibility with previous FlexCAN versions. When this bit is cleared, the following configuration is applied: • For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, the FlexCAN module uses its previous masking scheme with RXGMASK, RX14MASK, and RX15MASK. • The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, the FlexCAN module does not look for another matching MB. It overrides this MB with the new message and sets the CODE field to 0110 (overrun). Upon reset this bit is cleared, allowing legacy software to work without modification. 0 Individual Rx masking and queue feature are disabled. 1 Individual Rx masking and queue feature are enabled.
LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It extends the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. 0 Local Priority disabled. 1 Local Priority enabled. **Note:** This bit must be written in Freeze mode only.

Table 27-11. MCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>AEN</td><td>Abort Enable
This bit is supplied for backwards compatibility reasons. When set, it enables the Tx abort feature. This feature enables clean aborting of a pending transmission, as no frame is sent via the CAN bus without notification.
0 Abort disabled.
1 Abort enabled.</td></tr>
<tr><td>IDAM</td><td>ID Acceptance mode
This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in the following table. Note that all elements of the table are configured at the same time by this field (they are all the same format). See Section 27.3.3, Rx FIFO structure.
<table>
<tr><th>IDAM</th><th>Format</th><th>Explanation</th></tr>
<tr><td>0b00</td><td>A</td><td>One full ID (standard or extended) per filter element.</td></tr>
<tr><td>0b01</td><td>B</td><td>Two full standard IDs or two partial 14-bit extended IDs per filter element.</td></tr>
<tr><td>0b10</td><td>C</td><td>Four partial 8-bit IDs (standard or extended) per filter element.</td></tr>
<tr><td>0b11</td><td>D</td><td>All frames rejected.</td></tr>
</table>
Note: This bit must be written in Freeze mode only.</td></tr>
<tr><td>MAXMB</td><td>Maximum Number of Message Buffers
This 6-bit field defines the maximum number of message buffers that take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze mode.
Maximum MBs in use = MAXMB + 1

Note: MAXMB has to be programmed with a value smaller or equal to the number of available message buffers, otherwise the FlexCAN module cannot transmit or receive frames.</td></tr>
</table>

27.3.4.2 Control Register (CTRL)

The Control Register (CTRL) is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loopback mode, Listen Only mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. This register can be accessed at any time, however some fields must be changed only during either Disable mode or Freeze mode. Find more information in the fields descriptions below.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1		PSEG2			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-6. Control Register (CTRL)

Table 27-12. CTRL field descriptions

Field	Description
PRESDIV	Prescaler Division Factor This field defines the ratio between the CPI clock frequency and the serial clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to Section 27.4.8.4, Protocol timing. **Sclock frequency = CPI clock frequency / (PRESDIV + 1).** **Note:** This bit must be written in Freeze mode only.
RJW	Resync Jump Width This field defines the maximum number of time quanta (one time quantum is equal to the Sclock period) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. **Resync Jump Width = RJW + 1.** **Note:** This bit must be written in Freeze mode only.
PSEG1	PSEG1 — Phase Segment 1 This field defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. **Phase Buffer Segment 1(PSEG1 + 1) x Time-Quanta.** **Note:** This bit must be written in Freeze mode only.
PSEG2	PSEG2 — Phase Segment 2 This field defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. **Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.** **Note:** This bit must be written in Freeze mode only.
BOFF_MSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 0 Bus Off interrupt disabled. 1 Bus Off interrupt enabled.
ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled. 1 Error interrupt enabled.

Table 27-12. CTRL field descriptions (continued)

Field	Description
CLK_SRC	FlexCAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the FlexCAN clock or the XOSC clock. The selected clock is the one fed to the prescaler to generate the serial clock (Sclock). This bit should only be changed while the module is in Disable mode. See Section 27.4.8.4, Protocol timing, for more information. 0 The FlexCAN engine clock source is the XOSC clock. 1 The FlexCAN engine clock source is the FlexCAN clock.
TWRN_MSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the ESR register. This bit has no effect if the MCR[WRN_EN] bit is cleared and it is read as 0 when WRN_EN is cleared. 0 Tx Warning Interrupt disabled. 1 Tx Warning Interrupt enabled.
RWRN_MSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the ESR[RWRN_INT] flag. This bit has no effect if the MCR[WRN_EN] bit is cleared and it is read as 0 when WRN_EN is cleared. 0 Rx Warning Interrupt disabled. 1 Rx Warning Interrupt enabled.
LPB	Loopback This bit configures the FlexCAN module to operate in Loopback mode. In this mode, the FlexCAN module performs an internal loopback that can be used for self-test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The RXD pin is ignored and the Tx CAN output goes to the recessive state (logic 1). The FlexCAN module behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, the FlexCAN module ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loopback disabled. 1 Loopback enabled. **Note:** This bit must be written in Freeze mode only.
SMP	Sampling mode This bit defines the sampling mode of FlexCAN bits at the RXD input. 0 Just one sample determines the bit value. 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and two preceding samples. A majority rule is used. **Note:** This bit must be written in Freeze mode only.
BOFF_REC	Bus Off Recovery mode This bit defines how the FlexCAN module recovers from Bus Off state. If this bit is cleared, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is set, automatic recovery from Bus Off is disabled and the module remains in Bus Off state until the bit is cleared by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been set. If the negation occurs after 128 sequences of 11 recessive bits, then the FlexCAN module resynchronizes to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be set again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was cleared when the module entered Bus Off, setting it during Bus Off will not be effective for the current Bus Off recovery. 0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B. 1 Automatic recovering from Bus Off state disabled.

Table 27-12. CTRL field descriptions (continued)

Field	Description
TSYN	Timer Sync mode This bit enables a mechanism that resets the free-running timer each time a message is received in message buffer 0. This feature provides the means to synchronize multiple FlexCAN stations with a special “SYNC” message (that is, global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer sync feature disabled. 1 Timer sync feature enabled. **Note:** This bit must be written in Freeze mode only.
LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for message buffer transmission. When set, the LPRIO_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first. 1 Lowest number buffer is transmitted first. **Note:** This bit must be written in Freeze mode only.
LOM	Listen-Only mode This bit configures the FlexCAN module to operate in Listen Only mode. In this mode, transmission is disabled, all error counters are frozen, and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station are received. If the FlexCAN module detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it were trying to acknowledge the message. 0 Listen Only mode is deactivated. 1 FlexCAN module operates in Listen Only mode. **Note:** This bit must be written in Freeze mode only.
PROPSEG	Propagation Segment This field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. **Propagation Segment Time = (PROPSEG + 1) × Time-Quanta.** **Time-Quantum = one Sclock period.** **Note:** This bit must be written in Freeze mode only.

27.3.4.3 Free Running Timer (TIMER)

The Free Running Timer (TIMER) register represents a 16-bit free-running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the

user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-7. Free Running Timer (TIMER)

Table 27-13. TIMER field descriptions

Field	Description
TIMER	Holds the value for this timer.

27.3.4.4 Rx Global Mask (RXGMASK)

The Rx Global Mask (RXGMASK) register is provided for legacy support and for low cost MCUs that do not have the individual masking per message buffer feature. For MCUs supporting individual masks per message buffer, setting the BCC bit in MCR causes the RXGMASK register to have no effect on the module operation. For MCUs not supporting individual masks per message buffer, this register is always effective.

RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

Refer to Section 27.4.7, Rx FIFO, for important details on usage of RXGMASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 27-8. Rx Global Mask register (RXGMASK)

Table 27-14. RXGMASK field description

Field	Description
MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

27.3.4.5 Rx 14 Mask (RX14MASK)

The Rx 14 Mask (RX14MASK) register is provided for legacy support and for low cost MCUs that do not have the individual masking per message buffer feature. For MCUs supporting individual masks per message buffer, setting the BCC bit in MCR causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the identifier in Message Buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to Section 27.4.7, Rx FIFO, for important details on usage of RX14MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 27-9. Rx Buffer 14 Mask register (RX14MASK)

Table 27-15. RX14MASK field description

Field	Description
MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

27.3.4.6 Rx 15 Mask (RX15MASK)

The Rx 15 Mask (RX15MASK) register is provided for legacy support and for low cost MCUs that do not have the individual masking per message buffer feature. For MCUs supporting individual masks per message buffer, setting the BCC bit in MCR causes the RX15MASK Register to have no effect on the module operation.

When the BCC bit is cleared, RX15MASK is used as the acceptance mask for the identifier in message buffer 15. When the FEN bit in MCR is set (FIFO enabled), the RXG15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register.

Refer to Section 27.4.7, Rx FIFO, for important details on usage of RX15MASK on filtering process for Rx FIFO. The contents of this register must be programmed while the module is in Freeze mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 27-10. Rx Buffer 15 Mask register (RX15MASK)

Table 27-16. RX15MASK field description

Field	Description
MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

27.3.4.7 Error Counter Register (ECR)

The Error Counter Register (ECR) has two 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (Tx_Err_Counter field) and Receive Error Counter (Rx_Err_Counter field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only except in Freeze mode, where they can be written by the CPU.

Writing to the ECR while in Freeze mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

The FlexCAN module responds to any bus state as described in the protocol; for example, it transmits an ‘Error Active’ or ‘Error Passive’ flag, delays its transmission start time (‘Error Passive’), and has no effect on the bus when in the ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the ESR[FLT_CONF] field is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the ESR[FLT_CONF] field is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the ESR[FLT_CONF] field is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to zero.
- If the FlexCAN module is in ‘Bus Off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the ESR[FLT_CONF] field is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.
- If during system start-up only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ESR[ACK_ERR] bit). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.

- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume the ‘Error Active’ state.

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Rx_Err_Counter								Tx_Err_Counter							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-11. Error Counter Register (ECR)

27.3.4.8 Error and Status Register (ESR)

The Error and Status Register (ESR) reflects various error conditions, some general status of the device, and is the source of eight interrupts to the CPU. The reported error conditions (bits 16-21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16-23. Bits 22-28 are status bits.

Most bits in this register are read-only, except TWRN_INT, RWRN_INT, BOFF_INT, ERR_INT, and WAK_INT, which are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect). See Section 27.4.9, Interrupts, for more details.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W															w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	WAK_INT
W														w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-12. Error and Status Register (ESR)

Table 27-17. ESR field descriptions

Field	Description
TWRN_INT	Tx Warning Interrupt Flag If the WRN_EN bit in MCR is set, the TWRN_INT bit is set when the TX_WRN flag transitions from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the CTRL register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Tx error counter transitioned from < 96 to ≥ 96.
RWRN_INT	Rx Warning Interrupt Flag If the WRN_EN bit in MCR is set, the RWRN_INT bit is set when the RX_WRN flag transition from 0 to 1, meaning that the Rx error counters reached 96. If the corresponding mask bit in the CTRL register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Rx error counter transitioned from < 96 to ≥ 96.
BIT1_ERR	Bit1 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as recessive is received as dominant. **Note:** This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
BIT0_ERR	Bit0 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as dominant is received as recessive.
ACK_ERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, that is, a dominant bit has not been detected during the ACK SLOT. 0 No such occurrence. 1 An ACK error occurred since last read of this register
CRC_ERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, that is, the calculated CRC is different from the received. 0 No such occurrence. 1 A CRC error occurred since last read of this register.
FRM_ERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, that is, a fixed-form bit field contains at least one illegal bit. 0 No such occurrence. 1 A Form Error occurred since last read of this register.
STF_ERR	Stuffing Error This bit indicates that a Stuffing Error has been detected. 0 No such occurrence. 1 A Stuffing Error occurred since last read of this register.

Table 27-17. ESR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>TX_WRN</td><td>TX Error Warning
This bit indicates when repetitive errors are occurring during message transmission.
0 No such occurrence.
1 TX_Err_Counter ≥ 96.</td></tr>
<tr><td>RX_WRN</td><td>Rx Error Warning
This bit indicates when repetitive errors are occurring during message reception.
0 No such occurrence.
1 Rx_Err_Counter ≥ 96.</td></tr>
<tr><td>IDLE</td><td>CAN bus IDLE state
This bit indicates when CAN bus is in IDLE state.
0 No such occurrence.
1 CAN bus is now IDLE.</td></tr>
<tr><td>TXRX</td><td>Current FlexCAN status (transmitting/receiving)
This bit indicates if the FlexCAN module is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is set.
0 FlexCAN is receiving a message (IDLE = 0).
1 FlexCAN is transmitting a message (IDLE = 0).</td></tr>
<tr><td>FLT_CONF</td><td>Fault Confinement State
This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in the following table. If the CTRL[LOM] bit is set, the FLT_CONF field indicates “Error Passive”. Since the CTRL register is not affected by soft reset, the FLT_CONF field is not affected by soft reset if the LOM bit is set.
<table>
<tr><th>FLT_CONF Field</th><th>Meaning</th></tr>
<tr><td>0b00</td><td>Error Active</td></tr>
<tr><td>0b01</td><td>Error Passive</td></tr>
<tr><td>0b10</td><td>Bus Off</td></tr>
<tr><td>0b11</td><td>Bus Off</td></tr>
</table></td></tr>
<tr><td>BOFF_INT</td><td>Bus Off Interrupt
This bit is set when the FlexCAN module enters ‘Bus Off’ state. If the corresponding mask bit in the CTRL register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.
0 No such occurrence.
1 FlexCAN module entered ‘Bus Off’ state.</td></tr>
<tr><td>ERR_INT</td><td>Error Interrupt
This bit indicates that at least one of the Error Bits (bits 16–21) is set. If the corresponding mask bit in the CTRL register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.
0 No such occurrence.
1 Indicates setting of any Error bit in the ESR register.</td></tr>
</table>

Table 27-17. ESR field descriptions (continued)

Field	Description
WAK_INT	Wake-up Interrupt When the FlexCAN module is in Stop mode and a recessive to dominant transition is detected on the CAN bus, and if the WAK_MSK bit in the MCR is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 Indicates a recessive to dominant transition received on the CAN bus when the FlexCAN module is in Stop mode.

27.3.4.9 Interrupt Masks 1 Register (IMASK1)

The Interrupt Masks 1 (IMASK1) register allows enabling or disabling any amount of a range of 32 message buffer interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (in other words, when the corresponding IFLAG1 bit is set).

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-13. Interrupt Masks 1 Register (IMASK1)

Table 27-18. IMASK1 field descriptions

Field	Description
BUF31M – BUF0M	BUF31M–BUF0M — Buffer MB_i Mask Each bit enables or disables the respective FlexCAN message buffer (MB0 to MB31) interrupt. 0 The corresponding buffer interrupt is disabled. 1 The corresponding buffer interrupt is enabled. **Note:** Setting or clearing a bit in the IMASK1 register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

27.3.4.10 Interrupt Flags 1 Register (IFLAG1)

The Interrupt Flags 1 (IFLAG1) register defines the flags for 32 message buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt is generated. The interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the MCR[FEN] bit is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I–BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I, and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-14. Interrupt Flags 1 Register (IFLAG1)

Table 27-19. IFLAG1 field descriptions

Field	Description
BUF31I–BUF8I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN message buffer (MB8 to MB31) interrupt. 0 No such occurrence. 1 The corresponding message buffer has successfully completed transmission or reception.
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence. 1 MB7 completed transmission/reception or FIFO overflow.
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 4 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence. 1 MB6 completed transmission/reception or FIFO almost full.
BUF5I	Buffer MB5 Interrupt or “frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence. 1 MB5 completed transmission/reception or frames available in the FIFO.
BUF4I–BUF0I	Buffer MB_i Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence. 1 Corresponding MB completed transmission/reception.

27.3.4.11 Rx Individual Mask Registers (RXIMR0–RXIMR31)

The Rx Individual Mask Registers (RXIMR0–RXIMR31) are used as acceptance masks for ID filtering in Rx message buffers and the FIFO. If the FIFO is not enabled, one mask register is provided for each available message buffer, providing ID masking capability on a per message buffer basis. When the FIFO is enabled (the MCR[FEN] bit is set), the first 8 mask registers apply to the 8 elements of the FIFO filter table on a one-to-one correspondence, while the rest of the registers apply to the regular message buffers, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze mode. Out of Freeze mode, write accesses are blocked and read accesses return all zeros. Furthermore, if the MCR[BCC] bit is cleared, any read or write operation to these registers results in access error.

Address: See Table 27-21 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-15. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 27-20. RXIMR0–RXIMR63 field descriptions

Field	Description
MI31–MI0	Mask Bits For normal Rx message buffers, the mask bits affect the ID filter programmed on the message buffer. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Table 27-21. RXIMR0–RXIMR31 addresses

Address	Register	Address	Register
Base + 0x0880	RXIMR0	Base + 0x08C0	RXIMR16
Base + 0x0884	RXIMR1	Base + 0x08C4	RXIMR17
Base + 0x0888	RXIMR2	Base + 0x08C8	RXIMR18
Base + 0x088C	RXIMR3	Base + 0x08CC	RXIMR19
Base + 0x0890	RXIMR4	Base + 0x08D0	RXIMR20
Base + 0x0894	RXIMR5	Base + 0x08D4	RXIMR21
Base + 0x0898	RXIMR6	Base + 0x08D8	RXIMR22

Table 27-21. RXIMR0–RXIMR31 addresses (continued)

Address	Register	Address	Register
Base + 0x089C	RXIMR7	Base + 0x08DC	RXIMR23
Base + 0x08A0	RXIMR8	Base + 0x08E0	RXIMR24
Base + 0x08A4	RXIMR9	Base + 0x08E4	RXIMR25
Base + 0x08A8	RXIMR10	Base + 0x08E8	RXIMR26
Base + 0x08AC	RXIMR11	Base + 0x08EC	RXIMR27
Base + 0x08B0	RXIMR12	Base + 0x08F0	RXIMR28
Base + 0x08B4	RXIMR13	Base + 0x08F4	RXIMR29
Base + 0x08B8	RXIMR14	Base + 0x08F8	RXIMR30
Base + 0x08BC	RXIMR15	Base + 0x08FC	RXIMR31

27.4 Functional description

27.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed of a set of 32 message buffers (MB) that store configuration and control data, time stamp, message ID, and data (see Section 27.3.2, Message buffer structure). The memory corresponding to the first 8 message buffers can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into message buffers that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the message buffer with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of message buffers to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the message buffer ordering.

Before proceeding with the functional description, an important concept must be explained. A message buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx message buffer with a 0000 code is inactive (refer to Table 27-7). Similarly, a Tx message buffer with a 1000 or 1001 code is also inactive (refer to Table 27-8). An message buffer not programmed with 0000, 1000 or 1001 are temporarily deactivated (they do not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that message buffer (see Section 27.4.6.2, Message buffer deactivation).

27.4.2 Transmit process

In order to transmit a CAN frame, the CPU must prepare a message buffer for transmission by executing the following procedure:

1. If the message buffer is active (transmission pending), write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see Section 27.4.6.1, Transmission abort mechanism). If backwards compatibility is desired (MCR[AEN] is cleared), write 1000 to the Code field to inactivate the message buffer. However, the pending frame may be transmitted without notification (see Section 27.4.6.2, Message buffer deactivation).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control, and Code fields of the Control and Status word to activate the message buffer.

Once the message buffer is activated in the fourth step, it participates in the arbitration process and eventually is transmitted according to its priority. At the end of the successful transmission, the value of the TIMER register is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding interrupt mask register bit. The new code field after transmission depends on the code that was used to activate the message buffer in step 4 (see Table 27-7 and Table 27-8 in Section 27.3.2, Message buffer structure). When the Abort feature is enabled (the MCR[AEN] bit is set), after the interrupt flag is set for a message buffer configured as transmit buffer, the message buffer is blocked. Therefore, the CPU is not able to update it until the interrupt flag is cleared by the CPU. The CPU must clear the corresponding IFLAG before starting to prepare this message buffer for a new transmission or reception.

27.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole message buffer memory looking for the highest priority message to be transmitted. All message buffers programmed as transmit buffers are scanned to find the lowest ID[1] or the lowest message buffer number or the highest priority, depending on the LBUF and LPRIO_EN bits in the CTRL register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner message buffer defined in a previous arbitration was deactivated, or if there was no message buffer to transmit, but the CPU wrote to the C/S word of any message buffer after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any message buffer
- Upon leaving Freeze mode

1. Actually, if LBUF is cleared, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions in which they are transmitted in the CAN frame.

When the LBUF bit is set, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both cleared, the message buffer with the lowest ID is transmitted first. If LBUF is cleared and LPRIO_EN is set, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which message buffer should be transmitted first. Therefore, message buffers with PRIO = 000 have higher priority. If two or more message buffers have the same priority, the regular ID determines the priority of transmission. If two or more message buffers have the same priority (3 extra bits) and the same regular ID, the lowest message buffer is transmitted first.

Once the highest priority message buffer is selected, it is transferred to a temporary storage space called the serial message buffer (SMB), which has the same structure as a normal message buffer but is not user-accessible. This operation is called “move-out” and after it is done, write access to the corresponding message buffer is blocked (if the MCR[AEN] bit is set). The write access is released in the following events:

- After the message buffer is transmitted
- The FlexCAN module enters in HALT or BUS OFF
- The FlexCAN module loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. The FlexCAN module transmits up to 8 data bytes, even if the DLC (Data Length Code) value is bigger.

27.4.4 Receive process

To be able to receive CAN frames into the mailbox message buffers, the CPU must prepare one or more message buffers for reception by executing the following steps:

1. If the message buffer has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see Section 27.4.6.1, Transmission abort mechanism). If backwards compatibility is desired (MCR[AEN] is cleared), write ‘1000’ to the Code field to inactivate the message buffer. However, then the pending frame may be transmitted without notification (see Section 27.4.6.2, Message buffer deactivation). If the message buffer is already programmed as a receiver, write ‘0000’ to the Code field of the Control and Status word to keep the message buffer inactive.
2. Write the ID word.
3. Write ‘0100’ to the Code field of the Control and Status word to activate the message buffer.

Once the message buffer is activated in the third step, it can receive frames that match the programmed ID. At the end of a successful reception, the message buffer is updated by the MBM as follows:

1. The value of the TIMER register is written into the Time Stamp field.
2. The received ID, Data (8 bytes at most), and Length fields are stored.
3. The Code field in the Control and Status word is updated (see Table 27-7 and Table 27-8 in Section 27.3.2, Message buffer structure).

4. A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding interrupt mask register bit.

Upon receiving the message buffer interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer).
2. Read the ID field (optional – needed only if a mask was used).
3. Read the Data field.
4. Read the TIMER register (optional – releases the internal lock).

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the message buffer until this bit is cleared. Reading the TIMER register is not mandatory. If not executed the message buffer remains locked, unless the CPU reads the C/S word of another message buffer. Note that only a single message buffer is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to identify data coherency (see Section 27.4.6, Data coherence).

The CPU should synchronize to frame reception by the status flag bit for the specific message buffer in one of the IFLAG registers and not by the Code field of that message buffer. Polling the Code field does not work because once a frame is received and the CPU services the message buffer (by reading the C/S word followed by unlocking the message buffer), the Code field does not return to EMPTY. It remains FULL, as explained in Table 27-7. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the message buffer, the message buffer is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that message buffer may be lost. In summary: *never do polling by reading directly the C/S word of the message buffers. Instead, read the IFLAG registers.*

Note that the received ID field is always stored in the matching message buffer; thus the contents of the ID field in a message buffer may change if the match was due to masking. Note also that the FlexCAN module does receive frames transmitted by itself if there exists an Rx matching message buffer, provided the SRX_DIS bit in the MCR is not set. If SRX_DIS is set, the FlexCAN module cannot store frames transmitted by itself in any message buffer, even if it contains a matching message buffer, and no interrupt flag or interrupt signal is generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze mode (see Section 27.4.7, Rx FIFO). Upon receiving the “frames available” interrupt from the FIFO, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

27.4.5 Matching process

The matching process is an algorithm executed by the MBM that scans the message buffer memory looking for Rx message buffers programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other message buffers are scanned. If the FIFO is full, the matching algorithm always looks for a matching message buffer outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary message buffer called serial message buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular message buffers, the contents of the SMB are transferred to the FIFO or to the matched message buffer during the 6th bit of the end-of-frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox message buffers, a message buffer is said to be “free to receive” a new frame if the following conditions are satisfied:

- The message buffer is not locked (see Section 27.4.6.3, Message buffer lock mechanism)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the message buffer (read the C/S word and then unlocked the message buffer)

If the first message buffer with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free message buffer until it finds one. If it cannot find one that is free, then it overwrites the last matching message buffer (unless it is locked) and sets the Code field to OVERRUN (refer to Table 27-7 and Table 27-8). If the last matching message buffer is locked, then the new message remains in the SMB, waiting for the message buffer to be unlocked (see Section 27.4.6.3, Message buffer lock mechanism).

Suppose, for example, that the FIFO is disabled and there are two message buffers with the same ID, and the FlexCAN module starts receiving messages with that ID. Let us say that these message buffers are the second and the fifth in the array. When the first message arrives, the matching algorithm finds the first match in message buffer number 2. The code of this message buffer is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm finds message buffer number 2 again, but it is not “free to receive”, so it keeps looking and finds message buffer number 5, and stores the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching message buffers that are “free to receive”, so it decides to overwrite the last matched message buffer, which is number 5. In doing so, it sets the Code field of the message buffer to indicate OVERRUN.

The ability to match the same ID in more than one message buffer can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the message buffers. By programming more than one message buffer with the same ID, received messages are queued into the message buffers. The CPU can examine the Time Stamp field of the message buffers to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the MCR[BCC] bit is cleared, the matching algorithm stops at the first message buffer with a matching ID that it finds, whether this message buffer is free or not. As a result, the message queueing feature does not work if the BCC bit is cleared.

Matching to a range of IDs is possible by using ID acceptance masks. The FlexCAN module supports individual masking per message buffer. Please refer to Section 27.3.4.11, Rx Individual Mask Registers (RXIMR0–RXIMR31). During the matching algorithm, if a mask bit is set, then the corresponding ID bit is compared. If the mask bit is cleared, the corresponding ID bit is a “don’t care” bit. Note that the individual mask registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is set and while the module is in Freeze mode.

The FlexCAN module also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK, and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the MCR[BCC] bit is cleared.

27.4.6 Data coherence

In order to maintain data coherency and proper FlexCAN operation, the CPU must obey the rules described in Section 27.4.2, Transmit process, and Section 27.4.4, Receive process. Any form of CPU accessing a message buffer structure within the FlexCAN module other than those specified may cause the FlexCAN module to behave in an unpredictable way.

27.4.6.1 Transmission abort mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by setting the MCR[AEN] bit.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active message buffers configured as transmission must be aborted first and then they may be updated. If the abort code is written to a message buffer that is currently being transmitted, or to a message buffer that was already loaded into the SMB for transmission, the write operation is blocked and the message buffer is not deactivated, but the abort request is captured and kept pending until one of the following conditions is satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze mode

If none of the conditions above are reached, the message buffer is transmitted correctly, the interrupt flag is set in the IFLAG register, and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. On the other hand, if one of the above conditions is reached, the frame is not transmitted; therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG, and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked; therefore the message buffer is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active message buffer was deactivated. Although the AEN bit is set and the CPU wrote the abort code, in this case the message buffer is deactivated and not aborted, because the transmission did not start yet. One message buffer is only aborted when the abort request is captured and kept pending until one of the previous conditions is satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or if it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the message buffer was aborted (CODE=1001) or it was transmitted (CODE=1000).

NOTE

An abort request to a TxMB can block any write operation into its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a transmission by winning the CAN bus arbitration.

27.4.6.2 Message buffer deactivation

Deactivation is a mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active message buffers out of Freeze mode. Any CPU write access to the Control and Status word of a message buffer causes that message buffer to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the message buffers to decide which message buffer to transmit or receive. If the CPU updates the message buffer in the middle of a match or arbitration process, the data of that message buffer may no longer be coherent; therefore deactivation of that message buffer is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active message buffers when not in Freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If message buffers are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx message buffer with a matching ID is deactivated during the matching process after it was scanned, then this message buffer is marked as invalid to receive the frame, and the FlexCAN module keeps looking for another matching message buffer within the ones it has not scanned yet. If it cannot find one, then the message is lost. Suppose, for example, that two message buffers have a matching ID to a received frame, and the user deactivated the first matching message buffer after the FlexCAN module has scanned the second. The received frame is lost even if the second matching message buffer was “free to receive”.
- If a Tx message buffer containing the lowest ID is deactivated after the FlexCAN module has scanned it, then the FlexCAN module looks for another match within the message buffers that it has not scanned yet. Therefore, it may transmit a message buffer with an ID that may not be the lowest at the time, because a lower ID might be present in one of the message buffers that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx message buffer causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued, and the

Code field is not updated. In order to avoid this situation, the abort procedures described in Section 27.4.6.1, Transmission abort mechanism, should be used.

27.4.6.3 Message buffer lock mechanism

Besides message buffer deactivation, the FlexCAN module has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx message buffer, the FlexCAN module assumes that the CPU wants to read the whole message buffer in an atomic operation, and thus it sets an internal lock flag for that message buffer. The lock is released when the CPU reads the TIMER register (global unlock operation), or when it reads the Control and Status word of another message buffer. The message buffer locking is done to prevent a new frame from being written into the message buffer while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx message buffers, which have a code different than INACTIVE (‘0000’) or EMPTY[1] (‘0100’). Also, Tx message buffers cannot be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth message buffers of the array are programmed with the same ID, and the FlexCAN module has already received and stored messages into these two message buffers. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this message buffer is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” message buffers, so it decides to override MB number 5. However, this message buffer is locked, so the new message cannot be written there. It remains in the SMB waiting for the message buffer to be unlocked, and only then will be written to the message buffer. If the message buffer is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB with no indication of lost messages either in the Code field of the message buffer or in the ESR register.

While the message is being moved in from the SMB to the message buffer, the BUSY bit on the Code field is set. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the message buffer until the BUSY bit is cleared.

NOTE

If the BUSY bit is set or if the message buffer is empty, then reading the Control and Status word does not lock the message buffer.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is cleared and the message buffer is marked as invalid for the current matching round. Any pending message on the SMB is not transferred to the message buffer.

1. In previous FlexCAN versions, reading the C/S word locked the message buffer even if it was EMPTY. This behavior is honored when the BCC bit is cleared.

27.4.7 Rx FIFO

The receive-only FIFO is enabled by setting the MCR[FEN] bit. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 message buffers (0x80–0xFF) is now reserved for use of the FIFO engine (see Section 27.3.3, Rx FIFO structure). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a message buffer structure at the beginning of the memory.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing a message buffer in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the message buffer in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 5 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also Section 27.3.3, Rx FIFO structure):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

NOTE

A chosen format is applied to all eight registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight individual mask registers (RXIMR0–RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIMR8, continues to affect the regular message buffers, starting from MB8. If the BCC bit is cleared (or if the RXIMR registers are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK, and the other elements (0 to 5) are affected by RXGMASK.

27.4.7.1 Precautions when using Global Mask and Individual Mask registers

The user must account for aligning the position of the ID field in the Rx message buffers and in the RXIDA, RXIDB, and RXIDC fields of the ID Tables. (Please see Section 27.3.3, Rx FIFO structure, and Figure 27-4.) Mask filtering alignment is affected based on the setting of the FEN and BCC bits of MCR. Table 27-22 shows recommended actions depending on FEN and BCC settings.

Table 27-22. Recommended FEN and BCC settings

Case	MCR[FEN] RxFIFO	MCR[BCC] Rx Individual Mask	Notes
Case 1	FEN = 0	BCC = 0	RXGMASK, RX14MASK, and RX15MASK can safely be used. This allows backwards compatibility to older devices (for example, devices without the individual masks feature). In this case, individual masks are not used.
Case 2	FEN = 1	BCC = 0	1st alternative: Do not use RXGMASK, RX14MASK, and RX15MASK. In this case, leave the masks in their reset state.
Case 3	FEN = 1	BCC = 0	2nd alternative: Do not configure any message buffer as Rx (in other words, set all message buffers as either Tx or inactive). In this case, RXGMASK, RX14MASK, and RX15MASK can be used to affect ID Tables without affecting the filtering process for Rx message buffers.
Case 4	don't care	BCC = 1	If MCR[BCC] = 1, then the RXIMRs are enabled. Thus, RXGMASK, RX14MASK, and RX15MASK are not used. Particularly, when MCR[FEN] = 0, RxFIFO is disabled, then RXGMASK, RX14MASK, and RX15MASK do not affect filtering. Individual masks are used.

27.4.8 CAN protocol related features

27.4.8.1 Remote frames

A remote frame is a special kind of frame. The user can program a message buffer to be a request remote frame by writing the message buffer as transmit with the RTR bit set to 1. After the remote request frame is transmitted successfully, the message buffer becomes a receive message buffer, with the same ID as before.

When a remote request frame is received by the FlexCAN module, its ID is compared to the IDs of the transmit message buffers with the Code field '1010'. If there is a matching ID, then this message buffer frame is transmitted. Note that if the matching message buffer has the RTR bit set, then the FlexCAN module transmits a remote frame as a response.

A received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a remote request frame was received and matched a message buffer, this message buffer immediately enters the internal arbitration process, but is considered as a normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), the FlexCAN module does not generate an automatic response for remote request frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it is stored in the FIFO and presented to the CPU. Note that for filtering formats A and

B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

27.4.8.2 Overload frames

The FlexCAN module transmits overload frames due to detection of the following conditions on the CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of end of frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of error frame delimiter or overload frame delimiter

27.4.8.3 Time stamp

The value of the TIMER register is sampled at the beginning of the identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the TIMER register can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in Section 27.3.4.2, Control Register (CTRL).

27.4.8.4 Protocol timing

Figure 27-16 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the CTRL register defines whether the internal clock is connected to the XOSC clock or the FlexCAN clock. The clock source should be modified only while the module is in Disable mode (the MCR[MDIS] bit is set).

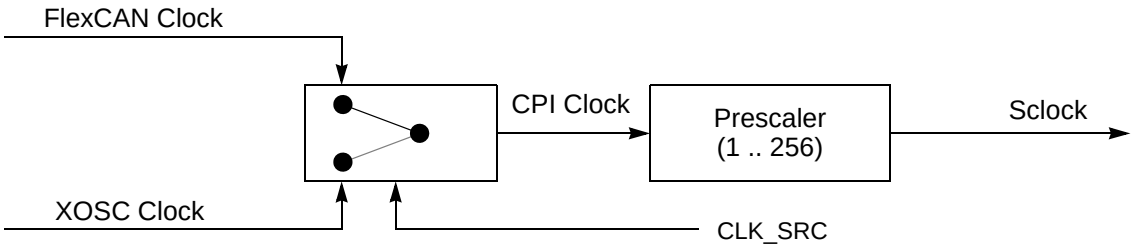


Figure 27-16. CAN engine clocking scheme

The XOSC clock should be selected whenever a tight tolerance is required in the CAN bus timing. The XOSC clock has better jitter performance than PLL generated clocks.

NOTE

If the selected CAN Protocol Interface (CPI) clock is faster than the peripheral clock, the functionality is no longer guaranteed. Please see the jitter specification in the MPC5675K Microcontroller Datasheet (MPC5675K).

The FlexCAN module supports a variety of means to set up bit timing parameters that are required by the CAN protocol. The CTRL register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2, and RJW. See Section 27.3.4.2, Control Register (CTRL).

The PRESDIV field controls a prescaler that generates the serial clock (Sclock), whose period defines the 'time quantum' used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

Eqn. 27-1

$$f_{Tq} = \frac{f_{CANCLK}}{(\text{Prescaler value})}$$

A bit time is subdivided into three segments[1] (reference Figure 27-17 and Table 27-23):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL register (plus 1) to be 2 to 8 time quanta long.

Eqn. 27-2

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})}$$

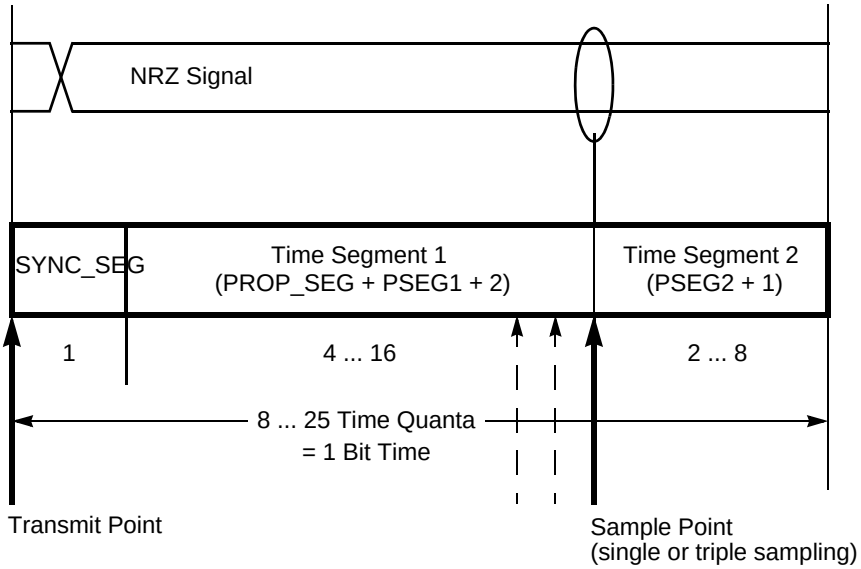


Figure 27-17. Segments within the Bit Time

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Refer also to the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 27-23. Time segment syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample. The sample point is based on the sum of PROP_SEG and PSEG1 + 2.

Table 27-24 gives an overview of the Bosch CAN 2.0B standard compliant segment settings and the related parameter values.

Table 27-24. Bosch CAN 2.0B standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

27.4.8.5 Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 27-18.

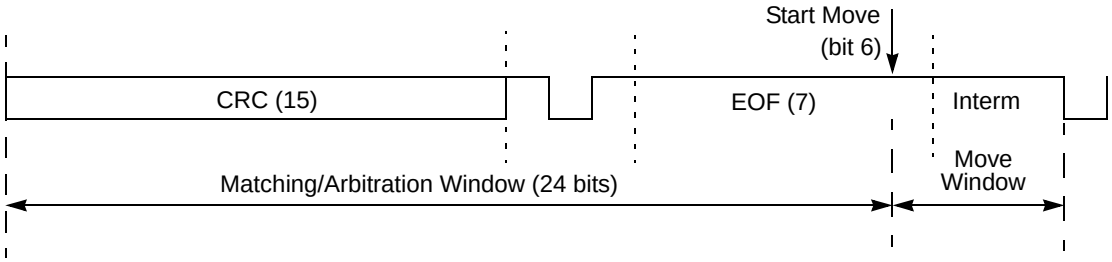


Figure 27-18. Arbitration, match, and move time windows

When doing matching and arbitration, the FlexCAN module needs to scan the whole message buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- The peripheral clock frequency cannot be smaller than the oscillator clock frequency, in other words, the PLL cannot be programmed to divide down the oscillator clock.

- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in Table 27-25.

Table 27-25. Minimum ratio between peripheral clock frequency and CAN bit rate

Number of message buffers	Minimum ratio
16	8
32	8

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in Table 27-25 can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, or PSEG2).

27.4.8.6 Freeze mode

This mode is entered by setting the MCR[HALT] bit, or when the MCU is put into debug mode. In both cases it is also necessary that the MCR[FRZ] bit is set and the module is not in any of the low power modes (Disable or Stop). When freeze mode is requested during transmission or reception, the FlexCAN module does the following:

- Waits to be in either Intermission, Passive Error, Bus Off, or Idle state
- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores the RXD pin and drives the TXD pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in MCR

After requesting freeze mode, the user must wait for the MCR[FRZ_ACK] bit to be set before executing any other action; otherwise the FlexCAN module may operate in an unpredictable way. In freeze mode, all memory-mapped registers are accessible.

Exiting freeze mode is done in one of the following ways:

- CPU clears the FRZ bit in the MCR.
- The MCU is removed from Debug mode and/or the HALT bit is cleared.

Once out of freeze mode, the FlexCAN module tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

27.4.8.7 Module disable mode

This low power mode is entered when the MDIS bit in the MCR Register is asserted. If the module is disabled during Freeze Mode, it requests to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, the FlexCAN module does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive

- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores its RXD pin and drives its TXD pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The bus interface unit continues to operate, enabling the CPU to access memory-mapped registers, except the TIMER register, the ECR register, and the message buffers, which cannot be accessed when the module is in disable mode. Exiting from this mode is done by clearing the MDIS bit, which resumes the clocks, and clearing the LPM_ACK bit.

27.4.8.8 Stop mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If the FlexCAN module receives the global stop mode request during freeze mode, it sets the LPM_ACK bit, clears the FRZ_ACK bit, and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop mode is requested during transmission or reception, the FlexCAN module does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores its RXD pin and drives its TXD pin as recessive
- Sets the MCR[NOT_RDY] and MCR[LPM_ACK] bits
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting Stop mode is done in one of the following ways:

- CPU resumes the clocks and removes the Stop mode request
- CPU resumes the clocks and Stop mode request as a result of the self wake mechanism

In the self wake mechanism, if the SLF_WAK bit in MCR was set at the time the FlexCAN module entered stop mode, then upon detection of a recessive-to-dominant transition on the CAN bus, the FlexCAN module sets the WAK_INT bit in the ESR and, if enabled by the WAK_MSK bit in MCR, generates a wake-up interrupt to the CPU. Upon receiving the interrupt, the CPU should resume the clocks and remove the Stop mode request. The FlexCAN module then waits for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. Table 27-26 details the effect of SLF_WAK and WAK_MSK upon wake-up from Stop mode. Note that wake-up from Stop mode only works when both bits are set.

Table 27-26. Wake-up from stop mode

SLF_WAK	WAK_MSK	MCU clocks enabled	Wake-up interrupt generated
0	0	No	No
0	1	No	No
1	0	No	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the Rx CAN input line while in Stop mode. See the WAK_SRC bit in Section 27.3.4.1, Module Configuration Register (MCR). This feature can be used to protect the FlexCAN module from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

NOTE

Not all MCUs are equipped with the low-pass filter. Consult the specific MCU documentation to determine if the low-pass filter is available and to determine its electrical parameters.

27.4.9 Interrupts

The module can generate 7 interrupt sources (5 interrupts due to ORed message buffers and 2 interrupts due to ORed interrupts from Bus Off, Error, and Tx Warning, Rx Warning, and Wake-up). See Chapter 32, Interrupt Controller (INTC), for more information.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has a flag bit assigned in the IFLAG registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1 (unless another interrupt is generated at the same time).

NOTE

It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags that are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (the MCR[FEN] bit is set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the FIFO Overflow flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the “frames available in FIFO flag” and bits 4-0 are unused. See Section 27.3.4.10, Interrupt Flags 1 Register (IFLAG1), for more information.

A combined interrupt for all MBs is also generated by an OR of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case, the CPU must read the IFLAG registers to determine which message buffer caused the interrupt.

The other five interrupt sources (Bus Off, Error, Tx Warning, Rx Warning, and Wake-up) generate interrupts like the MB ones, and can be read from the ESR register. The Bus Off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in the CTRL register, and the wake-up interrupt mask bit is located in the MCR.

27.4.10 Bus interface

CPU access to the FlexCAN registers is subject to the following rules:

- Read and write access to supervisor registers in User mode results in access error.

- Read and write access to unimplemented or reserved address space also results in access error. Any access to an unimplemented message buffer or Rx individual mask register locations results in access error. Any access to the Rx individual mask register space when the MCR[BCC] bit is cleared results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx individual mask registers can only be accessed in Freeze mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose the FlexCAN module is configured with 32 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x027F. The available memory in the mask registers space would be from 0x0884 to 0x08FF.

NOTE

Unused message buffer space must not be used as general purpose RAM while the FlexCAN module is transmitting and receiving CAN frames.

27.5 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

27.5.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to Table 27-4 to see what registers are affected by soft reset)
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains set while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset cannot be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK_SRC bit) should be selected while the module is in Disable mode. After the clock source is selected and the module is enabled (MDIS bit is cleared), the FlexCAN module automatically goes to Freeze mode. In Freeze mode, the FlexCAN module is un-synchronized to the CAN bus, the HALT and FRZ bits in MCR are set, the internal state machines are disabled, and the FRZ_ACK and NOT_RDY bits in the MCR are set. The TXD pin is in recessive state and the FlexCAN module does not initiate any transmission or reception of CAN frames. Note that the message buffers and the Rx individual mask registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that the FlexCAN module be put into Freeze mode (see Section 27.4.8.6, Freeze mode). The following is a generic initialization sequence applicable to the FlexCAN module:

1. Initialize the Module Configuration Register (MCR).
   a) Enable the individual filtering per message buffer and reception queue features by setting the BCC bit.
   b) Enable the warning interrupts by setting the WRN_EN bit.
   c) If required, disable frame self reception by setting the SRX_DIS bit.
   d) Enable the FIFO by setting the FEN bit.
   e) Enable the abort mechanism by setting the AEN bit.
   f) Enable the local priority feature by setting the LPRIO_EN bit.
2. Initialize the Control Register (CTRL).
   a) Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW.
   b) Determine the bit rate by programming the PRESDIV field.
   c) Determine the internal arbitration mode (LBUF bit).
3. Initialize the message buffers.
   a) The control and status word of all message buffers must be initialized.
   b) If FIFO was enabled, the 8-entry ID table must be initialized.
   c) Other entries in each message buffer should be initialized as required.
4. Initialize the Rx Individual Mask Registers.
5. Set the required interrupt mask bits in the IMASK registers (for all message buffer interrupts), in the CTRL register (for bus off and error interrupts) and in the MCR for wake-up interrupt.
6. Clear the MCR[HALT] bit.

Starting with the last event, the FlexCAN module attempts to synchronize to the CAN bus.

27.5.2 FlexCAN addressing and RAM size configurations

These RAM configurations can be implemented within the FlexCAN module:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for individual mask registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for individual mask registers

If 64 MBs are required, two FlexCAN modules can be multiplexed as shown in Figure 27-19.

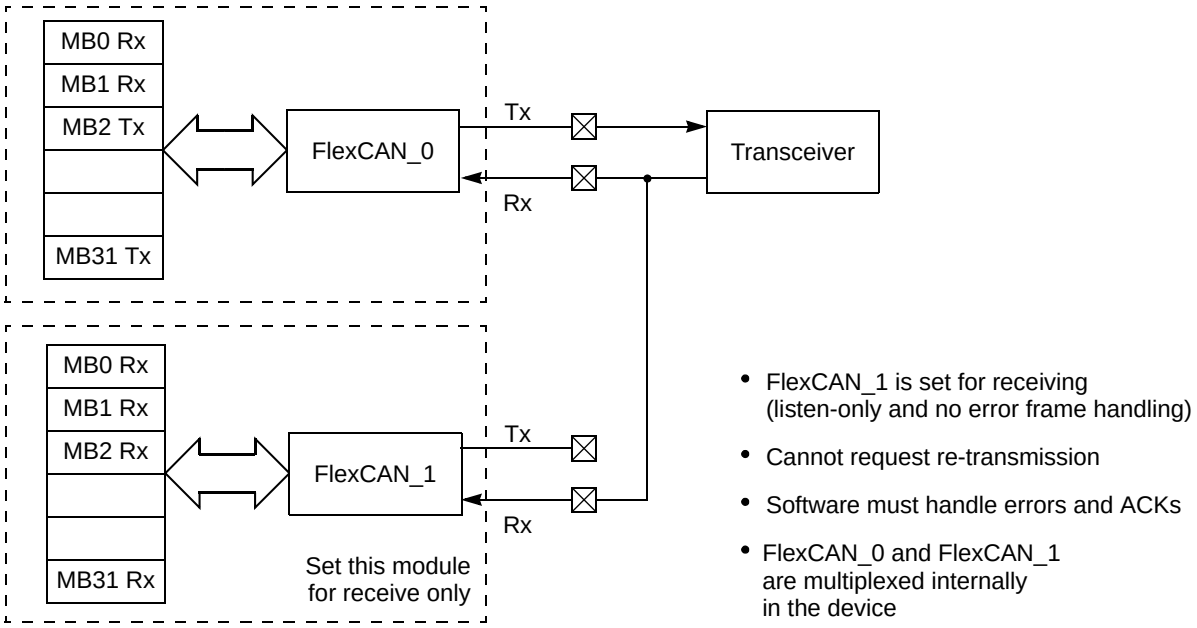


Figure 27-19. Usage of 64 MB with 32 MB + 32 MB

27.6 Programming Considerations

The following steps should be followed:

1. Clear CTRL[BOFF_MSK] bit. (Optional: Only if the Bus Off Interrupt is enabled).
2. Set MCR[SOFT_RST] bit.
3. Poll MCR[SOFT_RST] until this bit is cleared.
4. Set MCR[SOFT_RST] bit.
5. Read MCR[SOFT_RST] bit.
6. If MCR[SOFT_RST] bit is “0", return to step 4 again, else go to next step (step 7).
7. Poll MCR[SOFT_RST] until this bit is cleared.
8. Set CTRL[BOFF_MSK] bit (Optional: Only if Bus Off Interrupt is enabled).

This page is intentionally left blank.

Chapter 28
Motor Control Pulse Width Modulator Module (FlexPWM)

28.1 Introduction

The MPC5675K contains three FlexPWM modules, FlexPWM_0, FlexPWM_1, and FlexPWM_2. Each has the following features:

- Support of PWM output capabilities of third output channel per submodule (PWMX)
- 4 fault inputs for each instantiation

The FlexPWM interacts with the CTU as described in the CTU section of this document.

28.1.1 Overview

The pulse width modulator module (FlexPWM) contains four PWM submodules, each of which is set up to control a single half-bridge power stage. There are also four fault channels.

This PWM is capable of controlling most motor types: AC induction motors (ACIM), Permanent Magnet AC motors (PMAC), both brushless (BLDC) and brush (BDC) DC motors, switched (SRM) and variable reluctance motors (VRM), and stepper motors.

28.1.2 Features

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- PWM outputs can operate as complementary pairs or independent channels
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double-buffered PWM registers
  - — Integral reload rates from 1 to 16
  - — Half cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a “Force Out” event
- PWMX pin can optionally output a third PWM signal from each submodule

- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- The option to supply the source for each complementary PWM signal pair from any of the following:
  - External digital pin
  - Internal timer channel
  - External ADC input, taking into account values set in ADC high and low limit registers

28.1.3 Modes of operation

Care must be exercised when using this module in certain chip operating modes. Some motors (such as 3-phase AC motors) require regular software updates for proper operation. Failure to do so could result in destroying the motor or inverter. Because of this, PWM outputs are placed in their inactive states in STOP mode, and optionally under WAIT and debug modes. PWM outputs are reactivated (assuming they were active to begin with) when these modes are exited.

Table 28-1. Modes when PWM operation is restricted

Mode	Description
STOP	Peripheral and CPU clocks are stopped. PWM outputs are driven inactive.
WAIT	CPU clocks are stopped while peripheral clocks continue to run. PWM outputs are driven inactive as a function of the WAITEN bit.
DEBUG	CPU and peripheral clocks continue to run, but CPU may be stalled for periods of time. PWM outputs are driven inactive as a function of the DBGEN bit.

28.1.4 Block diagrams

28.1.4.1 Module level

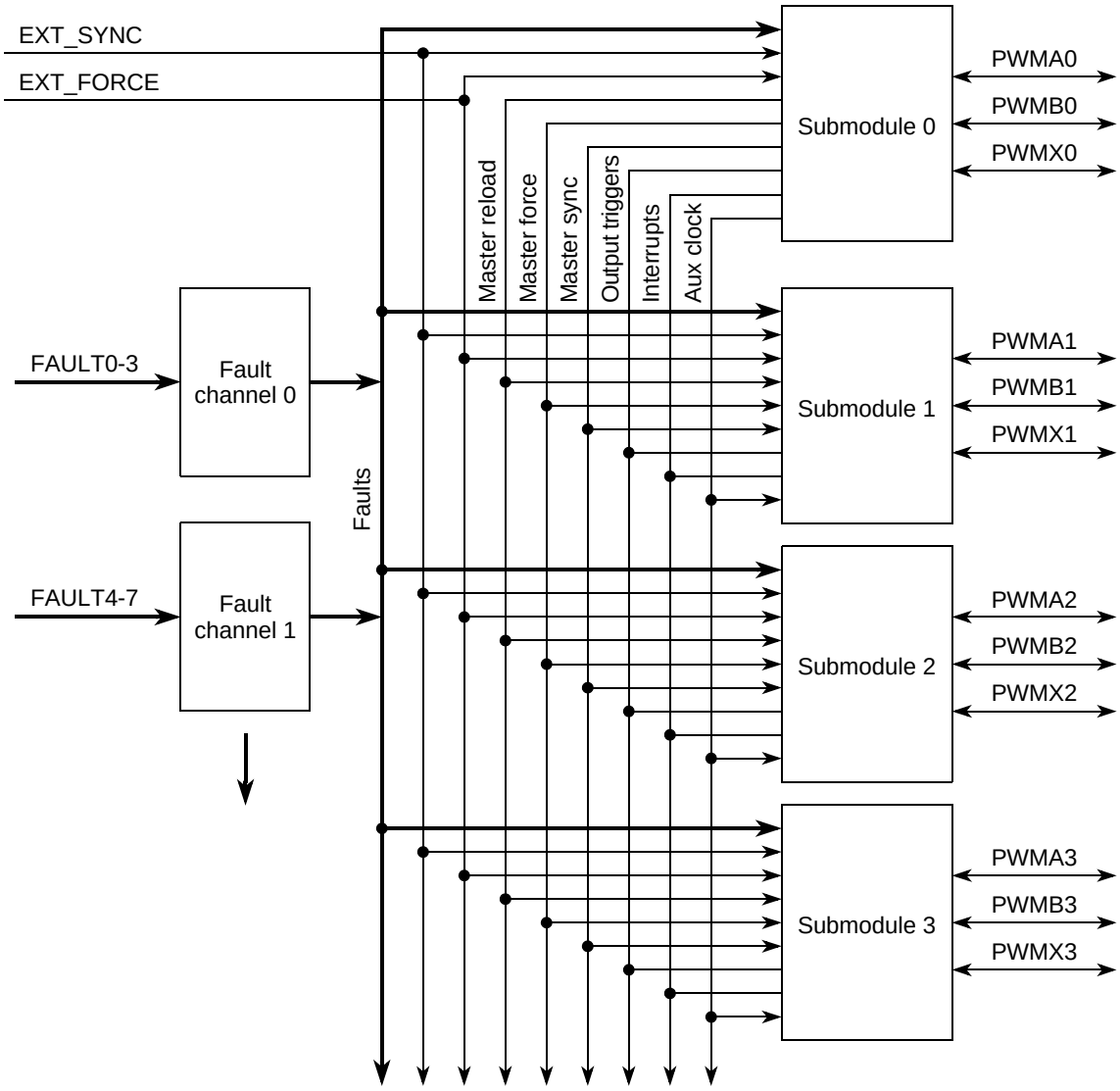


Figure 28-1. PWM block diagram

28.1.4.2 PWM submodule

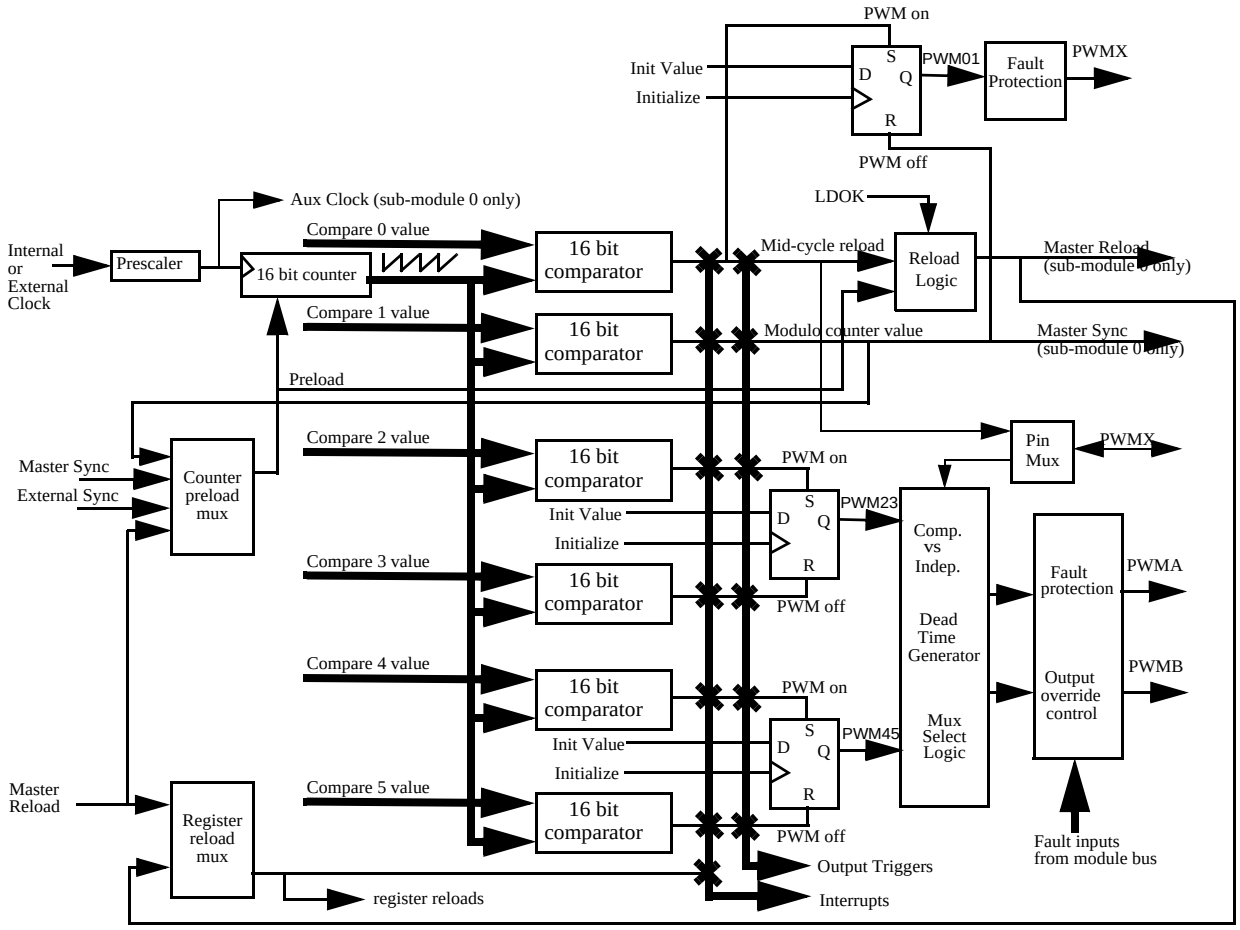


Figure 28-2. PWM submodule block diagram

28.2 External signal descriptions

The PWM module has external pins named PWMA[*n*], PWMB[*n*], PWMX[*n*], FAULT[*n*], EXT_SYNC, and EXT_FORCE. The PWM module also has an on-chip input called EXT_CLK and an on-chip output called OUT_TRIG[*n*].

28.2.1 PWMA[*n*] and PWMB[*n*]—External PWM pair

These pins are the output pins of the PWM channels. They can be independent PWM signals or a complementary pair. When not needed as an output, they can be used as inputs to the input capture circuitry.

28.2.2 PWMX[*n*]—Auxiliary PWM signal

These pins are the auxiliary output pins of the PWM channels. They can be independent PWM signals. When not needed as an output, they can be used as inputs to the input capture circuitry or they can be used to generate the MCTRL[IPOL] bit during deadtime correction.

28.2.3 FAULT[*n*]—Fault inputs

These input pins allow disabling selected PWM outputs.

28.2.4 EXT_SYNC—External Synchronization Signal

This input signal allows a source external to the PWM to initialize the PWM counter. In this manner the PWM can be synchronized to external circuitry.

28.2.5 EXT_FORCE—External Output Force Signal

This input signal allows a source external to the PWM to force an update of the PWM outputs. In this manner the PWM can be synchronized to external circuitry. An example would be to simultaneously switch all of the PWM outputs on a commutation boundary for trapezoidal control of a BLDC motor. The boundary can be established via external logic or an on-chip timer.

28.2.6 OUT_TRIG0[*n*] and OUT_TRIG1[*n*]—Output Triggers

These outputs allow the PWM submodules to control timing of the ADC conversions. See 28.3.2.15, Output Trigger Control Register (TCTRL), for a description of how to enable these outputs and how the compare registers match up to the output triggers.

28.2.7 EXT_CLK—External Clock Signal

This on-chip input signal allows an on-chip source external to the PWM (typically a timer) to control the PWM clocking. In this manner the PWM can be synchronized to the timer. This signal must be generated synchronously to the PWM's clock since it is not resynchronized in the PWM.

28.3 Memory map and registers

28.3.1 FlexPWM module memory map

Table 28-2 shows the base addresses for the FlexPWM modules on MPC5675K. Base addresses are the same in Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM). All PWM registers are 16 bits wide.

Table 28-2. FlexPWM module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM) Decoupled Parallel Mode (DPM)	FlexPWM_0	0xFFE2_4000
	FlexPWM_1	0xFFE2_8000
	FlexPWM_2	0xC3E7_4000

Table 28-3. FlexPWM memory map

Offset from FlexPWM_BASE	Register	Access	Reset value	Location
0x0000	Counter Register (CNT) for submodule 0	R	0x0000	on page 896
0x0002	Initial Count Register (INIT) for submodule 0	R/W	0x0000	on page 897
0x0004	Control 2 Register (CTRL2) for submodule 0	R/W	0x0000	on page 897
0x0006	Control 1 Register (CTRL1) for submodule 0	R/W	0x0400	on page 899
0x0008	Value Register 0 (VAL0) for submodule 0	R/W	0x0000	on page 901
0x000A	Value Register 1 (VAL1) for submodule 0	R/W	0x0000	on page 901
0x000C	Value Register 2 (VAL2) for submodule 0	R/W	0x0000	on page 902
0x000E	Value Register 3 (VAL3) for submodule 0	R/W	0x0000	on page 903
0x0010	Value Register 4 (VAL4) for submodule 0	R/W	0x0000	on page 903
0x0012	Value Register 5 (VAL5) for submodule 0	R/W	0x0000	on page 904
0x0014–0x0017	Reserved			
0x0018	Output Control Register (OCTRL) for submodule 0	R/W	0x0000	on page 904
0x001A	Status Register (STS) for submodule 0	R/W	0x0000	on page 905
0x001C	Interrupt Enable Register (INTEN) for submodule 0	R/W	0x0000	on page 906
0x001E	DMA Enable Register (DMAEN) for submodule 0	R/W	0x0000	on page 907
0x0020	Output Trigger Control Register (TCTRL) for submodule 0	R/W	0x0000	on page 908
0x0022	Fault Disable Mapping Register (DISMAP) for submodule 0	R/W	0xFFFF	on page 909
0x0024	Deadtime Count Register 0 (DTCNT0) for submodule 0	R/W	0x07FF	on page 909
0x0026	Deadtime Count Register 1 (DTCNT1) for submodule 0	R/W	0x07FF	on page 909
0x0028–0x002F	Reserved			
0x0030	Capture Control Register X (CAPTCTRLX) for submodule 0	R/W	0x0000	on page 910
0x0032	Capture Compare Register X (CAPTCOMPX) for submodule 0	R/W	0x0000	on page 912
0x0034	Capture Value 0 Register (CVAL0) for submodule 0	R	0x0000	on page 912

Table 28-3. FlexPWM memory map (continued)

Offset from FlexPWM_BASE	Register	Access	Reset value	Location
0x0036	Capture Value 0 Cycle Register (CVAL0C) for submodule 0	R	0x0000	on page 913
0x0038	Capture Value 1 Register (CVAL1) for submodule 0	R	0x0000	on page 913
0x003A	Capture Value 1 Cycle Register (CVAL1C) for submodule 0	R	0x0000	on page 913
0x003C–0x004F	Reserved			
0x0050	Counter Register (CNT) for submodule 1	R	0x0000	on page 896
0x0052	Initial Count Register (INIT) for submodule 1	R/W	0x0000	on page 897
0x0054	Control 2 Register (CTRL2) for submodule 1	R/W	0x0000	on page 897
0x0056	Control 1 Register (CTRL1) for submodule 1	R/W	0x0000	on page 899
0x0058	Value Register 0 (VAL0) for submodule 1	R/W	0x0000	on page 901
0x005A	Value Register 1 (VAL1) for submodule 1	R/W	0x0000	on page 901
0x005C	Value Register 2 (VAL2) for submodule 1	R/W	0x0000	on page 902
0x005E	Value Register 3 (VAL3) for submodule 1	R/W	0x0000	on page 903
0x0060	Value Register 4 (VAL4) for submodule 1	R/W	0x0000	on page 903
0x0062	Value Register 5 (VAL5) for submodule 1	R/W	0x0000	on page 904
0x0064–0x0067	Reserved			
0x0068	Output Control Register (OCTRL) for submodule 1	R/W	0x0000	on page 904
0x006A	Status Register (STS) for submodule 1	R/W	0x0000	on page 905
0x006C	Interrupt Enable Register (INTEN) for submodule 1	R/W	0x0000	on page 906
0x006E	DMA Enable Register (DMAEN) for submodule 1	R/W	0x0000	on page 907
0x0070	Output Trigger Control Register (TCTRL) for submodule 1	R/W	0x0000	on page 908
0x0072	Fault Disable Mapping Register (DISMAP) for submodule 1	R/W	0xFFFF	on page 909
0x0074	Deadtime Count Register 0 (DTCNT0) for submodule 1	R/W	0x07FF	on page 909
0x0076	Deadtime Count Register 1 (DTCNT1) for submodule 1	R/W	0x07FF	on page 909
0x0078–0x007F	Reserved			
0x0080	Capture Control Register X (CAPTCTRLX) for submodule 1	R/W	0x0000	on page 910
0x0082	Capture Compare Register X (CAPTCOMPX) for submodule 1	R/W	0x0000	on page 912
0x0084	Capture Value 0 Register (CVAL0) for submodule 1	R	0x0000	on page 912
0x0086	Capture Value 0 Cycle Register (CVAL0C) for submodule 1	R	0x0000	on page 913

Table 28-3. FlexPWM memory map (continued)

Offset from FlexPWM_BASE	Register	Access	Reset value	Location
0x0088	Capture Value 1 Register (CVAL1) for submodule 1	R	0x0000	on page 913
0x008A	Capture Value 1 Cycle Register (CVAL1C) for submodule 1	R	0x0000	on page 913
0x008C–0x009F	Reserved			
0x00A0	Counter Register (CNT) for submodule 2	R	0x0000	on page 896
0x00A2	Initial Count Register (INIT) for submodule 2	R/W	0x0000	on page 897
0x00A4	Control 2 Register (CTRL2) for submodule 2	R/W	0x0000	on page 897
0x00A6	Control 1 Register (CTRL1) for submodule 2	R/W	0x0000	on page 899
0x00A8	Value Register 0 (VAL0) for submodule 2	R/W	0x0000	on page 901
0x00AA	Value Register 1 (VAL1) for submodule 2	R/W	0x0000	on page 901
0x00AC	Value Register 2 (VAL2) for submodule 2	R/W	0x0000	on page 902
0x00AE	Value Register 3 (VAL3) for submodule 2	R/W	0x0000	on page 903
0x00B0	Value Register 4 (VAL4) for submodule 2	R/W	0x0000	on page 903
0x00B2	Value Register 5 (VAL5) for submodule 2	R/W	0x0000	on page 904
0x00B4–0x00B7	Reserved			
0x00B8	Output Control Register (OCTRL) for submodule 2	R/W	0x0000	on page 904
0x00BA	Status Register (STS) for submodule 2	R/W	0x0000	on page 905
0x00BC	Interrupt Enable Register (INTEN) for submodule 2	R/W	0x0000	on page 906
0x00BE	DMA Enable Register (DMAEN) for submodule 2	R/W	0x0000	on page 907
0x00C0	Output Trigger Control Register (TCTRL) for submodule 2	R/W	0x0000	on page 908
0x00C2	Fault Disable Mapping Register (DISMAP) for submodule 2	R/W	0xFFFF	on page 909
0x00C4	Deadtime Count Register 0 (DTCNT0) for submodule 2	R/W	0x07FF	on page 909
0x00C6	Deadtime Count Register 1 (DTCNT1) for submodule 2	R/W	0x07FF	on page 909
0x00C8–0x00CF	Reserved			
0x00D0	Capture Control Register X (CAPTCTRLX) for submodule 2	R/W	0x0000	on page 910
0x00D2	Capture Compare Register X (CAPTCOMPX) for submodule 2	R/W	0x0000	on page 912
0x00D4	Capture Value 0 Register (CVAL0) for submodule 2	R	0x0000	on page 912
0x00D6	Capture Value 0 Cycle Register (CVAL0C) for submodule 2	R	0x0000	on page 913
0x00D8	Capture Value 1 Register (CVAL1) for submodule 2	R	0x0000	on page 913

Table 28-3. FlexPWM memory map (continued)

Offset from FlexPWM_BASE	Register	Access	Reset value	Location
0x00DA	Capture Value 1 Cycle Register (CVAL1C) for submodule 2	R	0x0000	on page 913
0x00DC–0x00EF	Reserved			
0x00F0	Counter Register (CNT) for submodule 3	R	0x0000	on page 896
0x00F2	Initial Count Register (INIT) for submodule 3	R/W	0x0000	on page 897
0x00F4	Control 2 Register (CTRL2) for submodule 3	R/W	0x0000	on page 897
0x00F6	Control 1 Register (CTRL1) for submodule 3	R/W	0x0000	on page 899
0x00F8	Value Register 0 (VAL0) for submodule 3	R/W	0x0000	on page 901
0x00FA	Value Register 1 (VAL1) for submodule 3	R/W	0x0000	on page 901
0x00FC	Value Register 2 (VAL2) for submodule 3	R/W	0x0000	on page 902
0x00FE	Value Register 3 (VAL3) for submodule 3	R/W	0x0000	on page 903
0x0100	Value Register 4 (VAL4) for submodule 3	R/W	0x0000	on page 903
0x0102	Value Register 5 (VAL5) for submodule 3	R/W	0x0000	on page 904
0x0104–0x0107	Reserved			
0x0108	Output Control Register (OCTRL) for submodule 3	R/W	0x0000	on page 904
0x010A	Status Register (STS) for submodule 3	R/W	0x0000	on page 905
0x010C	Interrupt Enable Register (INTEN) for submodule 3	R/W	0x0000	on page 906
0x010E	DMA Enable Register (DMAEN) for submodule 3	R/W	0x0000	on page 907
0x0110	Output Trigger Control Register (TCTRL) for submodule 3	R/W	0x0000	on page 908
0x0112	Fault Disable Mapping Register (DISMAP) for submodule 3	R/W	0xFFFF	on page 909
0x0114	Deadtime Count Register 0 (DTCNT0) for submodule 3	R/W	0x07FF	on page 909
0x0116	Deadtime Count Register 1 (DTCNT1) for submodule 3	R/W	0x07FF	on page 909
0x0118–0x011F	Reserved			
0x0120	Capture Control Register X (CAPTCTRLX) for submodule 3	R/W	0x0000	on page 910
0x0122	Capture Compare Register X (CAPTCOMPX) for submodule 3	R/W	0x0000	on page 912
0x0124	Capture Value 0 Register (CVAL0) for submodule 3	R	0x0000	on page 912
0x0126	Capture Value 0 Cycle Register (CVAL0C) for submodule 3	R	0x0000	on page 913
0x0128	Capture Value 1 Register (CVAL1) for submodule 3	R	0x0000	on page 913
0x012A	Capture Value 1 Cycle Register (CVAL1C) for submodule 3	R	0x0000	on page 913

Table 28-3. FlexPWM memory map (continued)

Offset from FlexPWM_BASE	Register	Access	Reset value	Location
0x012C–0x013F	Reserved			
0x0140	Output Enable Register (OUTEN)	R/W	0x0000	on page 914
0x142	Output Mask Register (MASK)	R/W	0x0000	on page 914
0x0144	Software Controlled Output Register (SWCOUT)	R/W	0x0000	on page 915
0x0146	Deadtime Source Select Register (DTSRCSEL)	R/W	0x0000	on page 917
0x0148	Master Control Register (MCTRL)	R/W	0x0000	on page 918
0x014A	Reserved			
0x014C	Fault Control Register (FCTRL)	R/W	0x0000	on page 919
0x014E	Fault Status Register (FSTS)	R/W	0x0000	on page 920
0x0150	Fault Filter Register (FFILT)	R/W	0x0000	on page 921
0x0152–0x3FFF	Reserved			

28.3.2 Register descriptions

The address of a register is the sum of a base address and an address offset. The base address is defined at the core level and the address offset is defined at the module level. There are a set of registers for each PWM submodule, for the configuration logic, and for each fault channel.

28.3.2.1 Counter Register (CNT)

The read-only Counter Register (CNT) displays the state of the signed 16-bit submodule counter. This register is not byte-accessible.

Address: Base + 0x0000 (Submodule 0)
Base + 0x0050 (Submodule 1)
Base + 0x00A0 (Submodule 2)
Base + 0x00F0 (Submodule 3)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-3. Counter Register (CNT)

28.3.2.2 Initial Count Register (INIT)

Address: Base + 0x0002 (Submodule 0) Base + 0x00A2 (Submodule 2)
Base + 0x0052 (Submodule 1) Base + 0x00F2 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INIT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-4. Initial Count Register (INIT)

The 16-bit signed value in this buffered, read/write register defines the initial count value for the PWM in PWM clock periods. This value is loaded into the submodule counter when local sync, master sync, or master reload is asserted (based on the value of INIT_SEL) or when FORCE is asserted and force init is enabled. For PWM operation, the buffered contents of this register are loaded into the counter at the start of every PWM cycle. This register is not byte-accessible.

NOTE

The INIT register is buffered. The value written does not take effect until the MCTRL[LDOK] bit is set and the next PWM load cycle begins or CTRL1[LDMOD] is set. This register cannot be written when LDOK is set. Reading INIT reads the value in a buffer and not necessarily the value the PWM generator is currently using.

28.3.2.3 Control 2 Register (CTRL2)

Address: Base + 0x0004 (Submodule 0) Base + 0x00A4 (Submodule 2)
Base + 0x0054 (Submodule 1) Base + 0x00F4 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DBG EN	WAIT EN	IN DEP	PWM23_ INIT	PWM45_ INIT	PWMX_ INIT	INIT_SEL		FRC EN	0	FORCE_SEL		RELOAD _SEL	CLK_SEL		
W										FOR CE						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. Control 2 Register (CTRL2)

Table 28-4. CTRL2 field descriptions

Field	Description
DBGEN	Debug Enable When this bit is set, the PWM continues to run while the device is in debug mode. If the device enters debug mode and this bit is cleared, then the PWM outputs are disabled until debug mode is exited. At that point, the PWM pins resume operation as programmed in the PWM registers. For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in debug mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (such as DC motors), this bit might safely be set, enabling the PWM in debug mode. The key point is that PWM parameter updates will not occur in debug mode. Any motors requiring such updates should be disabled during debug mode. If in doubt, leave this bit cleared.

Table 28-4. CTRL2 field descriptions (continued)

Field	Description
WAITEN	WAIT Enable When this bit is set, the PWM continues to run while the device is in WAIT mode. In this mode, the peripheral clock continues to run, but the CPU clock does not. If the device enters WAIT mode and this bit is cleared, then the PWM outputs are disabled until WAIT mode is exited. At that point, the PWM pins resume operation as programmed in the PWM registers. For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in WAIT mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (such as DC motors), this bit might safely be set, enabling the PWM in WAIT mode. The key point is PWM parameter updates will not occur in this mode. Any motors requiring such updates should be disabled during WAIT mode. If in doubt, leave this bit cleared.
INDEP	Independent or Complementary Pair Operation This bit determines whether the PWMA and PWMB channels are independent PWMs or a complementary PWM pair. 0 PWMA and PWMB form a complementary PWM pair. 1 PWMA and PWMB outputs are independent PWMs.
PWM23_INIT	PWM23 Initial Value This read/write bit determines the initial value for PWMA and the value to which it is forced when FORCE_INIT is asserted. See Figure 28-48.
PWM45_INIT	PWM45 Initial Value This read/write bit determines the initial value for PWMB and the value to which it is forced when FORCE_INIT is asserted. See Figure 28-48.
PWMX_INIT	PWMX Initial Value This read/write bit determines the initial value for PWMX and the value to which it is forced when FORCE_INIT is asserted. See Figure 28-48.
INIT_SEL	Initialization Control Select These read/write bits control the source of the INIT signal that goes to the counter. 00 Local sync (PWMX) causes initialization. 01 Master reload from submodule 0 causes initialization. This setting should not be used in submodule 0 as it forces the INIT signal to logic 0. 10 Master sync from submodule 0 causes initialization. This setting should not be used in submodule 0 as it forces the INIT signal to logic 0. 11 EXT_SYNC causes initialization.
FRCEN	Force Initialization Enable This bit allows the FORCE_OUT signal to initialize the counter without regard to the signal selected by INIT_SEL. This is a software controlled initialization. 0 Initialization from a Force Out event is disabled. 1 Initialization from a Force Out event is enabled.
FORCE	Force Initialization If the FORCE_SEL bits = 000, writing a 1 to this bit results in a Force Out event. This causes the following actions to be taken: • The PWMA and PWMB output pins assume values based on the DTSRCSEL[SEL23] and DTSRCSEL[SEL45] bits. • If the FRCEN bit is set, the counter value is initialized with the INIT register value.

Table 28-4. CTRL2 field descriptions (continued)

Field	Description
FORCE_SEL	Force Source Select This read/write bit determines the source of the FORCE OUTPUT signal for this submodule. 000 The local force signal, FORCE, from this submodule is used to force updates. 001 The master force signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it holds the FORCE OUTPUT signal to logic 0. 010 The local reload signal from this submodule is used to force updates. 011 The master reload signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it holds the FORCE OUTPUT signal to logic 0. 100 The local sync signal from this submodule is used to force updates. 101 The master sync signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it holds the FORCE OUTPUT signal to logic 0. 110 The external force signal, EXT_FORCE, from outside the PWM module causes updates. 111 Reserved.
RELOAD_SEL	Reload Source Select This read/write bit determines the source of the RELOAD signal for this submodule. When this bit is set, the LDOK bit in submodule 0 should be used since the local LDOK bit is ignored. 0 The local RELOAD signal is used to reload registers. 1 The master RELOAD signal (from submodule 0) is used to reload registers. This setting should not be used in submodule 0 as it forces the RELOAD signal to logic 0.
CLK_SEL	Clock Source Select These read/write bits determine the source of the clock signal for this submodule. 00 The IPBus clock is used as the clock for the local prescaler and counter. 01 EXT_CLK is used as the clock for the local prescaler and counter. 10 Submodule 0's clock (AUX_CLK) is used as the source clock for the local prescaler and counter. This setting should not be used in submodule 0 as it forces the clock to logic 0. 11 Reserved.

28.3.2.4 Control 1 Register (CTRL1)

Address: Base + 0x0006 (Submodule 0) Base + 0x00A6 (Submodule 2)
Base + 0x0056 (Submodule 1) Base + 0x00F6 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LDFQ				HALF	FULL	DT		0	PRSC			0	LDMOD	0	DBLEN
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 28-6. Control 1 Register (CTRL1)

Table 28-5. CTRL1 field descriptions

Field	Description
LDFQ	Load Frequency These buffered read/write bits select the PWM load frequency according to Table 28-6**.** Reset clears the LDFQ bits, selecting loading every PWM opportunity. A PWM opportunity is determined by the HALF and FULL bits. The LDFQ*x* bits take effect when the current load cycle is complete regardless of the state of the LDOK bit. Reading the LDFQx bits reads the buffered values and not necessarily the values currently in effect. See Table 28-6.

Table 28-5. CTRL1 field descriptions (continued)

Field	Description
HALF	Half Cycle Reload This read/write bit enables half-cycle reloads. A half cycle is defined by when the submodule counter matches the VAL0 register and does not have to be halfway through the PWM cycle. 0 Half-cycle reloads disabled. 1 Half-cycle reloads enabled.
FULL	Full Cycle Reload This read/write bit enables full-cycle reloads. A full cycle is defined by when the submodule counter matches the VAL1 register. Either the HALF or FULL bit must be set in order to move the buffered data into the registers used by the PWM generators. If both the HALF and FULL bits are set, then reloads can occur twice per cycle. 0 Full-cycle reloads disabled. 1 Full-cycle reloads enabled (default).
DT	Deadtime These read-only bits reflect the sampled values of the PWMX input at the end of each deadtime. Sampling occurs at the end of deadtime 0 for DT[0] and the end of deadtime 1 for DT[1]. Reset clears these bits.
PRSC	Prescaler These buffered read/write bits select the divide ratio of the PWM clock frequency selected by CLK_SEL as illustrated in Table 28-7. **Note:** Reading the PRSCx bits reads the buffered values and not necessarily the values currently in effect. The PRSC*x* bits take effect at the beginning of the next PWM cycle and only when the load okay bit, LDOK, is set or LDMOD is set. This field cannot be written when LDOK is set.
LDMOD	Load Mode Select This read/write bit selects the timing of loading the buffered registers for this submodule. 0 Buffered registers of this submodule are loaded and take effect at the next PWM reload if LDOK is set. 1 Buffered registers of this submodule are loaded and take effect immediately upon LDOK being set.
DBLEN	Double Switching Enable This read/write bit enables the double switching PWM behavior. 0 Double switching disabled. 1 Double switching enabled.

Table 28-6. PWM reload frequency

LDFQ	PWM reload frequency	LDFQ	PWM reload frequency
0000	Every PWM opportunity	1000	Every 9 PWM opportunities
0001	Every 2 PWM opportunities	1001	Every 10 PWM opportunities
0010	Every 3 PWM opportunities	1010	Every 11 PWM opportunities
0011	Every 4 PWM opportunities	1011	Every 12 PWM opportunities
0100	Every 5 PWM opportunities	1100	Every 13 PWM opportunities
0101	Every 6 PWM opportunities	1101	Every 14 PWM opportunities
0110	Every 7 PWM opportunities	1110	Every 15 PWM opportunities

Table 28-6. PWM reload frequency (continued)

LDFQ	PWM reload frequency	LDFQ	PWM reload frequency
0111	Every 8 PWM opportunities	1111	Every 16 PWM opportunities

Table 28-7. PWM prescaler

PRSC	PWM clock frequency	PRSC	PWM clock frequency
000	f_{clk}	100	$f_{clk}/16$
001	$f_{clk}/2$	101	$f_{clk}/32$
010	$f_{clk}/4$	110	$f_{clk}/64$
011	$f_{clk}/8$	111	$f_{clk}/128$

28.3.2.5 Value Register 0 (VAL0)

Address: Base + 0x0008 (Submodule 0) Base + 0x00A8 (Submodule 2)
Base + 0x0058 (Submodule 1) Base + 0x00F8 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. Value Register 0 (VAL0)

The 16-bit signed value in this buffered, read/write register defines the mid-cycle reload point for the PWM in PWM clock periods. This value also defines when the PWMX signal is set and the local sync signal is reset. This register is not byte-accessible.

NOTE

The VAL0 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL0 cannot be written when LDOK is set. Reading VAL0 reads the value in a buffer. It is not necessarily the value the PWM generator is currently using.

28.3.2.6 Value Register 1 (VAL1)

Address: Base + 0x000A (Submodule 0) Base + 0x00AA (Submodule 2)
Base + 0x005A (Submodule 1) Base + 0x00FA (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. Value Register 1 (VAL1)

The 16-bit signed value written to this buffered, read/write register defines the modulo count value (maximum count) for the submodule counter. Upon reaching this count value, the counter reloads itself with the contents of the INIT register and asserts the local sync signal while resetting PWMX. This register is not byte-accessible.

NOTE

The VAL1 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL1 cannot be written when LDOK is set. Reading VAL1 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

Since the VAL1 register is used to determine the turn off count of the PWMX output and also to define the end count of the PWM cycle, the PWMX output cannot achieve 100% duty cycle.

28.3.2.7 Value Register 2 (VAL2)

Address: Base + 0x000C (Submodule 0), Base + 0x005C (Submodule 1), Base + 0x00AC (Submodule 2), Base + 0x00FC (Submodule 3) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-9. Value register 2 (VAL2)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM23 high (Figure 28-2). This register is not byte-accessible.

NOTE

The value in the VAL3 register must always be greater than or equal to the value in the VAL2 register.

NOTE

The VAL2 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL2 cannot be written when LDOK is set. Reading VAL2 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

28.3.2.8 Value Register 3 (VAL3)

Address: Base + 0x000E (Submodule 0)
Base + 0x005E (Submodule 1)
Base + 0x00AE (Submodule 2)
Base + 0x00FE (Submodule 3)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL3															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-10. Value register 3 (VAL3)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM23 low (Figure 28-2). This register is not byte-accessible.

NOTE

The value in the VAL3 register must always be greater than or equal to the value in the VAL2 register.

NOTE

The VAL3 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL3 cannot be written when LDOK is set. Reading VAL3 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

28.3.2.9 Value Register 4 (VAL4)

Address: Base + 0x0010 (Submodule 0)
Base + 0x0060 (Submodule 1)
Base + 0x00B0 (Submodule 2)
Base + 0x0100 (Submodule 3)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL4															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-11. Value register 4 (VAL4)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM45 high (Figure 28-2). This register is not byte-accessible.

NOTE

The value in the VAL5 register must always be greater than or equal to the value in the VAL4 register.

NOTE

The VAL4 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL4 cannot be written when LDOK is set. Reading VAL4 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

28.3.2.10 Value Register 5 (VAL5)

Address: Base + 0x0012(Submodule 0) Base + 0x00B2 (Submodule 2)
Base + 0x0062 (Submodule 1) Base + 0x0102 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL5															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-12. Value Register 5 (VAL5)

The 16-bit signed value in this buffered, read/write register defines the count value to set PWM45 low (Figure 28-2). This register is not byte-accessible.

NOTE

The value in the VAL5 register must always be greater than or equal to the value in the VAL4 register.

NOTE

The VAL5 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins or LDMOD is set. VAL5 cannot be written when LDOK is set. Reading VAL5 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

28.3.2.11 Output Control Register (OCTRL)

Address: Base + 0x0018 (Submodule 0) Base + 0x00B8 (Submodule 2)
Base + 0x0068 (Submodule 1) Base + 0x0108 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWMA_IN	PWMB_IN	PWMX_IN	0	0	POLA	POLB	POLX	0	0	PWMAFS		PWMBFS		PWMXFS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-13. Output Control Register (OCTRL)

Table 28-8. OCTRL field descriptions

Field	Description
PWMA_IN	PWMA Input This read-only bit shows the logic value currently being driven into the PWMA input.
PWMB_IN	PWMB Input This read-only bit shows the logic value currently being driven into the PWMB input.
PWMX_IN	PWMX Input This read-only bit shows the logic value currently being driven into the PWMX input.

Table 28-8. OCTRL field descriptions (continued)

Field	Description
POLA	PWMA Output Polarity This bit inverts the PWMA output polarity. 0 PWMA output not inverted. A high level on the PWMA pin represents the “on” or “active” state. 1 PWMA output inverted. A low level on the PWMA pin represents the “on” or “active” state.
POLB	PWMB Output Polarity This bit inverts the PWMB output polarity. 0 PWMB output not inverted. A high level on the PWMB pin represents the “on” or “active” state. 1 PWMB output inverted. A low level on the PWMB pin represents the “on” or “active” state.
POLX	PWMX Output Polarity This bit inverts the PWMX output polarity. 0 PWMX output not inverted. A high level on the PWMX pin represents the “on” or “active” state. 1 PWMX output inverted. A low level on the PWMX pin represents the “on” or “active” state.
PWMAFS	PWMA Fault State These bits determine the fault state for the PWMA output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1*x* Output is tristated.
PWMBFS	PWMB Fault State These bits determine the fault state for the PWMB output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1*x* Output is tristated.
PWMXFS	PWMX Fault State These bits determine the fault state for the PWMX output during fault conditions and STOP mode. It may also define the output state during WAIT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1*x* Output is tristated.

28.3.2.12 Status Register (STS)

Address: Base + 0x001A (Submodule 0) Base + 0x00BA (Submodule 2)
Base + 0x006A (Submodule 1) Base + 0x010A (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	RUF	REF	RF	0	0	0	0	CFX1	CFX0	CMPF					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-14. Status register (STS)

Table 28-9. STS field descriptions

Field	Description
RUF	Registers Updated Flag This read-only flag is set when one of the INIT, VAL*x*, or PRSC fields has been written resulting in non-coherent data in the set of double-buffered registers. Clear RUF by a proper reload sequence consisting of a reload signal while LDOK = 1. Reset clears RUF. 0 No register update has occurred since last reload. 1 At least one of the double-buffered registers has been updated since the last reload.
REF	Reload Error Flag This read/write flag is set when a reload cycle occurs while LDOK is 0 and the double-buffered registers are in a non-coherent state (RUF = 1). Clear REF by writing a logic 1 to the REF bit. Reset clears REF. 0 No reload error occurred. 1 Reload signal occurred with non-coherent data and LDOK = 0.
RF	Reload Flag This read/write flag is set at the beginning of every reload cycle regardless of the state of the LDOK bit. Clear RF by writing a logic 1 to the RF bit when VALDE is clear (non-DMA mode). RF can also be cleared by the DMA done signal when VALDE is set (DMA mode). Reset clears RF. 0 No new reload cycle since last RF clearing. 1 New reload cycle since last RF clearing.
CFX1	Capture Flag X1 This bit is set when the word count of the Capture X1 FIFO (CX1CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX1DE is clear (non-DMA mode) or by the DMA done signal if CX1DE is set (DMA mode). Reset clears this bit.
CFX0	Capture Flag X0 This bit is set when the word count of the Capture X0 FIFO (CX0CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX0DE is clear (non-DMA mode) or by the DMA done signal if CX0DE is set (DMA mode). Reset clears this bit.
CMPF	Compare Flags These bits are set when the submodule counter value matches the value of one of the VAL*x* registers. Clear these bits by writing a 1 to a bit position. 0 No compare event has occurred for a particular VAL*x* value. 1 A compare event has occurred for a particular VAL*x* value.

28.3.2.13 Interrupt Enable Register (INTEN)

Address: Base + 0x001C (Submodule 0) Base + 0x00BC (Submodule 2)
Base + 0x006C (Submodule 1) Base + 0x010C (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	REIE	RIE	0	0	0	0	CX1IE	CX0IE	CMPIE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-15. Interrupt Enable register (INTEN)

Table 28-10. INTEN field descriptions

Field	Description
REIE	Reload Error Interrupt Enable This read/write bit enables the reload error flag (REF) to generate CPU interrupt requests. Reset clears RIE. 0 REF CPU interrupt requests disabled. 1 REF CPU interrupt requests enabled.
RIE	Reload Interrupt Enable This read/write bit enables the reload flag (RF) to generate CPU interrupt requests. Reset clears RIE. 0 RF CPU interrupt requests disabled. 1 RF CPU interrupt requests enabled.
CX1IE	Capture X 1 Interrupt Enable This bit allows the CFX1 flag to create an interrupt request to the CPU. Do not set both this bit and the CX1DE bit. 0 Interrupt request disabled for CFX1. 1 Interrupt request enabled for CFX1.
CX0IE	Capture X 0 Interrupt Enable This bit allows the CFX0 flag to create an interrupt request to the CPU. Do not set both this bit and the CX0DE bit. 0 Interrupt request disabled for CFX0. 1 Interrupt request enabled for CFX0.
CMPIE	Compare Interrupt Enables These bits enable the CMPF flags to cause a compare interrupt request to the CPU. 0 The corresponding CMPF bit does not cause an interrupt request 1 The corresponding CMPF bit causes an interrupt request.

28.3.2.14 DMA Enable Register (DMAEN)

Address: Base + 0x001E (Submodule 0) Base + 0x00BE (Submodule 2)
Base + 0x006E (Submodule 1) Base + 0x010E (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	VALDE	FAND	CAPTDE		0	0	0	0	CX1DE	CX0DE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-16. DMA Enable register (DMAEN)

Table 28-11. DMAEN field descriptions

Field	Description
VALDE	Value Registers DMA Enable This read/write bit enables DMA write requests for the VAL*x* registers when RF is set. Reset clears VALDE. 0 DMA write requests are disabled. 1 DMA write requests for the VAL*x* registers are enabled. **NOTE** When some submodules use DMA to load their VALx registers and other submodules use non-DMA means that means direct writes from the CPU, the LDOK bits for the non-DMA submodules can be incorrectly cleared at the completion of the DMA controlled load cycle. This leads to the non-DMA channels not being properly updated. Submodules that use DMA to read the input capture registers do not cause a problem for non-DMA submodules. To manage this behavior, set the DMA enable bit to 1 also for non-DMA submodules, according to this the DMA will not incorrectly clear the LDOK bit for non-DMA submodules but they will be set to 1 at the end of each DMA cycle. When the CPU has to update the VALx registers of non-DMA submodules, first clear LDOK bit for non-DMA submodules.
FAND	FIFO Watermark AND Control This read/write bit works in conjunction with the CAPTDE field when it is set to watermark mode (CAPTDE = 01). While the CXxDE bits determine which FIFO watermarks the DMA read request is sensitive to, this bit determines if the selected watermarks are ANDed together or ORed together in order to create the request. 0 Selected FIFO watermarks are ORed together. 1 Selected FIFO watermarks are ANDed together.
CAPTDE	Capture DMA Enable Source Select These read/write bits select the source of enabling the DMA read requests for the capture FIFOs. Reset clears these bits. 00 Read DMA requests disabled. 01 Exceeding a FIFO watermark sets the DMA read request. This requires at least 1 of the CX1DE or CX0DE bits to also be set in order to determine which watermark(s) the DMA request is sensitive to. 10 A local sync (VAL1 matches counter) sets the read DMA request. 11 A local reload (RF being set) sets the read DMA request.
CX1DE	Capture X1 FIFO DMA Enable This read/write bit enables DMA read requests for the Capture X1 FIFO data when CFX1 is set. Reset clears CX1DE. Do not set both this bit and the CX1IE bit.
CX0DE	Capture X0 FIFO DMA Enable This read/write bit enables DMA read requests for the Capture X0 FIFO data when CFX0 is set. Reset clears CX0DE. Do not set both this bit and the CX0IE bit.

28.3.2.15 Output Trigger Control Register (TCTRL)

Address: Base + 0x0020 (Submodule 0) Base + 0x00C0 (Submodule 2)
Base + 0x0070 (Submodule 1) Base + 0x0110 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	OUT_TRIG_EN[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-17. Output Trigger Control register (TCTRL)

Table 28-12. TCTRL field descriptions

Field	Description
OUT_TRIG_EN[5:0]	Output Trigger Enables These bits enable the generation of OUT_TRIG0 and OUT_TRIG1 outputs based on the counter value matching the value in one or more of the VAL0-5 registers where OUT_TRIG_EN[0] refers to VAL0, OUT_TRIG_EN[1] refers to VAL1 and so on. VAL0, VAL2, and VAL4 are used to generate OUT_TRIG0. VAL1, VAL3, and VAL5 are used to generate OUT_TRIG1. The OUT_TRIGx signals are only asserted as long as the counter value matches the VAL*x* value; therefore, as many as six triggers can be generated (three each on OUT_TRIG0 and OUT_TRIG1) per PWM cycle per submodule. 0 OUT_TRIG*x* is not set when the counter value matches the VAL*x* value. 1 OUT_TRIG*x* is set when the counter value matches the VAL*x* value.

28.3.2.16 Fault Disable Mapping Register (DISMAP)

The Fault Disable Mapping Register (DISMAP) determines which PWM pins are disabled by the fault protection inputs, illustrated in Table 28-59 in Section 28.4.3.11, Fault protection. Reset sets all of the bits in the fault disable mapping register.

Address: Base + 0x0022 (Submodule 0) Base + 0x00C2 (Submodule 2)
Base + 0x0072 (Submodule 1) Base + 0x0112 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	1	DISX[3:0]				DISB[3:0]				DISA[3:0]			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 28-18. Fault Disable Mapping register (DISMAP)

Table 28-13. DISMAP field descriptions

Field	Description
DISX	PWMX Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULT*x* inputs. The PWMX output is turned off if there is a logic 1 on a FAULT*x* input and a 1 in the corresponding bit of the DISX field. A reset sets all DISX bits.
DISB	PWMB Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULT*x* inputs. The PWMB output is turned off if there is a logic 1 on a FAULT*x* input and a 1 in the corresponding bit of the DISB field. A reset sets all DISB bits.
DISA	PWMA Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULT*x* inputs. The PWMA output is turned off if there is a logic 1 on a FAULT*x* input and a 1 in the corresponding bit of the DISA field. A reset sets all DISA bits.

28.3.2.17 Deadtime Count Registers (DTCNT0, DTCNT1)

Deadtime operation is only applicable to complementary channel operation. The 12-bit values written to these registers are in terms of IPBus clock cycles regardless of the setting of PRSC and/or CLK_SEL.

Reset sets the deadtime count registers to a default value of 0x0FFF, selecting a deadtime of 4095 IPBus clock cycles. These registers are not byte-accessible.

Address: Base + 0x0024 (Submodule 0) Base + 0x00C4 (Submodule 2)
Base + 0x0074 (Submodule 1) Base + 0x0114 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT0										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 28-19. Deadtime Count Register 0 (DTCNT0)

Address: Base + 0x0026 (Submodule 0) Base + 0x00C6 (Submodule 2)
Base + 0x0076 (Submodule 1) Base + 0x0116 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT1										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 28-20. Deadtime Count register 1 (DTCNT1)

The DTCNT0 field controls the deadtime during 0 to 1 transitions of the PWMA output (assuming normal polarity). The DTCNT1 field controls the deadtime during 0 to 1 transitions of the complementary PWMB output.

28.3.2.18 Capture Control X Register (CAPTCTRLX)

Address: Base + 0x0030 (Submodule 0) Base + 0x00D0 (Submodule 2)
Base + 0x0080 (Submodule 1) Base + 0x0120 (Submodule 3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CX1CNT			CX0CNT			CFXWM		EDG CNTX_EN	INP_ SELX	EDGX1		EDGX0		ONE SHOTX	ARMX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-21. Capture Control X register (CAPTCTRLX)

Table 28-14. CAPTCTRLX field descriptions

Field	Description
CX1CNT	Capture X1 FIFO Word Count This field reflects the number of words in the CVAL1 FIFO.
CX0CNT	Capture X0 FIFO Word Count This field reflects the number of words in the CVAL0 FIFO.
CFXWM	Capture X FIFOs Water Mark This field represents the water mark level for capture X FIFOs. The capture flags, CFX1 and CFX0, are not set until the word count of the corresponding FIFO is greater than this water mark level.

Table 28-14. CAPTCTRLX field descriptions (continued)

Field	Description
EDGCNTX_EN	Edge Counter X Enable This bit enables the edge counter, which counts rising and falling edges on the PWMX input signal. 0 Edge counter disabled and held in reset. 1 Edge counter enabled.
INPSELX	Input Select X This bit selects between the raw PWMX input signal and the output of the edge counter/compare circuitry as the source for the input capture circuit. 0 Raw PWMX input signal selected as source. 1 Output of edge counter/compare selected as source. **Note:** When INPSELX = 1, the internal edge counter is enabled and the rising and/or falling edges specified by the EDGX0 and EDGX1 fields are ignored. The software must still place a value other than 00 in either or both of the EDGX0 and/or EDGX1 fields in order to enable one or both of the capture registers.
EDGX1	Edge X 1 These bits control the input capture 1 circuitry by determining which input edges cause a capture event. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
EDGX0	Edge X 0 These bits control the input capture 0 circuitry by determining which input edges cause a capture event. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
ONESHOTX	One Shot Mode Aux This bit selects between free running and one shot mode for the input capture circuitry. 0 Free running mode is selected If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and capture circuit 0 is re-armed. The process continues indefinitely. If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit. 1 One shot mode is selected. If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and the ARMX bit is cleared. No further captures are performed until the ARMX bit is set again. If only one of the capture circuits is enabled, then a single capture occurs on the enabled capture circuit and the ARMX bit is then cleared.

Table 28-14. CAPTCTRLX field descriptions (continued)

Field	Description
ARMX	Arm X Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one shot mode and the enabled capture circuit(s) has had a capture event(s). In order to restart capture functionality, disable the capture function by setting ARMX=0, read CVAL0 and CVAL1 until CX0CNT and CX1CNT are zero, then re-enable capture by setting ARMX=1. 0 Input capture operation is disabled. 1 Input capture operation as specified by the EDGX bits is enabled.

28.3.2.19 Capture Compare X Register (CAPTCMPX)

Address: Base + 0x0032 (Submodule 0), Base + 0x0082 (Submodule 1), Base + 0x00D2 (Submodule 2), Base + 0x0122 (Submodule 3) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDGCNTX								EDGCMPX							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-22. Capture Compare X register (CAPTCMPX)

Table 28-15. CAPTCMPX field descriptions

Field	Description
EDGCNTX	Edge Counter X This read-only field contains the edge counter value for the PWMX input capture circuitry.
EDGCMPX	Edge Compare X This read/write field is the compare value associated with the edge counter for the PWMX input capture circuitry.

28.3.2.20 Capture Value 0 Register (CVAL0)

Address: Base + 0x0034 (Submodule 0), Base + 0x0084 (Submodule 1), Base + 0x00D4 (Submodule 2), Base + 0x0124 (Submodule 3) — Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPTVAL0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-23. Capture Value 0 register (CVAL0)

This read-only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX0 bits. This is actually a 4-deep FIFO and not a single register. This register is not byte-accessible.

28.3.2.21 Capture Value 0 Cycle Register (CVAL0CYC)

Address: Base + 0x0036 (Submodule 0) Base + 0x00D6 (Submodule 2)
Base + 0x0086 (Submodule 1) Base + 0x0126 (Submodule 3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL0CYC		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-24. Capture Value 0 Cycle register (CVAL0CYC)

This read-only register stores the cycle number corresponding to the value captured in CVAL0. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. CVAL0CYC will count up and roll over to 0. CVAL0CYC and CVAL1CYC can be used to verify that the rising and falling edges occurred in the same PWM cycle. This is actually a 4 deep FIFO and not a single register.

28.3.2.22 Capture Value 1 Register (CVAL1)

Address: Base + 0x0038 (Submodule 0) Base + 0x00D8 (Submodule 2)
Base + 0x0088 (Submodule 1) Base + 0x0128 (Submodule 3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPTVAL1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-25. Capture Value 1 register (CVAL1)

This read-only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX1 bits. This is actually a 4 deep FIFO and not a single register. This register is not byte-accessible.

28.3.2.23 Capture Value 1 Cycle Register (CVAL1CYC)

Address: Base + 0x003A (Submodule 0) Base + 0x00DA (Submodule 2)
Base + 0x008A (Submodule 1) Base + 0x012A (Submodule 3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL1CYC		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-26. Capture Value 1 Cycle register (CVAL1CYC)

This read-only register stores the cycle number corresponding to the value captured in CVAL1. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. CVAL0CYC will count up and roll over to 0. CVAL0CYC and CVAL1CYC can be used to verify that the rising and falling edges occurred in the same PWM cycle. This is actually a 4-deep FIFO and not a single register.

28.3.3 Configuration Registers

28.3.3.1 Output Enable Register (OUTEN)

Address: Base + 0x0140 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PWMA_EN[3:0]				PWMB_EN[3:0]				PWMX_EN[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-27. Output Enable register (OUTEN)

The relationship between the fields of OUTEN and the submodules is as follows:

- PWM*x*_EN[3] enables/disables submodule 3
- PWM*x*_EN[2] enables/disables submodule 2
- PWM*x*_EN[1] enables/disables submodule 1
- PWM*x*_EN[0] enables/disables submodule 0

Table 28-16. OUTEN field descriptions

Field	Description
PWMA_EN[3:0]	PWMA Output Enables These bits enable the PWMA outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMA pin is being used for input capture. 0 PWMA output disabled. 1 PWMA output enabled.
PWMB_EN[3:0]	PWMB Output Enables These bits enable the PWMB outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMB pin is being used for input capture. 0 PWMB output disabled. 1 PWMB output enabled.
PWMX_EN[3:0]	PWMX Output Enables These bits enable the PWMX outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMX pin is being used for input capture or deadtime correction. 0 PWMX output disabled. 1 PWMX output enabled.

28.3.3.2 Mask Register (MASK)

Address: Base + 0x0142 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	MASKA[3:0]				MASKB[3:0]				MASKX[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-28. Mask register (MASK)

The relationship between the fields of MASK and the submodules is as follows:

- MASK*x*[3] enables/disables submodule 3

- MASK*x*[2] enables/disables submodule 2
- MASK*x*[1] enables/disables submodule 1
- MASK*x*[0] enables/disables submodule 0

Table 28-17. MASK field descriptions

Field	Description
MASKA[3:0]	PWMA Masks These bits mask the PWMA outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMA output normal. 1 PWMA output masked.
MASKB[3:0]	PWMB Masks These bits mask the PWMB outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMB output normal. 1 PWMB output masked.
MASKX[3:0]	PWMX Masks These bits mask the PWMX outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMX output normal. 1 PWMX output masked.

NOTE

The MASK*x* bits are double-buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 28-49 to see how FORCE_OUT is generated. Reading the MASK bits reads the buffered value and not necessarily the value currently in effect.

28.3.3.3 Software Controlled Output Register (SWCOUT)

Address: Base + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OUT 23_3	OUT 45_3	OUT 23_2	OUT 45_2	OUT 23_1	OUT 45_1	OUT 23_0	OUT 45_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-29. Software Controlled Output Register (SWCOUT)

Table 28-18. SWCOUT field descriptions

Field	Description
OUT23_3	Software Controlled Output 23_3 This bit is only used when SEL23 for submodule 3 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWM23. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWM23.

Table 28-18. SWCOUT field descriptions (continued)

Field	Description
OUT45_3	Software Controlled Output 45_3 This bit is only used when SEL45 for submodule 3 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWM45. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWM45.
OUT23_2	Software Controlled Output 23_2 This bit is only used when SEL23 for submodule 2 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWM23. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWM23.
OUT45_2	Software Controlled Output 45_2 This bit is only used when SEL45 for submodule 2 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWM45. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWM45.
OUT23_1	Software Controlled Output 23_1 This bit is only used when SEL23 for submodule 1 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWM23. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWM23.
OUT45_1	Software Controlled Output 45_1 This bit is only used when SEL45 for submodule 1 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWM45. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWM45.
OUT23_0	Software Controlled Output 23_0 This bit is only used when SEL23 for submodule 0 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWM23. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWM23.
OUT45_0	Software Controlled Output B45_0 This bit is only used when SEL45 for submodule 0 is set to 0b10. It allows software to control which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWM45. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWM45.

NOTE

These bits are double-buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 28-49 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

28.3.3.4 Deadtime Source Select Register (DTSRCSEL)

Address: Base + 0x0146 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SEL23_3		SEL45_3		SEL23_2		SEL45_2		SEL23_1		SEL45_1		SEL23_0		SEL45_0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-30. Deadtime Source Select Register (DTSRCSEL)

Table 28-19. DTSRCSEL field descriptions

Field	Description
SEL23_3	PWM23_3 Control Select This field selects possible overrides to the generated PWM23 signal in submodule 3 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM23_3 signal is used by the deadtime logic. 01 Inverted generated PWM23_3 signal is used by the deadtime logic. 10 OUT23_3 bit is used by the deadtime logic. 11 Reserved.
SEL45_3	PWM45_3 Control Select This field selects possible overrides to the generated PWM45 signal in submodule 3 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM45_3 signal is used by the deadtime logic. 01 Inverted generated PWM45_3 signal is used by the deadtime logic. 10 OUT45_3 bit is used by the deadtime logic. 11 Reserved
SEL23_2	PWM23_2 Control Select This field selects possible overrides to the generated PWM23 signal in submodule 2 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM23_2 signal is used by the deadtime logic. 01 Inverted generated PWM23_2 signal is used by the deadtime logic. 10 OUT23_2 bit is used by the deadtime logic. 11 Reserved.
SEL45_2	PWM45_2 Control Select This field selects possible overrides to the generated PWM45 signal in submodule 2 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM45_2 signal is used by the deadtime logic. 01 Inverted generated PWM45_2 signal is used by the deadtime logic. 10 OUT45_2 bit is used by the deadtime logic. 11 Reserved.
SEL23_1	PWM23_1 Control Select This field selects possible overrides to the generated PWM23 signal in submodule 1 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM23_1 signal is used by the deadtime logic. 01 Inverted generated PWM23_1 signal is used by the deadtime logic. 10 OUT23_1 bit is used by the deadtime logic. 11 Reserved.

Table 28-19. DTSRCSEL field descriptions (continued)

Field	Description
SEL45_1	PWM45_1 Control Select This field selects possible overrides to the generated PWM45 signal in submodule 1 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM45_1 signal is used by the deadtime logic. 01 Inverted generated PWM45_1 signal is used by the deadtime logic. 10 OUT45_1 bit is used by the deadtime logic. 11 Reserved.
SEL23_0	PWM23_0 Control Select This field selects possible overrides to the generated PWM23 signal in submodule 0 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM23_0 signal is used by the deadtime logic. 01 Inverted generated PWM23_0 signal is used by the deadtime logic. 10 OUT23_0 bit is used by the deadtime logic. 11 Reserved.
SEL45_0	PWM45_0 Control Select This field selects possible overrides to the generated PWM45 signal in submodule 0 that are passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWM45_0 signal is used by the deadtime logic. 01 Inverted generated PWM45_0 signal is used by the deadtime logic. 10 OUT45_0 bit is used by the deadtime logic. 11 Reserved.

NOTE

The deadtime source select bits are double-buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 28-49 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

28.3.3.5 Master Control Register (MCTRL)

Address: Base + 0x0148 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	RUN				0	0	0	0	LDOK			
W									CLDOK							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-31. Master Control Register (MCTRL)

The relationship between the fields of MCTRL and the submodules is as follows:

- *Field*[3] refers to submodule 3
- *Field*[2] refers to submodule 2
- *Field*[1] refers to submodule 1
- *Field*[0] refers to submodule 0

Table 28-20. MCTRL field descriptions

Field	Description
RUN[3:0]	Run This read/write bit enables the clocks to the PWM generator. When RUN equals zero, the submodule counter is reset. A reset clears RUN. 0 Do not load new values. 1 PWM generator enabled. **Note:** For proper initialization of the LDOK and RUN bits, see Section 28.4.4.5, Initialization.
CLDOK[3:0]	Clear Load Okay This write-only bit clears the LDOK bit. Write a 1 to this location to clear the corresponding LDOK. If a reload occurs with LDOK set at the same time that CLDOK is written, then the reload is not performed and LDOK is cleared. This bit is self-clearing and always reads as a 0.
LDOK[3:0]	Load Okay This read/set bit loads the PRSC bits of CTRL1 and the INIT and VAL*x* registers into a set of buffers. The buffered prescaler divisor, submodule counter modulus value, and PWM pulse width take effect at the next PWM reload. Set LDOK by reading it when it is logic 0 and then writing a logic 1 to it. The VAL*x*, INIT, and PRSC fields cannot be written while LDOK is set. LDOK is automatically cleared after the new values are loaded, or can be manually cleared before a reload by writing a logic 1 to CLDOK. This bit cannot be written with a zero. LDOK can be set in DMA mode when the DMA indicates that it has completed the update of all PRSC, INIT, and VAL*x* fields. Reset clears LDOK. 0 Do not load new values. 1 Load prescaler, modulus, and PWM values. **Note:** For proper initialization of the LDOK and RUN bits, see Section 28.4.4.5, Initialization.

28.3.4 Fault Channel Registers

28.3.4.1 Fault Control Register (FCTRL)

Address: Base + 0x014C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FLVL				FAUTO				FSAFE				FIE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-32. Fault Control Register (FCTRL)

Table 28-21. FCTRL field descriptions

Field	Description
FLVL	Fault Level These read/write bits select the active logic level of the individual fault inputs. A reset clears FLVL. 0 A logic 0 on the fault input indicates a fault condition. 1 A logic 1 on the fault input indicates a fault condition.

Table 28-21. FCTRL field descriptions (continued)

Field	Description
FAUTO	Automatic Fault Clearing These read/write bits select automatic or manual clearing of faults. A reset clears FAUTO. 0 Manual fault clearing. PWM outputs disabled by this fault are not enabled until the FFLAG*x* bit is clear at the start of a half cycle. This is further controlled by the FSAFE bits. 1 Automatic fault clearing. PWM outputs disabled by this fault are enabled when the FFPIN*x* bit is clear at the start of a half cycle without regard to the state of FFLAG*x* bit.
FSAFE	Fault Safety Mode These read/write bits select the safety mode during manual fault clearing. A reset clears FSAFE. 0 Normal mode. PWM outputs disabled by this fault are not enabled until the FFLAG*x* bit is clear at the start of a half cycle without regard to the state of the FFPIN*x* bit. The PWM outputs disabled by this fault input are not re-enabled until the actual FAULT*x* input signal deasserts since the fault input combinationally disables the PWM outputs (as programmed in DISMAP). 1 Safe mode. PWM outputs disabled by this fault are not enabled until the FFLAG*x* bit is clear and the FFPIN*x* bit is clear at the start of a half cycle. **Note:** The FFPIN*x* bit may indicate a fault condition still exists even though the actual fault signal at the FAULT*x* pin is clear due to the fault filter latency.
FIE	Fault Interrupt Enables This read/write bit enables CPU interrupt requests generated by the FAULT*x* pins. A reset clears FIE. 0 FAULT*x* CPU interrupt requests disabled. 1 FAULT*x* CPU interrupt requests enabled. **Note:** The fault protection circuit is independent of the FIEx bit and is always active. If a fault is detected, the PWM outputs are disabled according to the disable mapping register.

28.3.4.2 Fault Status Register (FSTS)

Address: Base + 0x014E Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	FTEST	FFPIN				0	0	0	0	FFLAG			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-33. Fault Status Register (FSTS)

Table 28-22. FSTS field descriptions

Field	Description
FTEST	Fault Test These read/write bits simulate a fault condition. Setting this bit causes a simulated fault to be sent into all of the fault filters. The condition propagates to the fault flags and possibly the PWM outputs depending on the DISMAP settings. Clearing this bit removes the simulated fault condition. 0 No fault. 1 Cause a simulated fault.
FFPIN	Filtered Fault Pins These read-only bits reflect the current state of the filtered FAULT*x* pins converted to high polarity. A logic 1 indicates a fault condition exists on the filtered FAULT*x* pin. A reset has no effect on FFPIN.

Table 28-22. FSTS field descriptions (continued)

Field	Description
FFLAG	Fault Flags These read-only flags are set within 2 CPU cycles after a transition to active on the FAULT*x* pin. Clear FFLAG*x* by writing a logic 1 to it. A reset clears FFLAG. 0 No fault on the FAULT*x* pin. 1 Fault on the FAULT*x* pin. **Note:** The FFLAG[3:0] flags are set out of reset. They should be cleared before enabling the Fault Control feature.

28.3.4.3 Fault Filter Register (FFILT)

Address: Base + 0x0150 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GSTR	0	0	0	0	FILT_CNT			FILT_PER							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-34. Fault Filter Register (FFILT)

The settings in this register are shared among each of the fault input filters.

Table 28-23. FFILT field descriptions

Field	Description
GSTR	Fault Glitch Stretch Enable This bit enables the fault glitch stretching logic. This logic ensures that narrow fault glitches are stretched to be at least 2 IPBus clock cycles wide. In some cases a narrow fault input can cause problems due to the short PWM output shutdown/re-activation time. The stretching logic ensures that a glitch on the fault input, when the fault filter is disabled, is registered in the fault flags. 0 Fault input glitch stretching is disabled. 1 Input fault signals are stretched to at least 2 IPBus clock cycles.
FILT_CNT	Fault Filter Count These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in Section 28.3.4.3.1, Input filter considerations.
FILT_PER	Fault Filter Period These bits represent the sampling period (in IPBus clock cycles) of the fault pin input filter. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in Section 28.3.4.3.1, Input filter considerations.

28.3.4.3.1 Input filter considerations

The FILT_PER value should be set such that the sampling period is larger than the period of the expected noise. This way a noise spike only corrupts one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT + 3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of ((FILT_CNT + 4) × FILT_PER × IPBus clock period). Note that even when the filter is enabled, there is a combinational path to disable the PWM outputs. This is to ensure rapid response to fault conditions and also to ensure fault response if the PWM module loses its clock. The latency induced by the filter is seen in the time to set the FFLAG and FFPIN bits of the FSTS register.

28.4 Functional description

28.4.1 Block diagram

A block diagram of the PWM is shown in Figure 28-1.

28.4.2 PWM capabilities

This section describes some capabilities of the PWM module.

28.4.2.1 Center-aligned PWMs

Each submodule has its own timer that is capable of generating PWM signals on two output pins. The edges of each of these signals are controlled independently as shown in Figure 28-35.

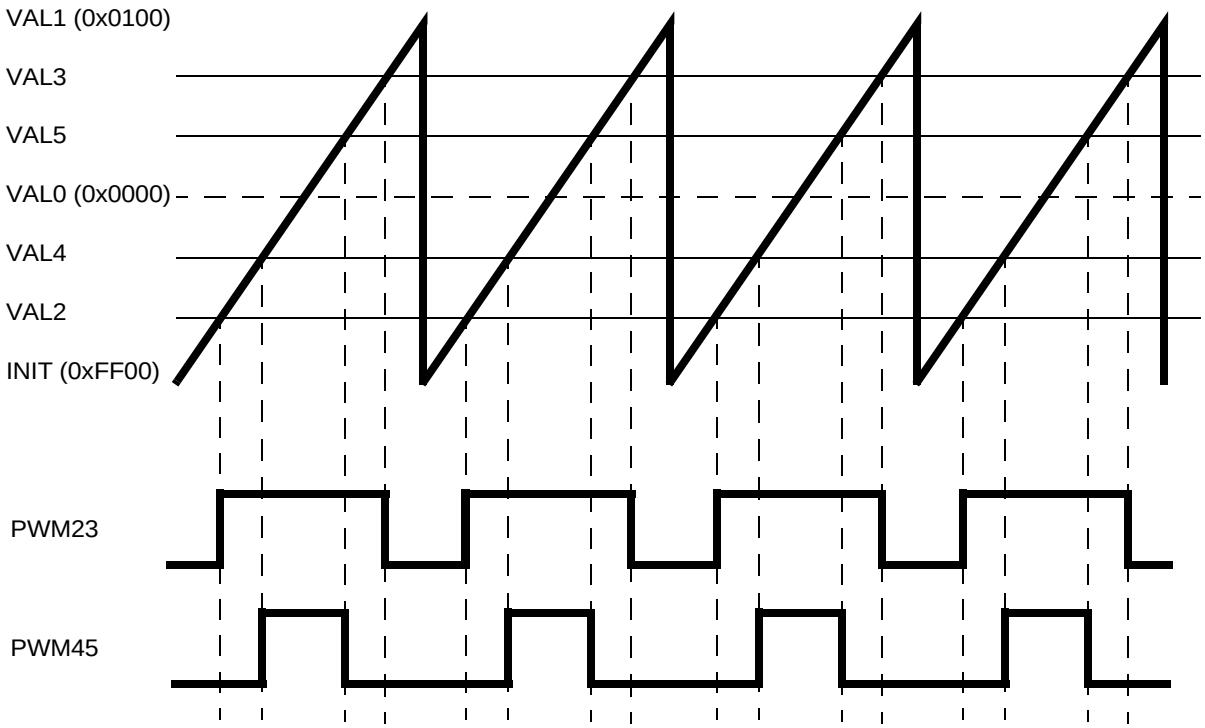


Figure 28-35. Center-aligned example

The submodule timers only count in the up direction and then reset to the INIT value. Instead of having a single value that determines pulse width, there are two values that must be specified: the turn on edge and

the turn off edge. This double action edge generation not only gives the user control over the pulse width, but over the relative alignment of the signal as well. As a result, there is no need to support separate PWM alignment modes since the PWM alignment mode is inherently a function of the turn on and turn off edge values.

Figure 28-35 also illustrates an additional enhancement to the PWM generation process. When the counter resets, it is reloaded with a user-specified value, which may or may not be zero. If the value chosen is the 2's complement of the modulus value, then the PWM generator operates in "signed" mode. This means that if each PWM's turn on and turn off edge values are also the same number but only different in their sign, the "on" portion of the output signal is centered around a count value of zero. Therefore, only one PWM value needs to be calculated in software and then this value and its negative are provided to the submodule as the turn off and turn on edges respectively. This technique results in a pulse width that always consists of an odd number of timer counts. If all PWM signal edge calculations follow this same convention, then the signals are center-aligned with respect to each other, which is the goal. Of course, center alignment between the signals is not restricted to symmetry around the zero count value, as any other number would also work. However, centering on zero provides the greatest range in signed mode and also simplifies the calculations.

28.4.2.2 Edge-aligned PWMs

When the turn on edge for each pulse is specified to be the INIT value, then edge-aligned operation results, as illustrated in Figure 28-36. Therefore, only the turn off edge value needs to be periodically updated to change the pulse width.

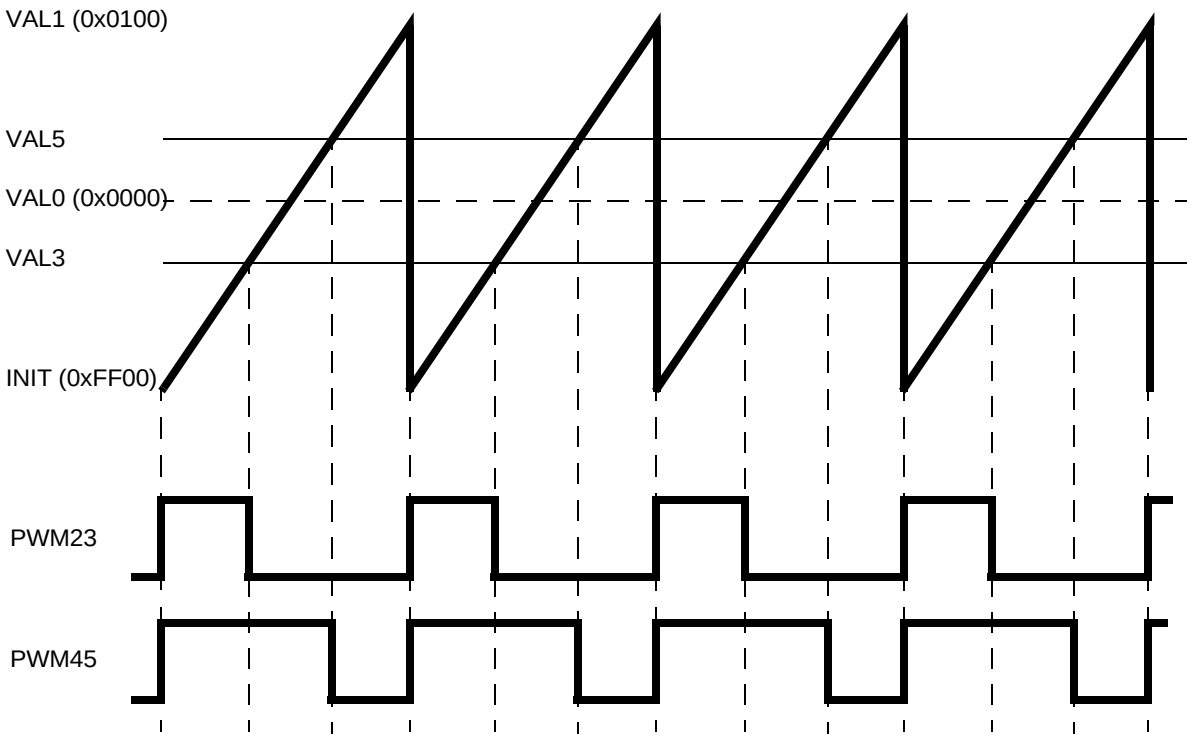


Figure 28-36. Edge-aligned example (INIT=VAL2=VAL4)

With edge-aligned PWMs, another example of the benefits of signed mode can be seen. A common way to drive an H-bridge is to use a technique called “bipolar” PWMs where a 50% duty cycle results in zero volts on the load. Duty cycles less than 50% result in negative load voltages and duty cycles greater than 50% generate positive load voltages. If the module is set to signed mode operation (the INIT and VAL1 values are the same number with opposite signs), then there is a direct proportionality between the PWM turn off edge value and the motor voltage, including the sign. So once again, signed mode of operation simplifies the software interface to the PWM module since no offset calculations are required to translate the output variable control algorithm to the voltage on an H-Bridge load.

28.4.2.3 Phase-shifted PWMs

In the previous sections, the benefits of signed mode of operation were discussed in the context of simplifying the required software calculations by eliminating the requirement to bias up signed variables before applying them to the module. However, if numerical biases are applied to the turn on and turn off edges of different PWM signal, the signals are phase-shifted with respect to each other, as illustrated in Figure 28-37. This results in certain advantages when applied to a power stage. For example, when operating a multi-phase inverter at a low modulation index, all of the PWM switching edges from the different phases occur at nearly the same time. This can be troublesome from a noise standpoint, especially if ADC readings of the inverter must be scheduled near those times. Phase shifting the PWM signals can open up timing windows between the switching edges to allow a signal to be sampled by the ADC. However, phase shifting does not affect the duty cycle, so average load voltage is not affected.

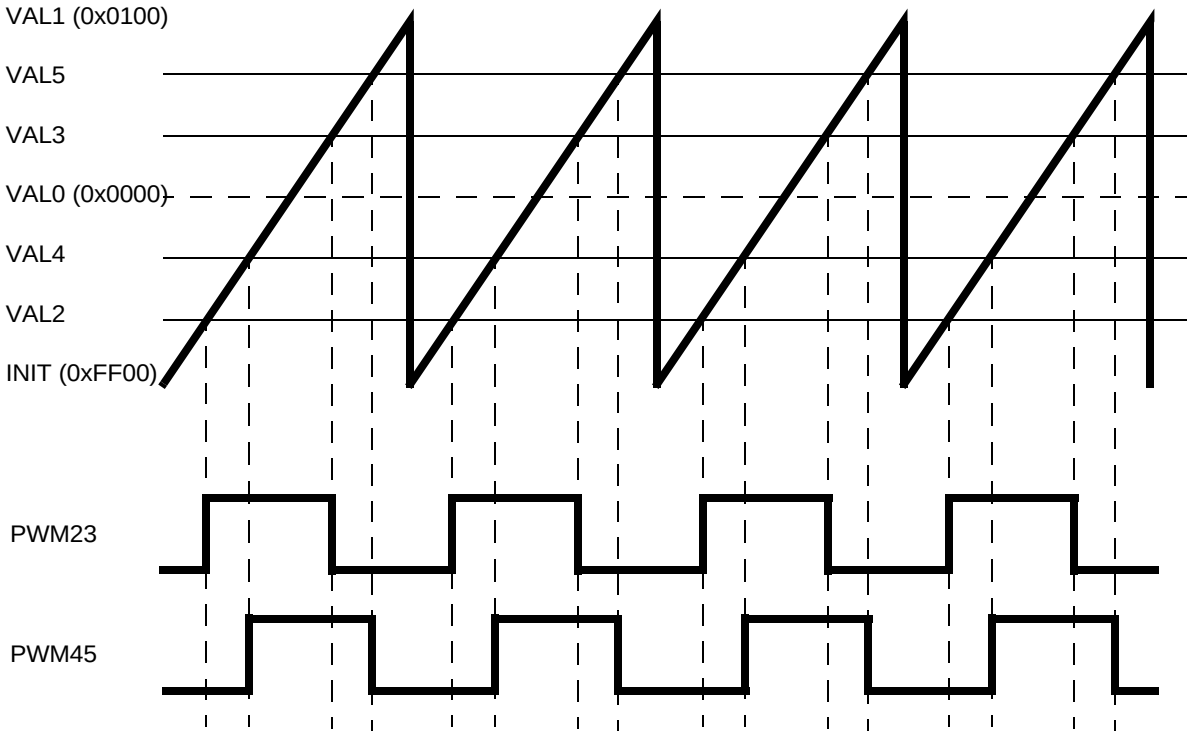


Figure 28-37. Phase-shifted outputs example

An additional benefit of phase-shifted PWMs can be seen in Figure 28-38. In this case, an H-Bridge circuit is driven by 4 PWM signals to control the voltage waveform on the primary of a transformer. Both left and

right side PWMs are configured to always generate a square wave with 50% duty cycle. This works out nicely for the H-Bridge since no narrow pulse widths are generated, reducing the high frequency switching requirements of the transistors. Notice that the square wave on the right side of the H-Bridge is phase-shifted compared to the left side of the H-Bridge. As a result, the transformer primary sees the bottom waveform across its terminals. The RMS value of this waveform is directly controlled by the amount of phase shift of the square waves. Regardless of the phase shift, no DC component appears in the load voltage as long as the duty cycle of each square wave remains at 50% making this technique ideally suited for transformer loads. As a result, this topology is frequently used in industrial welders to adjust the amount of energy delivered to the weld arc.

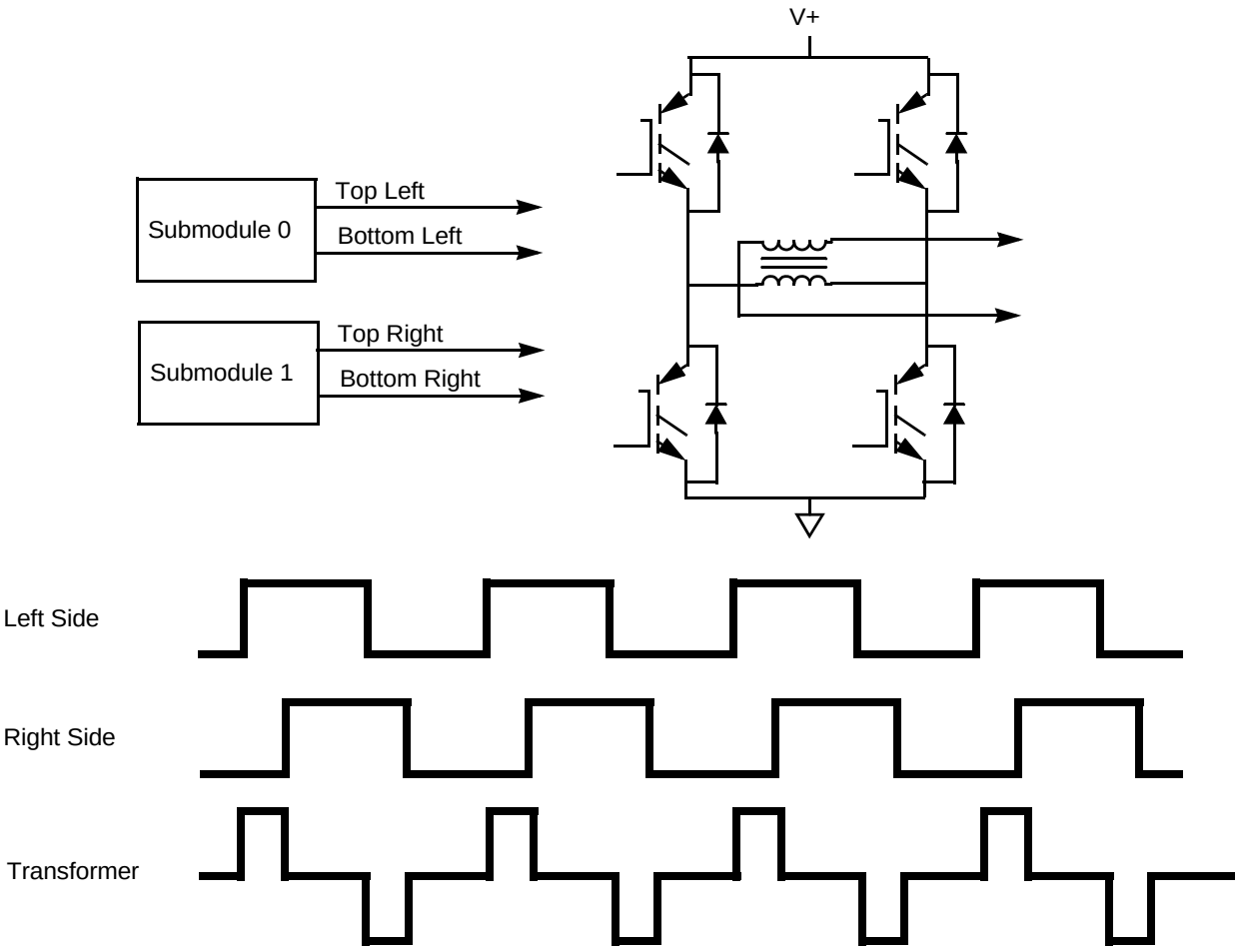


Figure 28-38. Phase-shifted PWMs applied to a transformer primary

28.4.2.4 Double switching PWMs

Double switching PWM output is supported to aid in single shunt current measurement and three phase reconstruction. This method supports two independent rising edges and two independent falling edges per PWM cycle. The VAL2 and VAL3 registers are used to generate the even channel (labelled as PWMA in the figure) while VAL4 and VAL5 are used to generate the odd channel. The two channels are combined using XOR logic (see Figure 28-49) as shown in Figure 28-39. The DBLPWM signal can be run through the deadtime insertion logic.

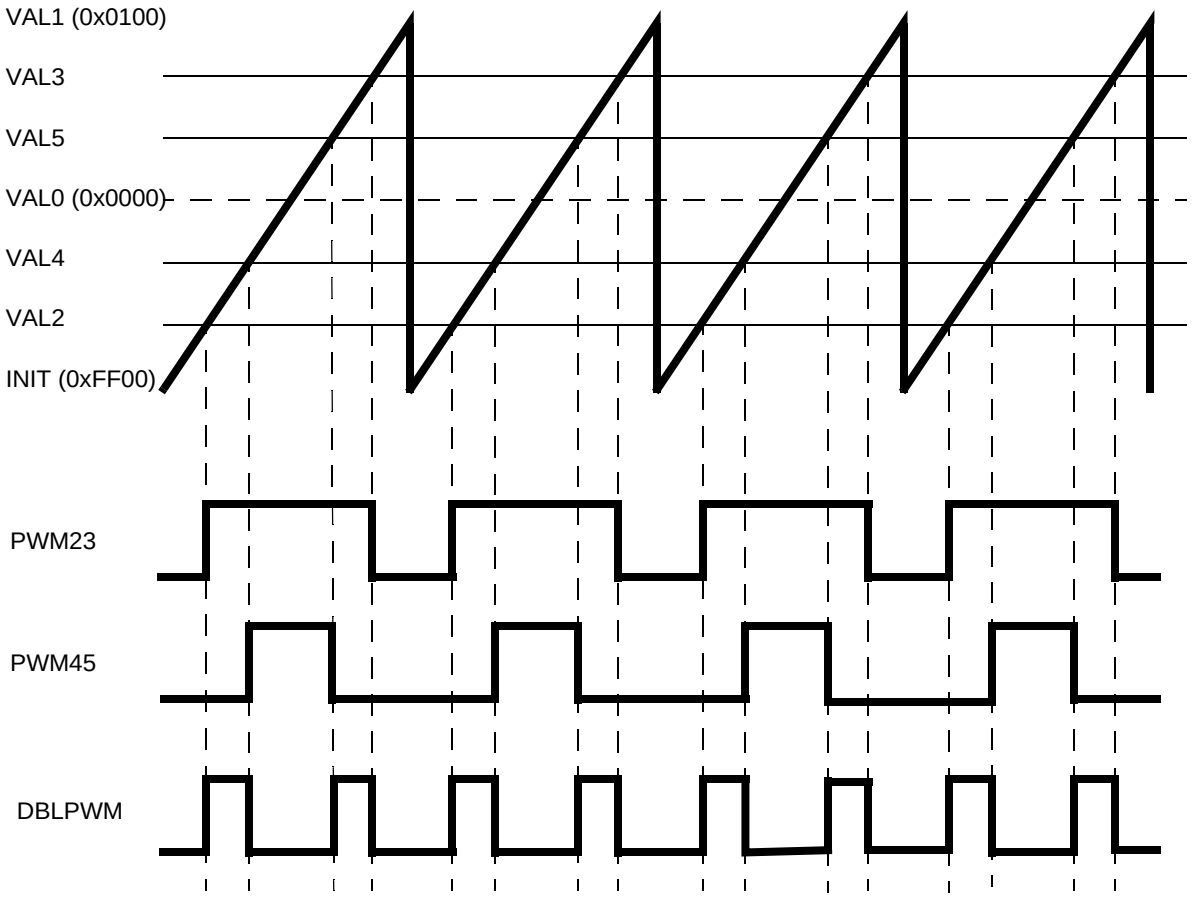


Figure 28-39. Double switching output example

28.4.2.5 ADC Triggering

In cases where the timing of the ADC triggering is critical, it must be scheduled as a hardware event instead of software activated. With this PWM module, multiple ADC triggers can be generated in hardware per PWM cycle without the requirement of another timer module. Figure 28-40 shows how this is accomplished. When specifying complementary mode of operation, only two edge comparators are required to generate the output PWM signals for a given submodule. This means that the other comparators are free to perform other functions. In this example, the software doesn't have to quickly respond after the first conversion to set up other conversions that must occur in the same PWM cycle.

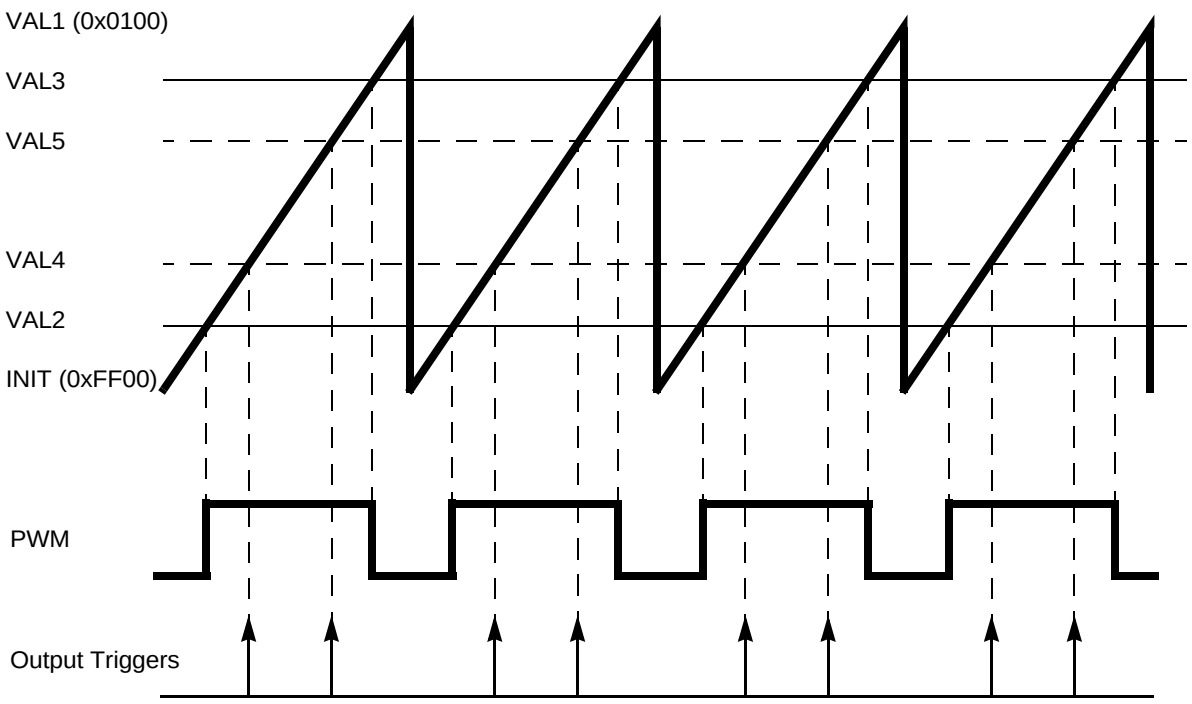


Figure 28-40. Multiple output trigger generation in hardware

Since each submodule has its own timer, it is possible for each submodule to run at a different frequency. One of the options possible with this PWM module is to have one or more submodules running at a lower frequency, but still synchronized to the timer in submodule0. Figure 28-41 shows how this feature can be used to schedule ADC triggers over multiple PWM cycles. A suggested use for this configuration would be to use the lower frequency submodule to control the sampling frequency of the software control algorithm where multiple ADC triggers can now be scheduled over the entire sampling period. In Figure 28-41, ALL submodule comparators are shown being used for ADC trigger generation.

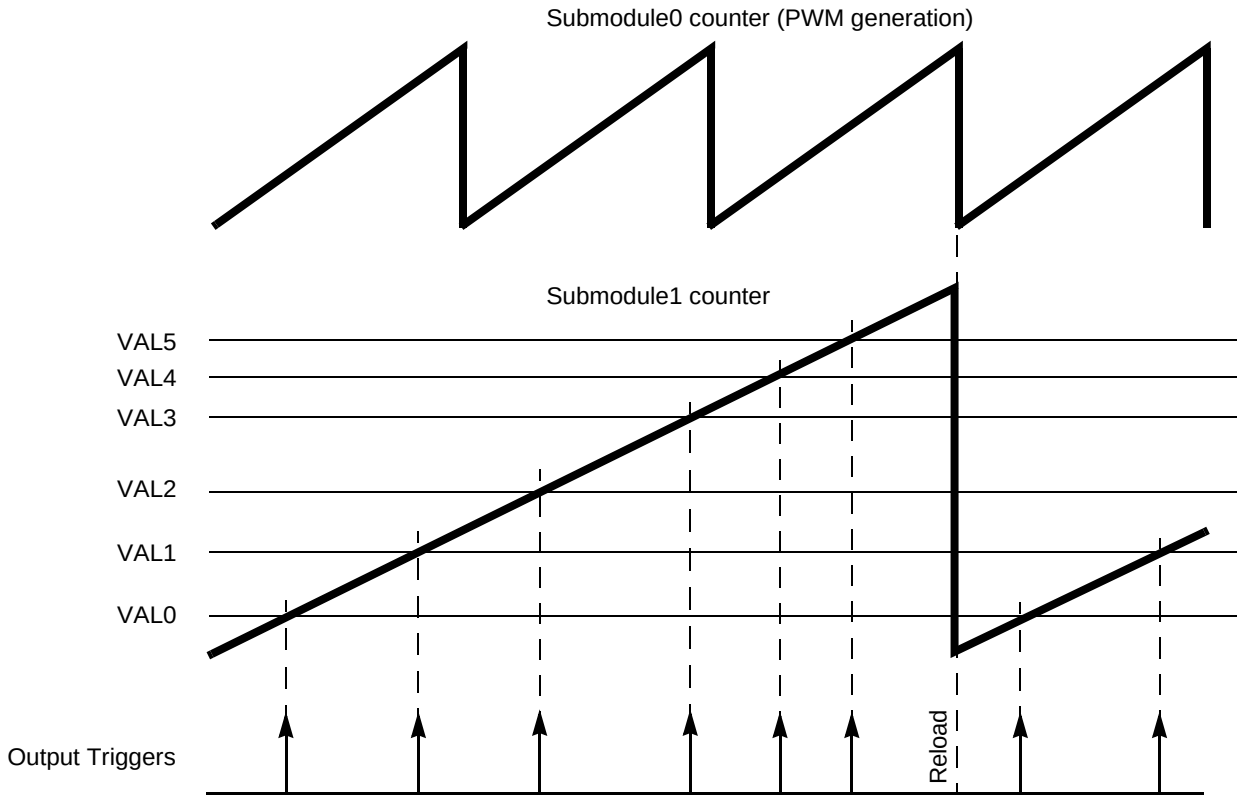


Figure 28-41. Multiple output triggers over several PWM cycles

28.4.2.6 Enhanced capture capabilities (E-Capture)

When a PWM pin is not being used for PWM generation, it can be used to perform input captures. Recall that for PWM generation BOTH edges of the PWM signal are specified via separate compare register values. When programmed for input capture, both of these registers work on the same pin to capture multiple edges, toggling from one to the other in either a free-running or one-shot fashion. By programming the desired edge of each capture circuit, period and pulse width of an input signal can be measured without the requirement to re-arm the circuit. In addition, each edge of the input signal can clock an 8-bit counter where the counter output is compared to a user-specified value (EDGCMP). When the counter output equals EDGCMP, the value of the submodule timer is captured and the counter is automatically reset. This feature allows the module to count a specified number of edge events and then perform a capture and interrupt. Figure 28-42 illustrates some of the functionality of the E-Capture circuit.

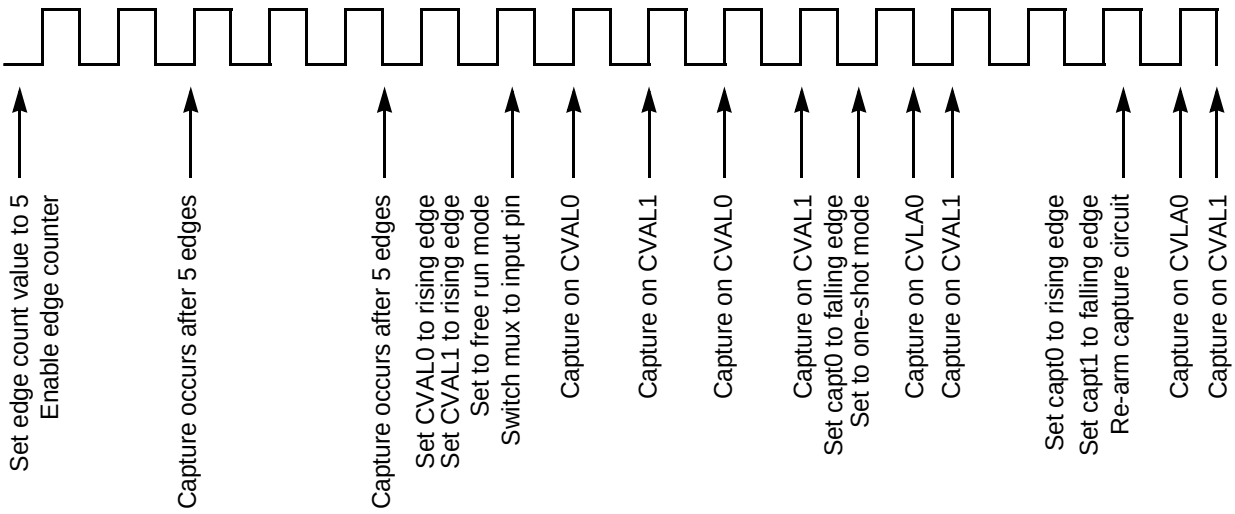


Figure 28-42. Capture capabilities of the E-Capture circuit

When a submodule is being used for PWM generation, its timer counts up to the modulus value used to specify the PWM frequency and then is re-initialized. Therefore, using this timer for input captures on one of the other pins (e.g, PWMX) has limited utility since it does not count through all of the numbers and the timer reset represents a discontinuity in the 16-bit number range. However, when measuring a signal that is synchronous to the PWM frequency, the timer modulus range is perfectly suited for the application. As an example, consider Figure 28-43. In this application the output of a PWM power stage is connected to the PWMX pin that is configured for free-running input captures. Specifically, the CVAL0 capture circuitry is programmed for rising edges and the CVAL1 capture circuitry is set for falling edges. This results in new load pulse width data being acquired every PWM cycle. To calculate the pulse width, subtract the CVAL0 register value from the CVAL1 register value. This measurement is extremely beneficial when performing dead-time distortion correction on a half bridge circuit driving an inductive load. Also, these values can be directly compared to the VAL*x* registers responsible for generating the PWM outputs to obtain a measurement of system propagation delays.

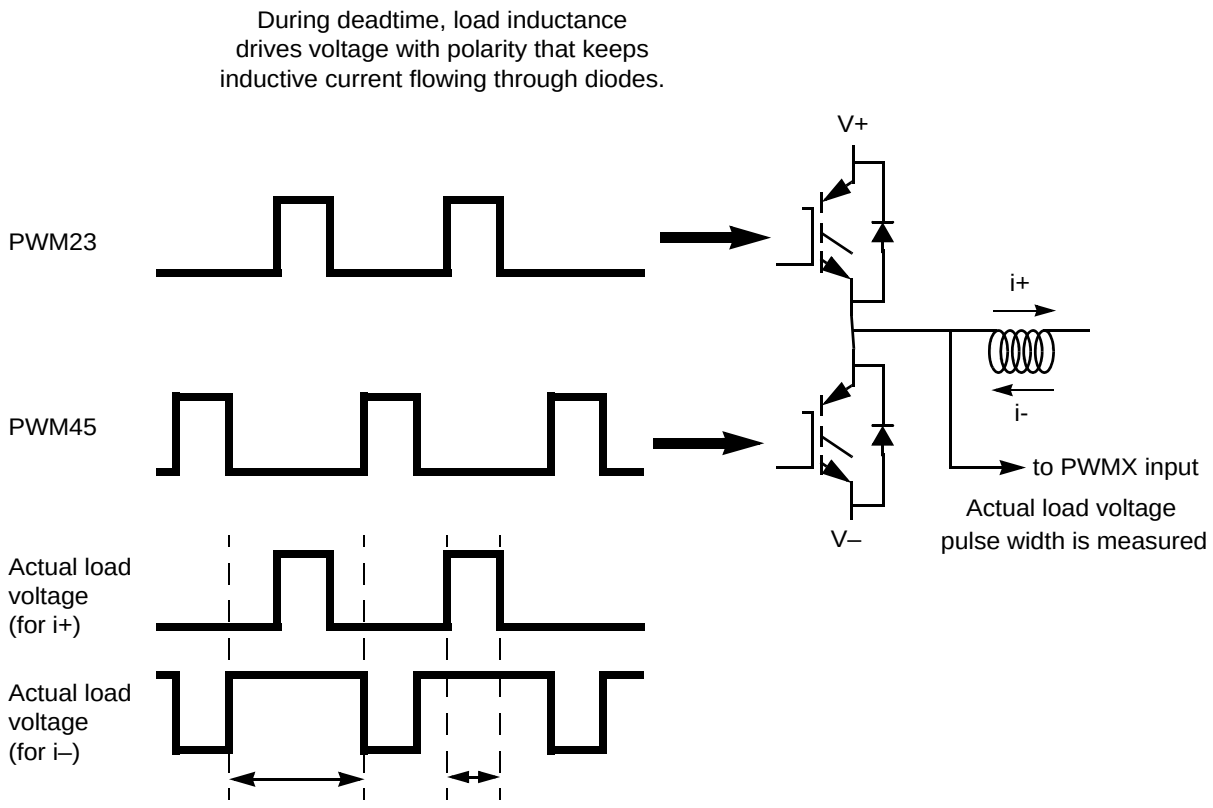


Figure 28-43. Output pulse width measurement possible with the E-Capture circuit

28.4.2.7 Synchronous switching of multiple outputs

Before the PWM signals are routed to the output pins, they are processed by a hardware block that permits all submodule outputs to be switched synchronously. This feature can be extremely useful in commutated motor applications where the next commutation state can be laid in ahead of time and then immediately switched to the outputs when the appropriate condition or time is reached. Not only do all the changes occur synchronously on all submodule outputs, but they occur immediately after the trigger event occurs, eliminating any interrupt latency.

The synchronous output switching is accomplished via a signal called FORCE_OUT. This signal originates from the local FORCE bit within the submodule, from submodule0, or from external to the PWM module and, in most cases, is supplied from an external timer channel configured for output compare. In a typical application, software sets up the desired states of the output pins in preparation for the next FORCE_OUT event using the DTSRCSEL[SEL23 and SEL45] bits. This selection lays dormant until the FORCE_OUT signal transitions and then all outputs are switched simultaneously. The signal switching is performed upstream from the deadtime generator so that any abrupt changes that might occur do not violate deadtime on the power stage when in complementary mode.

Figure 28-44 shows a popular application that can benefit from this feature. On a brushless DC motor it is desirable in many cases to spin the motor without need of Hall-effect sensor feedback. Instead, the back EMF of the motor phases is monitored and this information is used to schedule the next commutation event. The top waveforms of Figure 28-44 are a simplistic representation of these back EMF signals. Timer

compare events (represented by the long vertical lines in the diagram) are scheduled based on the zero crossings of the back-EMF waveforms. The PWM module is configured via software ahead of time with the next state of the PWM pins in anticipation of the compare event. When it happens, the output compare of the timer drives the FORCE_OUT signal, which immediately changes the state of the PWM pins to the next commutation state with no software latency.

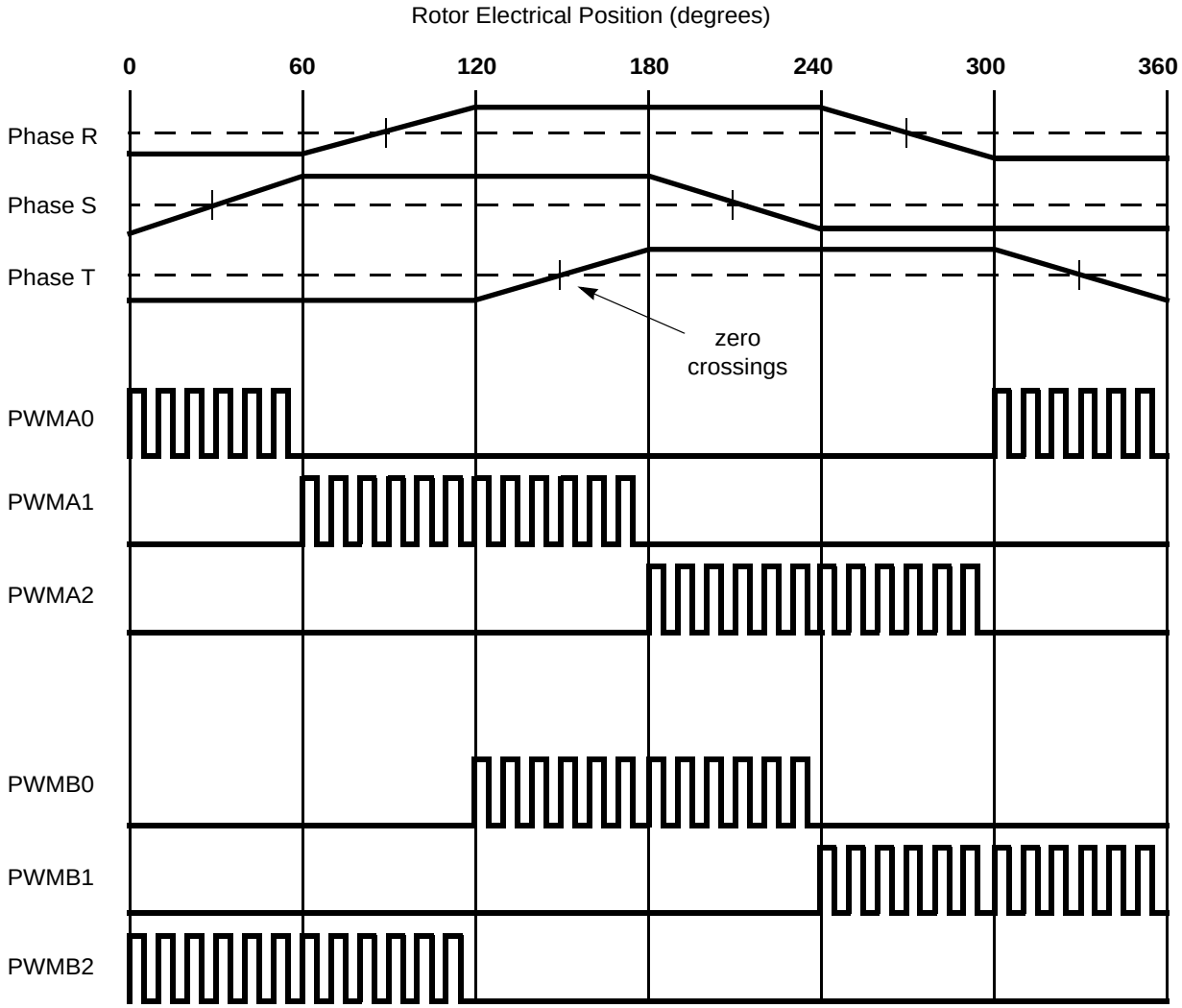


Figure 28-44. Sensorless BLDC commutation using the force out function

28.4.3 Functional details

This section describes the implementation of various sections of the PWM in greater detail.

28.4.3.1 PWM clocking

Figure 28-45 shows the logic used to generate the main counter clock. Each submodule can select between three clock signals: the IPBus clock, EXT_CLK, and AUX_CLK. The EXT_CLK is generated by an

on-chip resource such as a timer module and goes to all of the submodules. The AUX_CLK signal is broadcast from submodule0 and can be selected as the clock source by other submodules so that the 8-bit prescaler and RUN bit from submodule0 can control all of the submodules. When AUX_CLK is selected as the source for the submodule clock, the RUN bit from submodule0 is used instead of the local RUN bit from this submodule.

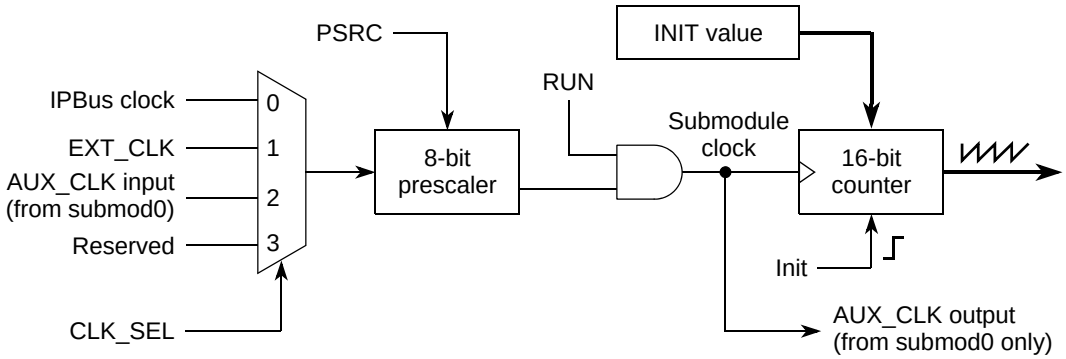


Figure 28-45. Clocking block diagram for each PWM submodule

To permit lower PWM frequencies, the prescaler produces the PWM clock frequency by dividing the IPBus clock frequency by 1–128. The prescaler bits, PRSC, in the control register (CTRL1), select the prescaler divisor. This prescaler is buffered and is not used by the PWM generator until the LDOK bit is set and a new PWM reload cycle begins or LDMOD is set.

28.4.3.2 Register reload logic

The register reload logic determines when the outer set of registers for all double-buffered register pairs are transferred to the inner set of registers. The register reload event can be scheduled to occur every *n* PWM cycles using the LDFQ bits and the FULL bit. A half cycle reload option is also supported (HALF) where the reload can take place in the middle of a PWM cycle. The half cycle point is defined by the VAL0 register and does not have to be exactly in the middle of the PWM cycle.

As illustrated in Figure 28-46 the reload signal from submodule0 can be broadcast as the Master Reload signal, allowing the reload logic from submodule0 to control the reload of registers in other submodules.

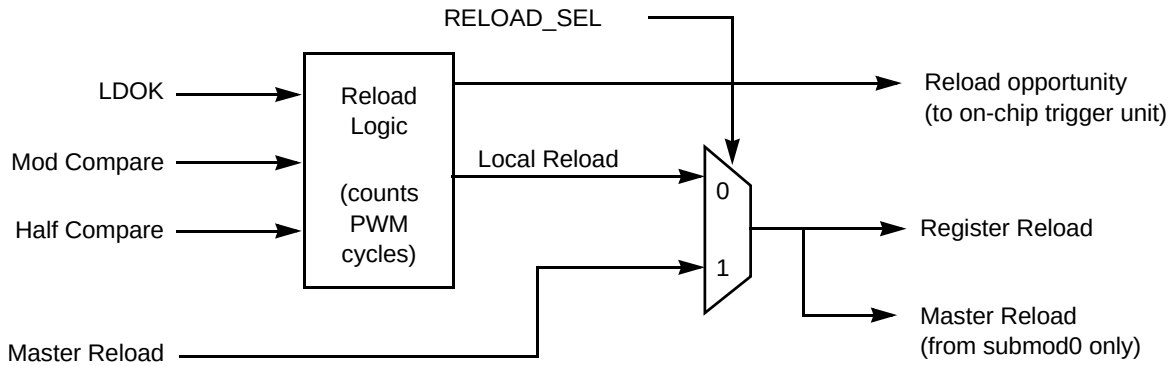


Figure 28-46. Register reload logic

28.4.3.3 Counter synchronization

Referring to Figure 28-47, the 16-bit counter counts up until its output equals VAL1, which specifies the counter modulus value. The resulting compare causes a rising edge to occur on the Local Sync signal, which is one of four possible sources used to cause the 16-bit counter to be initialized with INIT. If Local Sync is selected as the counter initialization signal, then VAL1 within the submodule effectively controls the timer period (and thus the PWM frequency generated by that submodule) and everything works on a local level.

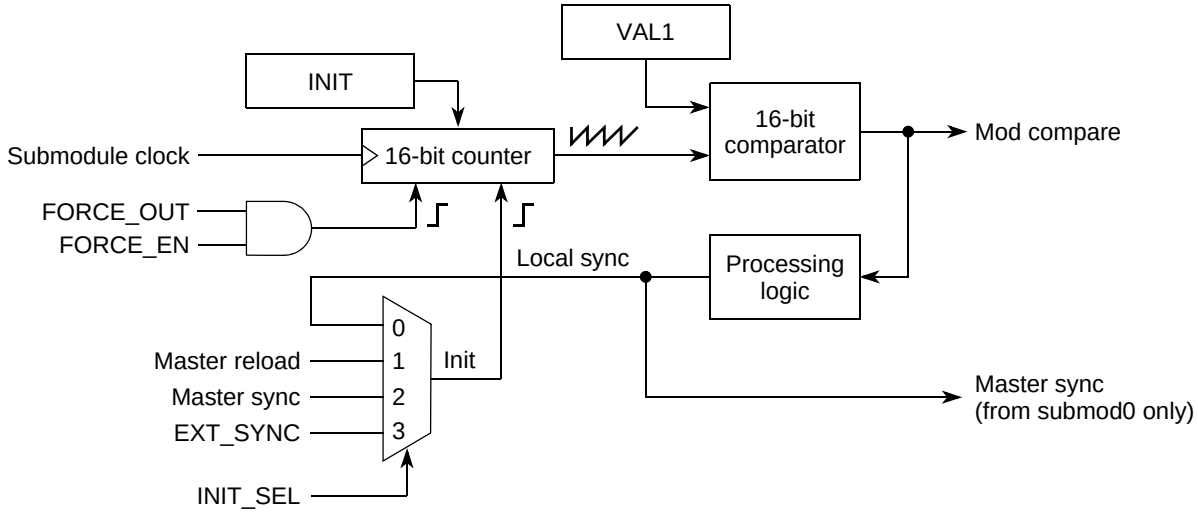


Figure 28-47. Submodule timer synchronization

The Master Sync signal originates as the Local Sync from submodule0. If configured to do so, the timer period of any submodule can be locked to the period of the timer in submodule0. The VAL1 register and associated comparator of the other submodules can then be freed up for other functions such as PWM generation, input captures, output compares, or output triggers.

The EXT_SYNC signal originates on chip or off chip depending on the system architecture. This signal may be selected as the source for counter initialization so that an external source can control the period of all submodules.

If the Master Reload signal is selected as the source for counter initialization, then the period of the counter is locked to the register reload frequency of submodule0. Since the reload frequency is usually commensurate to the sampling frequency of the software control algorithm, the submodule counter period therefore equals the sampling period. As a result, this timer can be used to generate output compares or output triggers over the entire sampling period, which may consist of several PWM cycles. The Master Reload signal can only originate from submodule0.

The counter can optionally initialize upon the assertion of the FORCE_OUT signal, assuming that the FORCE_EN bit is set. As indicated by Figure 28-47, this constitutes a second init input into the counter, which causes the counter to initialize regardless of which signal is selected as the counter init signal. The FORCE_OUT signal is provided mainly for commutated applications. When PWM signals are commutated on an inverter controlling a brushless DC motor, it is necessary to restart the PWM cycle at the beginning of the commutation interval. This action effectively resynchronizes the PWM waveform to the commutation timing. Otherwise, the average voltage applied to a motor winding integrated over the

entire commutation interval is a function of the timing between the asynchronous commutation event with respect to the PWM cycle. The effect is more critical at higher motor speeds where each commutation interval may consist of only a few PWM cycles. If the counter is not initialized at the start of each commutation interval, the result is an oscillation caused by the beating between the PWM frequency and the commutation frequency.

28.4.3.4 PWM generation

Figure 28-48 illustrates how PWM generation is accomplished in each submodule. In each case, two comparators and associated VAL*x* registers are utilized for each PWM output signal. One comparator and VAL*x* register is used to control the turn on edge while a second comparator and VAL*x* register control the turn off edge.

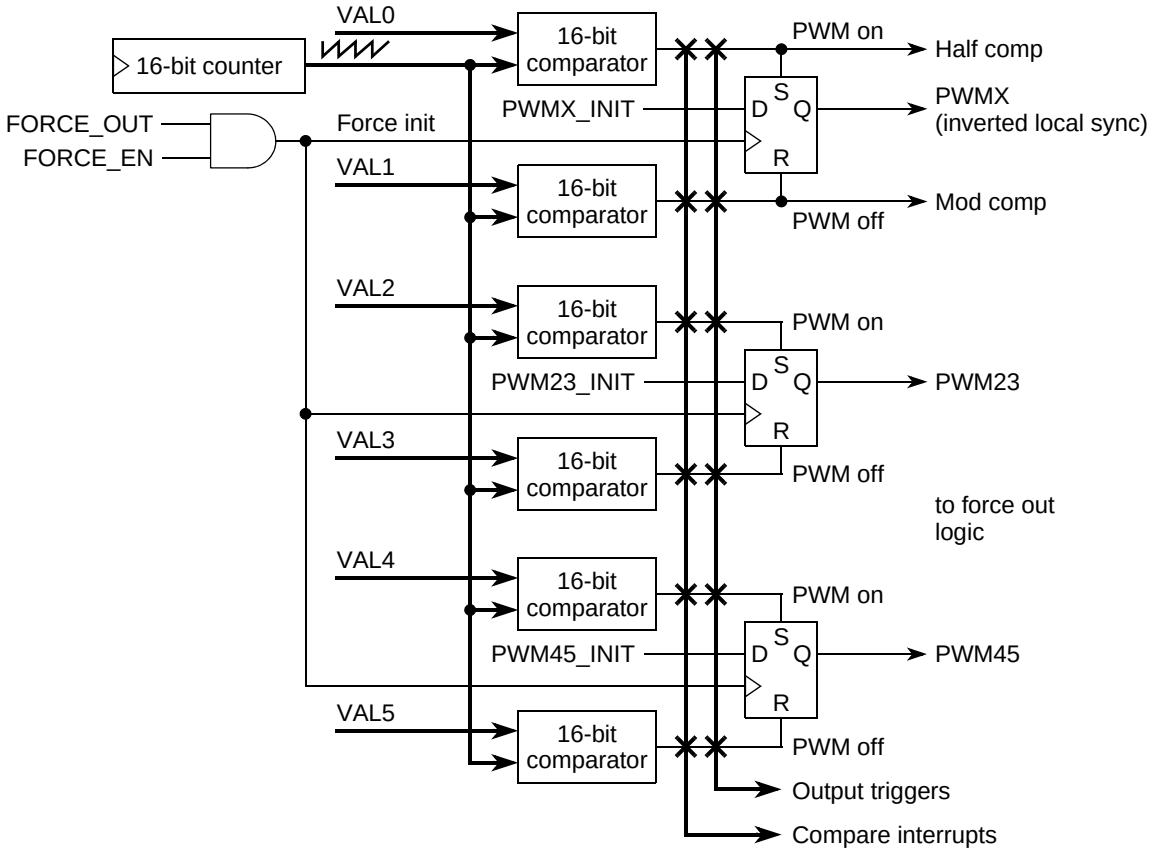


Figure 28-48. PWM generation hardware

The generation of the Local Sync signal is performed exactly the same way as the other PWM signals in the submodule. While comparator 0 causes a rising edge of the Local Sync signal, comparator 1 generates a falling edge. Comparator 1 is also hardwired to the reload logic to generate the half cycle reload indicator.

If VAL1 is controlling the modulus of the counter and VAL0 is half of the VAL1 register minus the INIT value, then the half cycle reload pulse occurs exactly halfway through the timer count period, and the Local Sync has a 50% duty cycle. On the other hand, if the VAL1 and VAL0 registers are not required for register reloading or counter initialization, they can be used to modulate the duty cycle of the Local Sync signal, effectively turning it into an auxiliary PWM signal (PWMX) assuming that the PWMX pin is not being

used for another function such as input capture or deadtime distortion correction. Including the Local Sync signal, each submodule is capable of generating 3 PWM signals where software has complete control over each edge of each of the signals.

If the comparators and edge value registers are not required for PWM generation, they can also be used for other functions such as output compares, generating output triggers, or generating interrupts at timed intervals.

The 16-bit comparators shown in Figure 28-48 are “equal to or greater than” not just “equal to” comparators. In addition, if both the set and reset of the flip-flop are both asserted, then the flop output goes to 0.

28.4.3.5 Output compare capabilities

By using the VAL*x* registers in conjunction with the submodule timer and 16-bit comparators, buffered output compare functionality can be achieved with no additional hardware required. Specifically, the following output compare functions are possible:

- An output compare sets the output high
- An output compare sets the output low
- An output compare generates an interrupt
- An output compare generates an output trigger

Referring again to Figure 28-48, an output compare is initiated by programming a VAL*x* register for a timer compare, which in turn causes the output of the D flip-flop to either set or reset. For example, if an output compare is desired on the PWMA signal that sets it high, VAL2 would be programmed with the counter value where the output compare should take place. However, to prevent the D flip-flop from being reset again after the compare has occurred, the VAL3 register must be programmed to a value outside of the modulus range of the counter. Therefore, a compare that would result in resetting the D flip-flop output would never occur. Conversely, if an output compare is desired on the PWMA signal that sets it low, the VAL3 register is programmed with the appropriate count value and the VAL2 register is programmed with a value outside the counter modulus range. Regardless of whether a high compare or low compare is programmed, an interrupt or output trigger can be generated when the compare event occurs.

28.4.3.6 Force out logic

For each submodule, software can select between the following signal sources for the FORCE_OUT signal:

- The local FORCE bit
- The Master Force signal from submodule0
- The local Reload signal
- The Master Reload signal from submodule0
- The Local Sync signal
- The Master Sync signal from submodule0
- The EXT_FORCE signal from on or off chip depending on the chip architecture

The local signals are used when the user wants to change the signals on the output pins of the submodule without regard for synchronization with other submodules. However, if it is required that all signals on all submodule outputs change at the same time, the Master signals or EXT_FORCE signal should be selected.

Figure 28-49 illustrates the Force logic. The SEL23 and SEL45 fields each choose from one of three signals that can be supplied to the submodule outputs: the PWM signal, the inverted PWM signal, or a binary level specified by software via the OUT23 and OUT45 bits. The selection can be determined ahead of time and, when a FORCE_OUT event occurs, these values are presented to the signal selection mux, which immediately switches the requested signal to the output of the mux for further processing downstream.

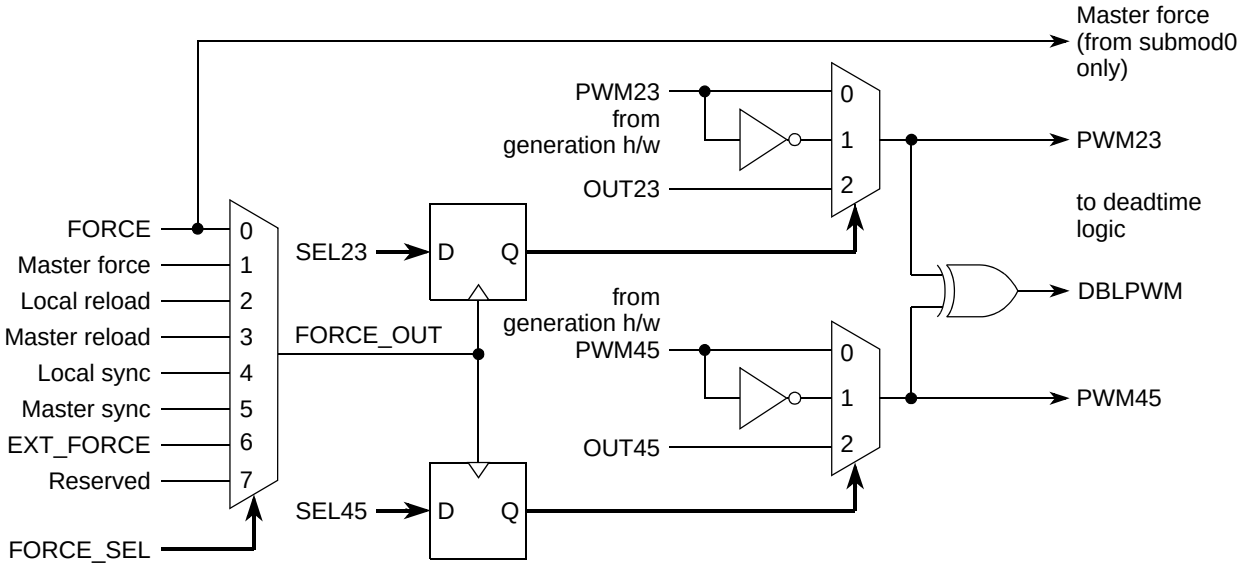


Figure 28-49. Force out logic

The local FORCE signal of submodule0 can be broadcast as the Master Force signal to other submodules. This feature allows the local FORCE bit of submodule0 to synchronously update all of the submodule outputs at the same time. The EXT_FORCE signal originates from outside the PWM module from a source such as a timer or digital comparators in the Analog-to-Digital Converter.

28.4.3.7 Independent or complementary channel operation

Writing a logic 1 to the INDEP bit of the CNFG register configures the pair of PWM outputs as two independent PWM channels. Each PWM output is controlled by its own VAL*x* pair operating independently of the other output.

Writing a logic 0 to the INDEP bit configures the PWM output as a pair of complementary channels. The PWM pins are paired as shown in Figure 28-50 in complementary channel operation.

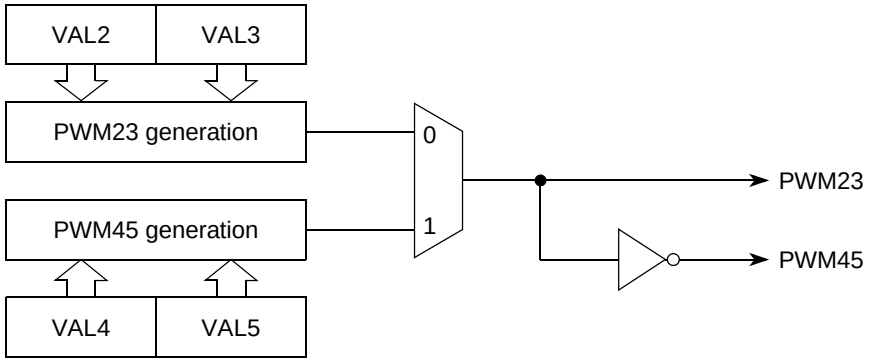


Figure 28-50. Complementary channel pair

The complementary channel operation is for driving top and bottom transistors in a motor drive circuit, such as the one in Figure 28-51.

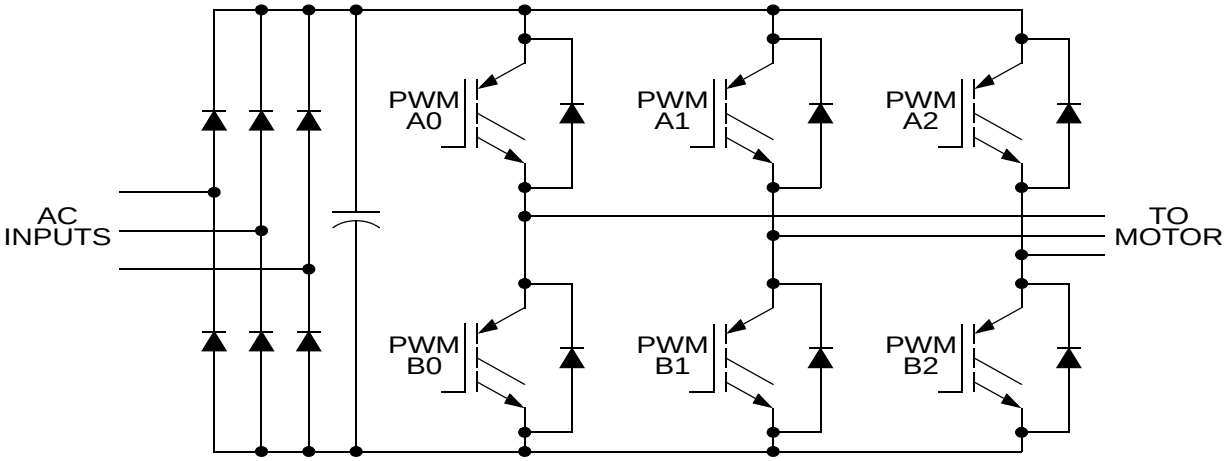


Figure 28-51. Typical 3-phase AC motor drive

Complementary operation allows the use of the deadtime insertion feature.

28.4.3.8 Deadtime insertion logic

Figure 28-52 shows the deadtime insertion logic of each submodule, which creates non-overlapping complementary signals when not in independent mode.

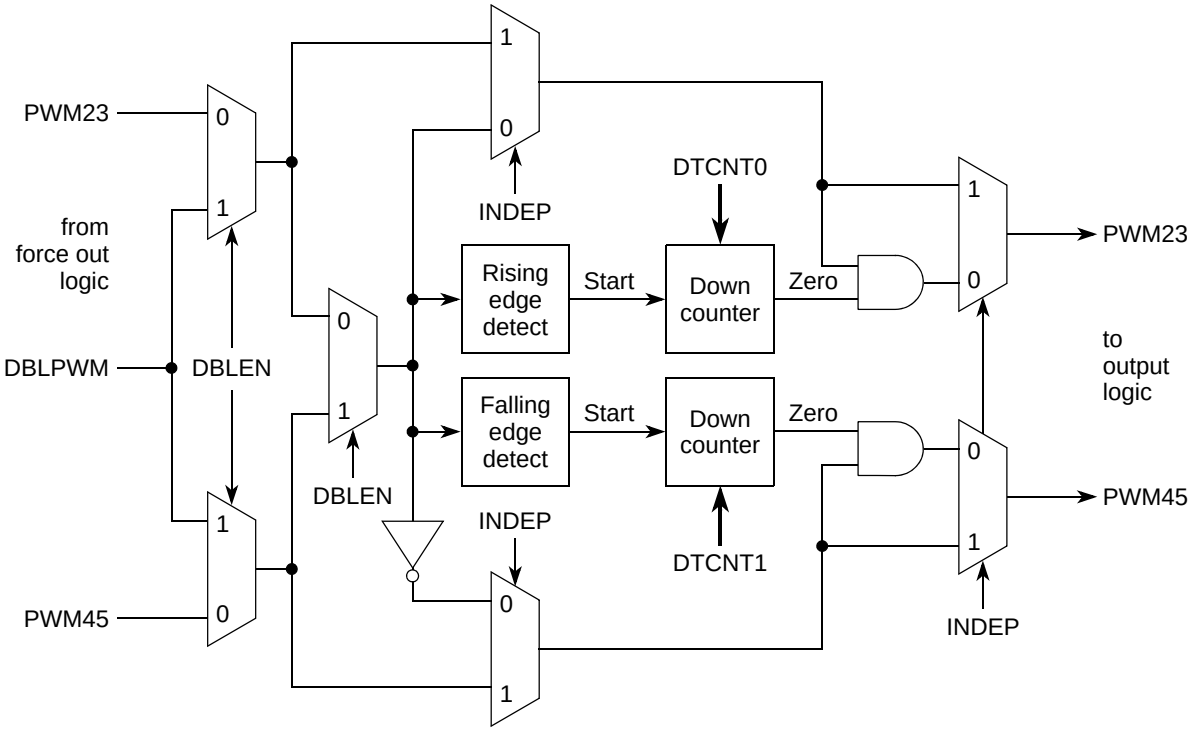


Figure 28-52. Deadtime insertion and fine control logic

While in the complementary mode, a PWM pair can be used to drive top/bottom transistors, as shown in Figure 28-52. When the top PWM channel is active, the bottom PWM channel is inactive, and vice versa.

NOTE

To avoid short-circuiting the DC bus and endangering the transistor, there must be no overlap of conducting intervals between top and bottom transistor. But the transistor's characteristics may make its switching-off time longer than switching-on time. To avoid the conducting overlap of top and bottom transistors, deadtime needs to be inserted in the switching period, as illustrated in Figure 28-53.

The deadtime generators automatically insert software-selectable activation delays into the pair of PWM outputs. The deadtime registers (DTCNT0 and DTCNT1) specify the number of IPBus clock cycles to use for deadtime delay. Every time the deadtime generator inputs change state, deadtime is inserted. Deadtime forces both PWM outputs in the pair to the inactive state.

When deadtime is inserted in complementary PWM signals connected to an inverter driving an inductive load, the PWM waveform on the inverter output has a different duty cycle than what appears on the output pins of the PWM module. This results in a distortion in the voltage applied to the load. A method of correcting this, adding to or subtracting from the PWM value used, is discussed next.

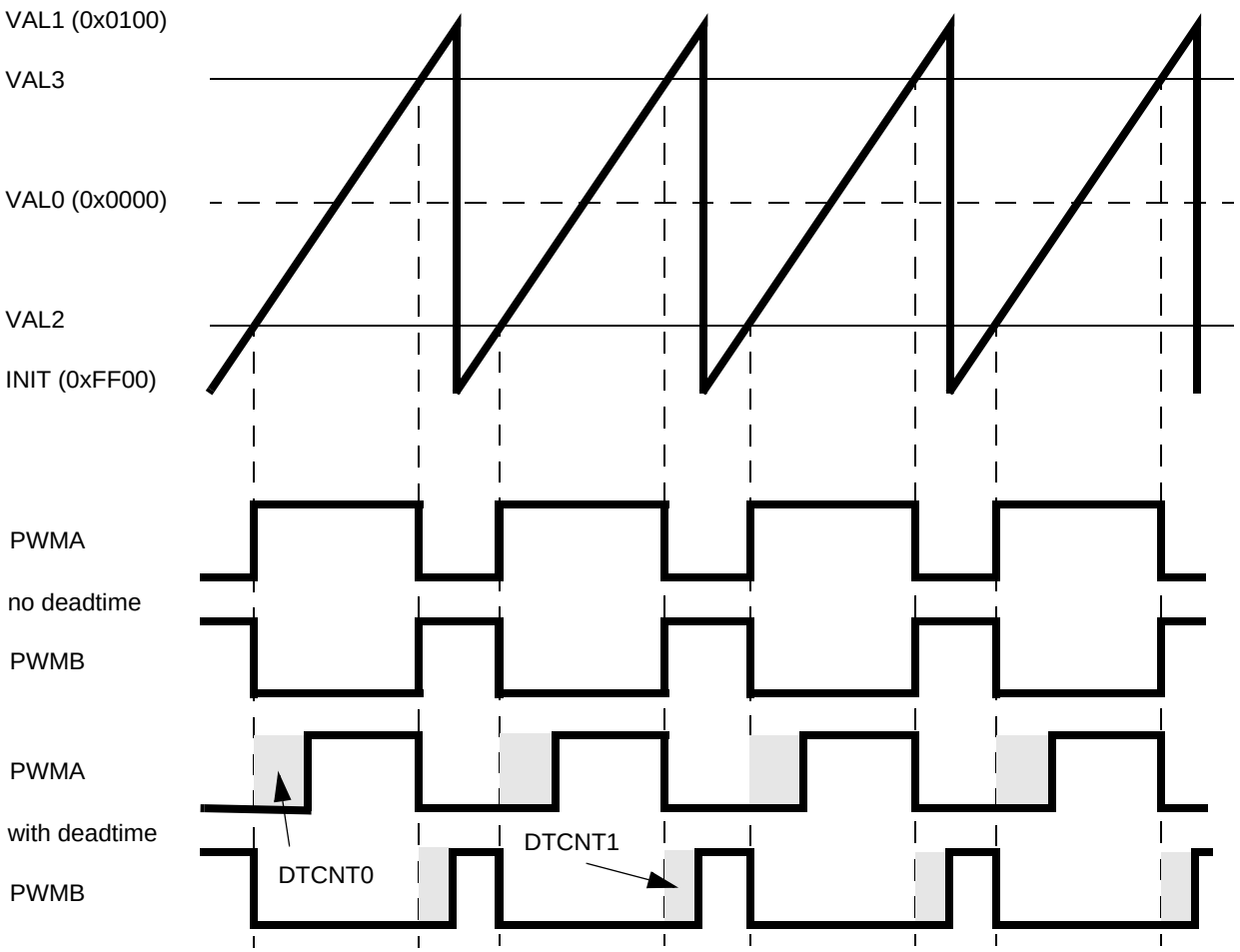


Figure 28-53. Deadtime insertion

28.4.3.8.1 Top/bottom correction

In complementary mode, either the top or the bottom transistor controls the output voltage. However, deadtime has to be inserted to avoid overlap of conducting interval between the top and bottom transistor. Both transistors in complementary mode are off during deadtime, allowing the output voltage to be determined by the current status of load and introduce distortion in the output voltage. See Figure 28-54. On AC induction motors running open-loop, the distortion typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

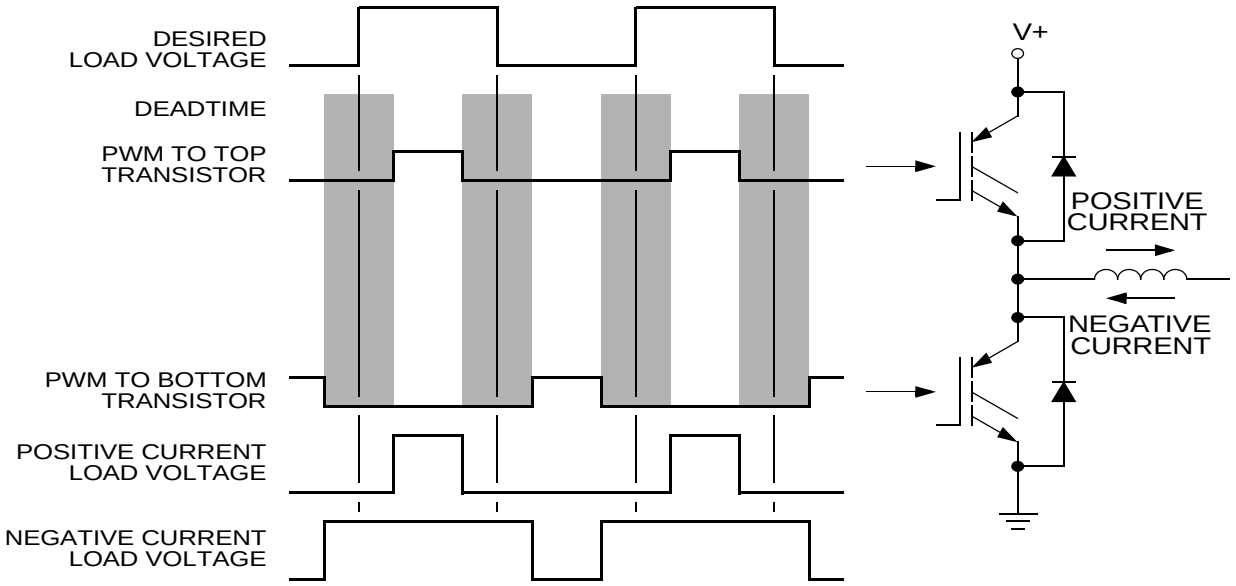


Figure 28-54. Deadtime distortion

During deadtime, load inductance distorts output voltage by keeping current flowing through the diodes. This deadtime current flow creates a load voltage that varies with current direction. With a positive current flow, the load voltage during deadtime is equal to the bottom supply, putting the top transistor in control. With a negative current flow, the load voltage during deadtime is equal to the top supply putting the bottom transistor in control.

Remembering that the original PWM pulse widths were shortened by deadtime insertion, the averaged sinusoidal output is less than the desired value. However, when deadtime is inserted, it creates a distortion in the motor current waveform. This distortion is aggravated by dissimilar turn-on and turn-off delays of each of the transistors. By giving the PWM module information on which transistor is controlling at a given time, this distortion can be corrected.

For a typical circuit in complementary channel operation, only one of the transistors is effective in controlling the output voltage at any given time. This depends on the direction of the motor current for that transistor pair. See Figure 28-54**.** To correct distortion, one of two different factors must be added to the desired PWM value, depending on whether the top or bottom transistor is controlling the output voltage. Therefore, the software is responsible for calculating both compensated PWM values prior to placing them in the VAL*x* registers. Either the VAL2/VAL3 or the VAL4/VAL5 register pair controls the pulse width at any given time. For a given PWM pair, whether the VAL2/VAL3 or VAL4/VAL5 pair is active depends on the state of the current status pin, PWM*x*, for that driver.

To correct deadtime distortion, software can decrease or increase the value in the appropriate VAL*x* register.

- In edge-aligned operation, decreasing or increasing the PWM value by a correction value equal to the deadtime typically compensates for deadtime distortion.
- In center-aligned operation, decreasing or increasing the PWM value by a correction value equal to one-half the deadtime typically compensates for deadtime distortion.

28.4.3.8.2 Manual correction

To detect the current status, the voltage on each PWMx pin is sampled twice in a PWM period, at the end of each deadtime. The value is stored in the DTx bits in the CTRL1 register. The DTx bits are a timing marker especially indicating when to toggle between PWM value registers.

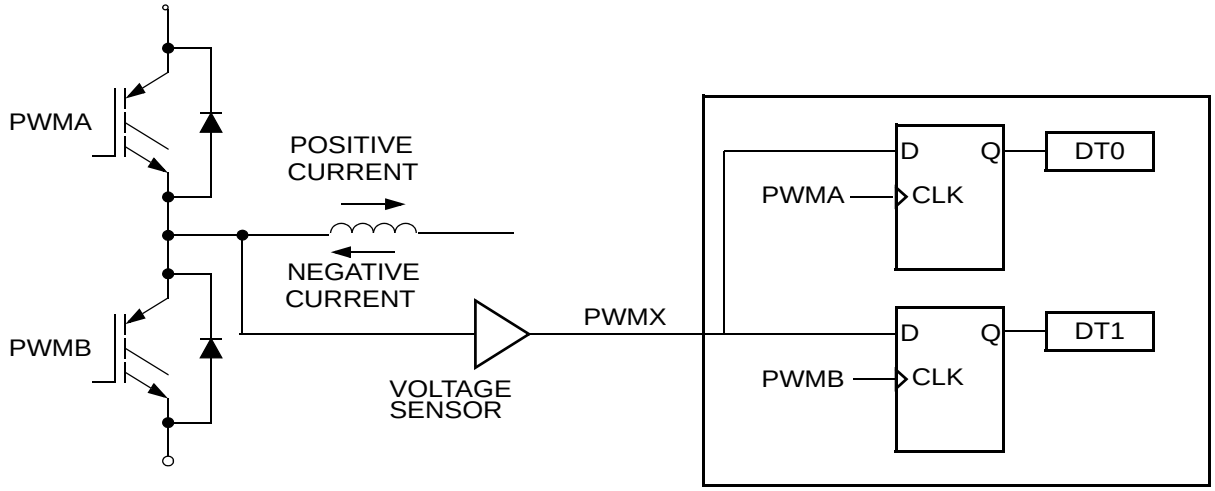


Figure 28-55. Current-status sense scheme for deadtime correction

Both D flip-flops latch low, DT0 = 0, DT1 = 0, during deadtime periods if current is large and flowing out of the complementary circuit. See Figure 28-55. Both D flip-flops latch the high, DT0 = 1, DT1 = 1, during deadtime periods if current is also large and flowing into the complementary circuit.

However, under low-current, the output voltage of the complementary circuit during deadtime is somewhere between the high and low levels. The current cannot free-wheel through the opposition anti-body diode, regardless of polarity, giving additional distortion when the current crosses zero. Sampled results will be DT0 = 0 and DT1 = 1. Thus, the best time to change one PWM value register to another is just before the current zero crossing.

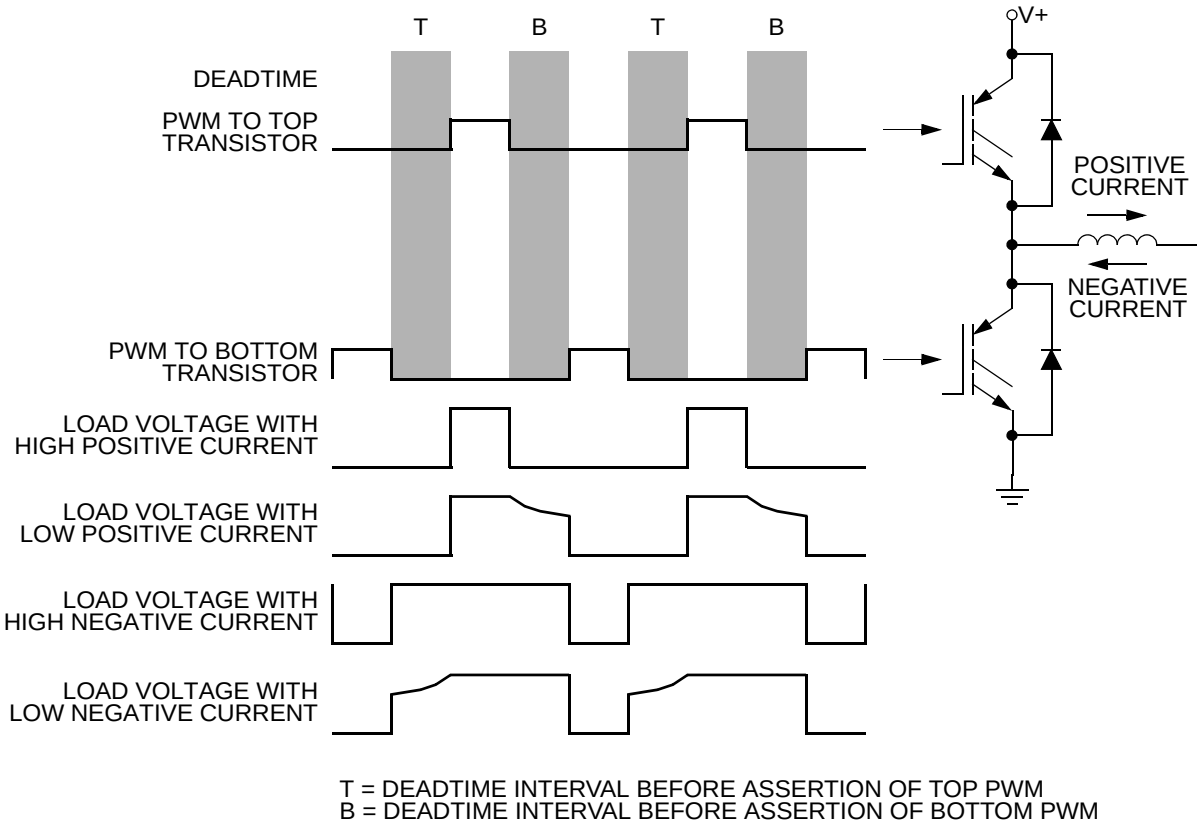


Figure 28-56. Output voltage waveforms

NOTE

In complementary mode, if necessary, the application should switch the source of the PWMi waveform from the VAL2 and VAL3 registers to the VAL4 and VAL5 registers.

28.4.3.9 Output logic

Figure 28-57 shows the output logic of each submodule including how each PWM output has individual fault disabling, polarity control, and output enable. This allows for maximum flexibility when interfacing to the external circuitry.

The PWM23 and PWM45 signals that are output from the deadtime logic in Figure 28-57 are positive true signals. In other words, a high level on these signals should result in the corresponding transistor in the PWM inverter being turned ON. The voltage level required at the PWM output pin to turn the transistor ON or OFF is a function of the logic between the pin and the transistor. Therefore, it is imperative that the user program the POLA and POLB bits before enabling the output pins. A fault condition can result in the PWM output being tristated, forced to a logic 1, or forced to a logic 0, depending on the values programmed into the PWMxFS fields.

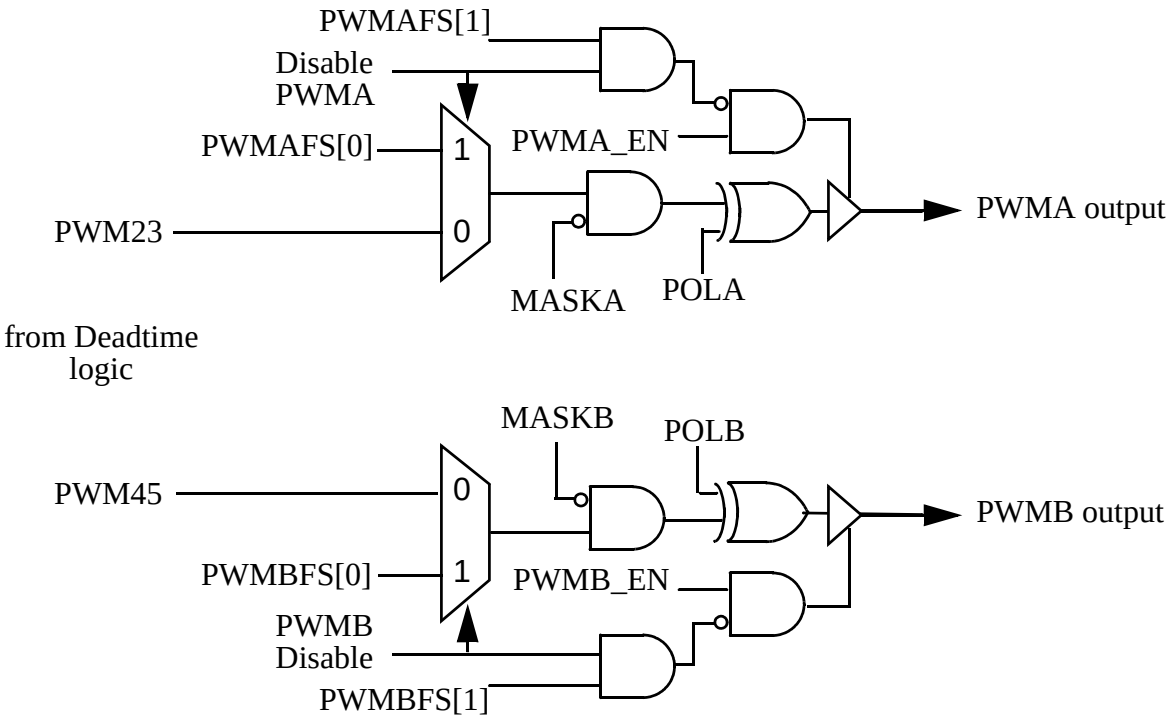


Figure 28-57. Output logic section

28.4.3.10 E-Capture

Commensurate with the idea of controlling both edges of an output signal, the Enhanced Capture (E-Capture) logic is designed to measure both edges of an input signal. As a result, when a submodule pin is configured for input capture, the CVAL*x* registers associated with that pin are used to record the edge values.

Figure 28-58 illustrates the block diagram of the E-Capture circuit. Upon entering the pin input, the signal is split into two paths. One goes straight to a mux input where software can select to pass the signal directly to the capture logic for processing. The other path connects the signal to an 8-bit counter that counts both the rising and falling edges of the signal. The output of this counter is compared to an 8-bit value that is specified by the user (EDGCMP*x*) and when the two values are equal, the comparator generates a pulse that resets the counter. This pulse is also supplied to the mux input where software can select it to be processed by the capture logic. This feature permits the E-Capture circuit to count up to 256 edge events before initiating a capture event. This feature is useful for dividing down high frequency signals for capture processing so that capture interrupts don't overwhelm the CPU. Also, this feature can be used to generate an interrupt after *n* events have been counted.

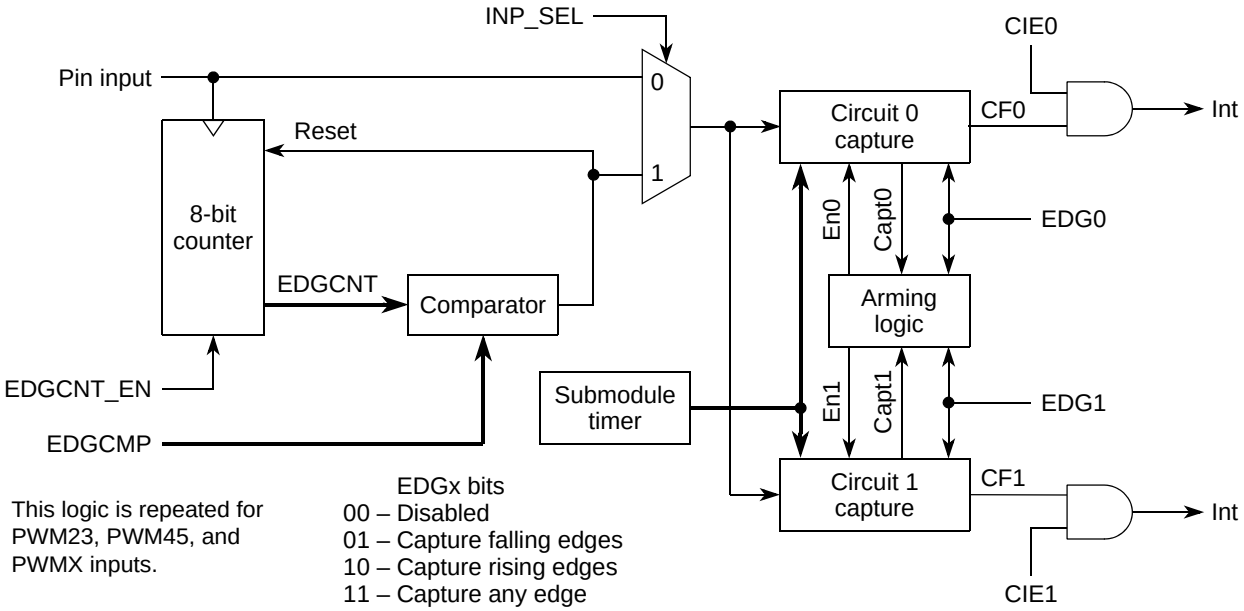


Figure 28-58. Enhanced capture (E-Capture) logic

Based on the mode selection, the mux selects either the pin input or the compare output from the count/compare circuit to be processed by the capture logic. The selected signal is routed to two separate capture circuits that work in tandem to capture sequential edges of the signal. The type of edge to be captured by each circuit is determined by the EDG*x*1 and EDG*x*0 bits whose functionality is listed in Figure 28-58. Also, controlling the operation of the capture circuits is the arming logic, which allows captures to be performed in a free-running (continuous) or one-shot fashion. In free-running mode, the capture sequences are performed indefinitely. If both capture circuits are enabled, they work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one-shot mode, only one capture sequence is performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

28.4.3.11 Fault protection

Fault protection can control any combination of PWM output pins. Faults are generated by a logic 1 on any of the FAULT*x* pins. This polarity can be changed via the FLVL bits. Each FAULT*x* pin can be mapped arbitrarily to any of the PWM outputs. When fault protection hardware disables PWM outputs, the PWM generator continues to run, only the output pins are forced to logic 0, logic 1, or tristated depending on the values of the PWM*x*FS bits.

The fault decoder disables PWM pins selected by the fault logic and the disable mapping register (DISMAP). See Figure 28-59 for an example of the fault disable logic. Each bank of bits in DISMAP controls the mapping for a single PWM pin. Refer to Table 28-24.

The fault protection is enabled even when the PWM module is not enabled; therefore, a fault is latched in and must be cleared in order to prevent an interrupt when the PWM is enabled.

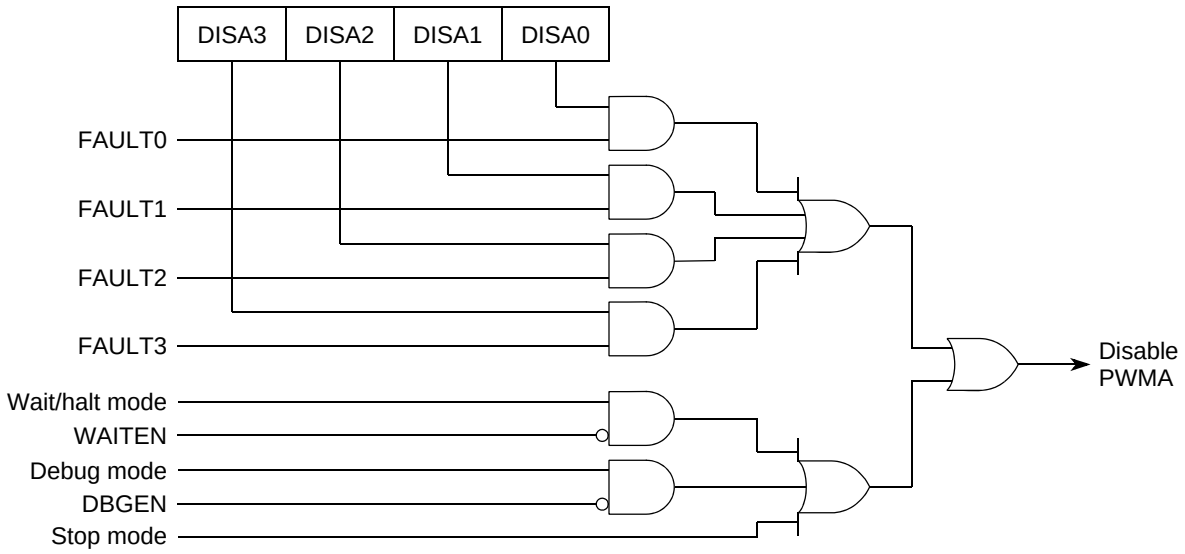


Figure 28-59. Fault decoder for PWMA

Table 28-24. Fault mapping

PWM Pin	Controlling Register Bits
PWMA	DISA[3:0]
PWMB	DISB[3:0]
PWMX	DISX[3:0]

28.4.3.11.1 Fault pin filter

Each fault pin has a programmable filter that can be bypassed. The sampling period of the filter can be adjusted with the FILT_PER field of the FFILT*x* register. The number of consecutive samples that must agree before an input transition is recognized can be adjusted using the FILT_CNT field of the same register. Setting FILT_PER to all 0 disables the input filter for a given FAULT*x* pin.

Upon detecting a logic 0 on the filtered FAULT*x* pin (or a logic 1 if FLVL*x* is set), the corresponding FFPIN*x* and fault flag, FFLAG*x*, bits are set. The FFPIN*x* bit remains set as long as the filtered FAULT*x* pin is zero. Clear FFLAG*x* by writing a logic 1 to FFLAG*x*.

If the FIE*x*, FAULT*x* pin interrupt enable bit is set, the FFLAG*x* flag generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears the FFLAG*x* flag by writing a logic 1 to the bit
- Software clears the FIE*x* bit by writing a logic 0 to it
- A reset occurs

Even with the filter enabled, there is a combinational path from the FAULT*x* inputs to the PWM pins. This logic is also capable of holding a fault condition in the event of loss of clock to the PWM module.

28.4.3.11.2 Automatic fault clearing

Setting an automatic clearing mode bit, FAUTO*x*, configures faults from the FAULT*x* pin for automatic clearing.

When FAUTO*x* is set, disabled PWM pins are enabled when the FAULT*x* pin returns to logic 1 and a new PWM full or half cycle begins. See Figure 28-60. If FFULL*x* is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle. Clearing the FFLAG*x* flag does not affect disabled PWM pins when FAUTO*x* is set.

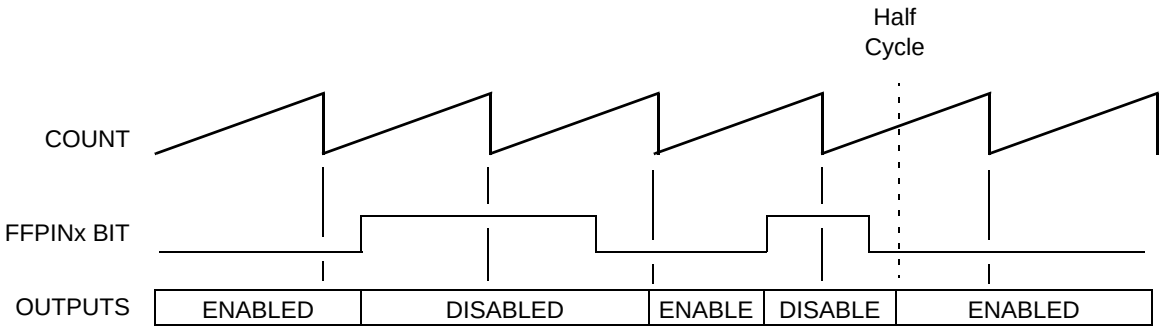


Figure 28-60. Automatic fault clearing

28.4.3.11.3 Manual fault clearing

Clearing the automatic clearing mode bit, FAUTO*x*, configures faults from the FAULT*x* pin for manual clearing:

- If the fault safety mode bits, FSAFE*x*, are clear, then PWM pins disabled by the FAULT*x* pins are enabled when:
  - Software clears the corresponding FFLAG*x* flag.
  - The next PWM full or half cycle begins regardless of the logic level detected by the filter at the FAULT*x* pin. See Figure 28-61. If FFULL*x* is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle.
- If the fault safety mode bits, FSAFE*x*, are set, then PWM pins disabled by the FAULT*x* pins are enabled when:
  - Software clears the corresponding FFLAG*x* flag.
  - The filter detects a logic 1 on the FAULT*x* pin at the start of the next PWM full or half cycle boundary. See Figure 28-62. If FFULL*x* is set, then the disabled PWM pins are enabled only at the start of a full cycle and not at the half cycle.

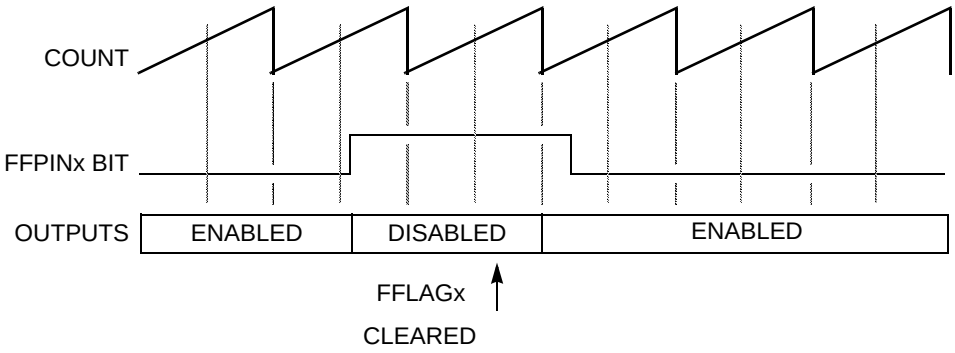


Figure 28-61. Manual fault clearing (FSAFE = 0)

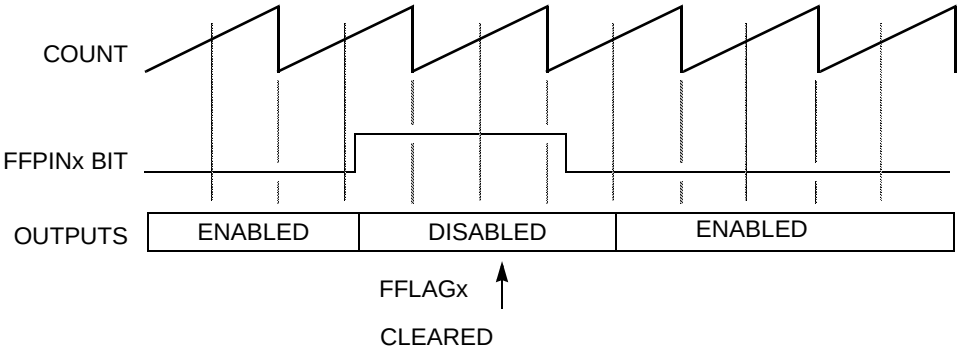


Figure 28-62. Manual fault clearing (FSAFE = 1)

NOTE

Fault protection also applies during software output control when the SEL23 and SEL45 fields are set to select OUT23 and OUT45 bits. Fault clearing still occurs at half-PWM cycle boundaries while the PWM generator is engaged; RUN equals one. But the OUTx bits can control the PWM pins while the PWM generator is off; RUN equals zero. Thus, fault clearing occurs at IPBus cycles while the PWM generator is off and at the start of PWM cycles when the generator is engaged.

28.4.3.11.4 Fault testing

The FTEST bit simulates a fault condition on each of the fault inputs.

28.4.4 PWM generator loading

28.4.4.1 Load enable

The LDOK bit enables loading of the following PWM generator parameters:

- The prescaler divisor—from the PRSC bits in the CTRL1 register
- The PWM period and pulse width—from the INIT and VAL*x* registers

LDOK allows software to finish calculating all of these PWM parameters so they can be synchronously updated. The PSRC, INIT, and VAL*x* registers are loaded by software into a set of outer buffers. When LDOK is set, these values are transferred to an inner set of registers at the beginning of the next PWM reload cycle to be used by the PWM generator. These values can be transferred to the inner set of registers immediately upon setting LDOK if LDMOD is set. Set LDOK by reading it when it is a logic 0 and then writing a logic 1 to it. After loading, LDOK is automatically cleared.

28.4.4.2 Load frequency

The LDFQ bits in the CTRL1 register select an integral loading frequency of 1–16 PWM reload opportunities. The LDFQ bits take effect at every PWM reload opportunity, regardless of the state of the LDOK bit. The HALF and FULL bits in the CTRL1 register control reload timing. If FULL is set, a reload opportunity occurs at the end of every PWM cycle when the count equals VAL1. If HALF is set, a reload opportunity occurs at the half cycle when the count equals VAL0. If both HALF and FULL are set, a reload opportunity occurs twice per PWM cycle when the count equals VAL1 and when it equals VAL0.

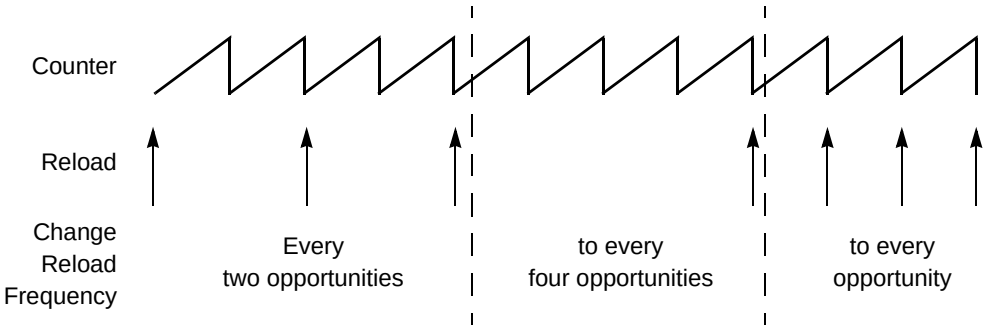


Figure 28-63. Full cycle reload frequency change

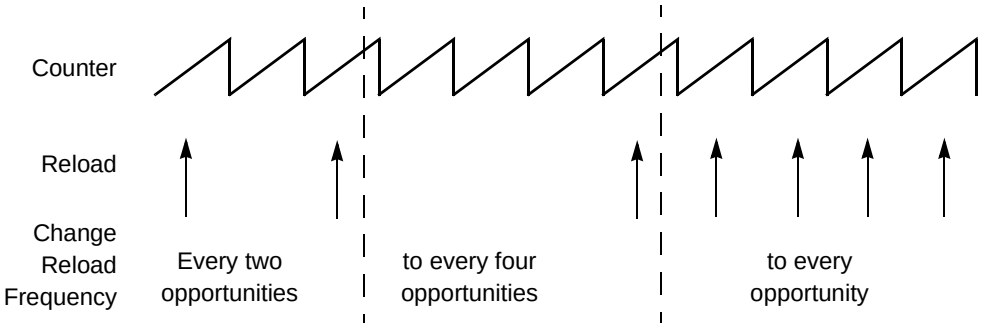


Figure 28-64. Half cycle reload frequency change

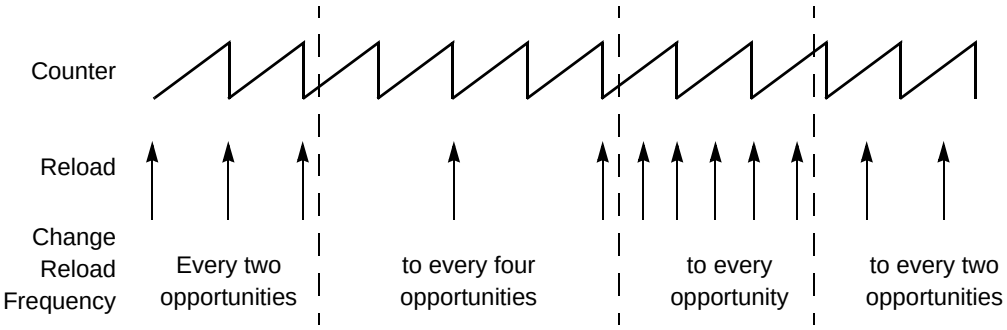


Figure 28-65. Full and half cycle reload frequency change

28.4.4.3 Reload flag

At every reload opportunity the PWM Reload Flag (RF) in the CTRL1 register is set. Setting RF happens even if an actual reload is prevented by the LDOK bit. If the PWM reload interrupt enable bit RIE is set, the RF flag generates CPU interrupt requests, allowing software to calculate new PWM parameters in real time. When RIE is not set, reloads still occur at the selected reload rate without generating CPU interrupt requests.

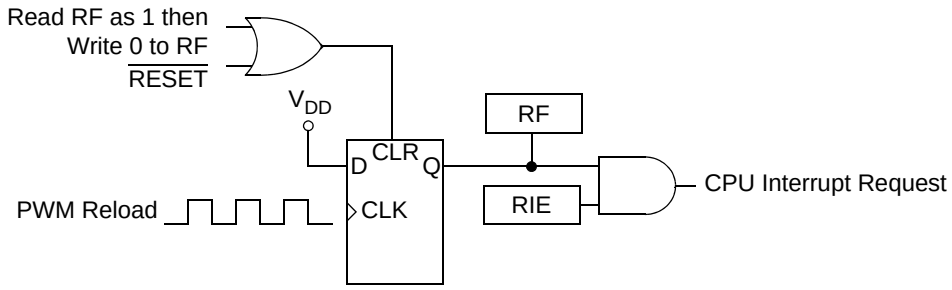


Figure 28-66. PWMF reload interrupt request

28.4.4.4 Reload errors

Whenever one of the VAL*x* or PSRC registers is updated, the RUF flag is set to indicate that the data is not coherent. RUF is cleared by a successful reload, which consists of the reload signal while LDOK is set. If RUF is set and LDOK is clear when the reload signal occurs, a reload error has taken place and the REF bit is set. If RUF is clear when a reload signal asserts, then the data is coherent and no error is flagged.

28.4.4.5 Initialization

Initialize all registers and set the LDOK bit before setting the RUN bit.

NOTE

Even if LDOK is not set, setting RUN also sets the RF flag. To prevent a CPU interrupt request, clear the RIE bit before setting RUN.

The PWM generator uses the last values loaded if RUN is cleared and then set while LDOK equals zero.

When the RUN bit is cleared:

- The RF flag and pending CPU interrupt requests are not cleared
- All fault circuitry remains active
- Software/external output control remains active
- Deadtime insertion continues during software/external output control

28.5 Resets

The FlexPWM module can only be reset by the IPG_HARD_ASYNC_RESET_B signal. This forces all registers to their reset states and tri-states the PWM outputs.

28.6 Clocks

Each PWM submodule receives its own copy of the IPBus clock. This allows for better power management by turning off the clock to unused submodules. The fault logic also has its own version of the IPBus clock.

Table 28-25. Clock summary

Clock	Used by
ipg_clk_sub0	Submodule 0
ipg_clk_sub1	Submodule 1
ipg_clk_sub2	Submodule 2
ipg_clk_sub3	Submodule 3
ipg_clk_flt	Fault logic

28.7 Interrupts

Each of the submodules within the FlexPWM can generate an interrupt from several sources. The fault logic can also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 28-26. Interrupt summary

Core interrupt	Interrupt flag	Interrupt enable	Name	Description
COF0	CMPF_0	CMPIE_0	Submodule 0 compare interrupt	Compare event has occurred
CAF0	CFX1_0, CFX0_0	CFX1IE_0, CFX0IE_0	Submodule 0 input capture interrupt	Input capture event has occurred
RF0	RF_0	RIE_0	Submodule 0 reload interrupt	Reload event has occurred
COF1	CMPF_1	CMPIE_1	Submodule 1 compare interrupt	Compare event has occurred
CAF1	CFX1_1, CFX0_1	CFX1IE_1, CFX0IE_1	Submodule 1 input capture interrupt	Input capture event has occurred
RF1	RF_1	RIE_1	Submodule 1 reload interrupt	Reload event has occurred
COF2	CMPF_2	CMPIE_2	Submodule 2 compare interrupt	Compare event has occurred
CAF2	CFX1_2, CFX0_2	CFX1IE_2, CFX0IE_2	Submodule 2 input capture interrupt	Input capture event has occurred
RF2	RF_2	RIE_2	Submodule 2 reload interrupt	Reload event has occurred
COF3	CMPF_3	CMPIE_3	Submodule 3 compare interrupt	Compare event has occurred

Table 28-26. Interrupt summary (continued)

Core interrupt	Interrupt flag	Interrupt enable	Name	Description
CAF3	CFX1_3, CFX0_3	CFX1IE_3, CFX0IE_3	Submodule 3 input capture interrupt	Input capture event has occurred
RF3	RF_3	RIE_3	Submodule 3 reload interrupt	Reload event has occurred
REF	REF_0	REIE_0	Submodule 0 reload error interrupt	Reload error has occurred
	REF_1	REIE_1	Submodule 1 reload error interrupt	
	REF_2	REIE_2	Submodule 2 reload error interrupt	
	REF_3	REIE_3	Submodule 3 reload error interrupt	
FFLAG	FFLAG	FIE	Fault input interrupt	Fault condition has been detected

28.8 DMA

Each submodule can request a DMA read access for its capture FIFOs and a DMA write request for its double-buffered VAL*x* registers.

Table 28-27. DMA summary

DMA Request	DMA enable	Name	Description
Submodule 0 read request	CX0DE_0	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_0	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_0	Capture FIFO read request source select	Selects source of read DMA request
Submodule 0 write request	VALDE_0	VAL*x* write request	VAL*x* registers need to be updated
Submodule 1 read request	CX0DE_1	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_1	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_1	Capture FIFO read request source select	Selects source of read DMA request
Submodule 1 write request	VALDE_1	VAL*x* write request	VAL*x* registers need to be updated
Submodule 2 read request	CX0DE_2	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_2	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_2	Capture FIFO read request source select	Selects source of read DMA request
Submodule 2 write request	VALDE_2	VAL*x* write request	VAL*x* registers need to be updated

Table 28-27. DMA summary (continued)

DMA Request	DMA enable	Name	Description
Submodule 3 read request	CX0DE_3	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_3	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_3	Capture FIFO read request source select	Selects source of read DMA request
Submodule 3 write request	VALDE_3	VAL*x* write request	VAL*x* registers need to be updated

NOTE

When some submodules use DMA to load their VALx registers and other submodules use non-DMA means that means direct writes from the CPU, the LDOK bits for the non-DMA submodules can be incorrectly cleared at the completion of the DMA controlled load cycle. This leads to the non-DMA channels not being properly updated.

Submodules that use DMA to read the input capture registers do not cause a problem for non-DMA submodules.

To manage this behavior, set the DMA enable bit to 1 also for non-DMA submodules, according to this the DMA will not incorrectly clear the LDOK bit for non-DMA submodules but they will be set to 1 at the end of each DMA cycle. When the CPU has to update the VALx registers of non-DMA submodules, first clear LDOK bit for non-DMA submodules.

Chapter 29
FlexRay Communication Controller (FLEXRAY)

29.1 Introduction

29.1.1 Reference

The following documents are referenced.

- *FlexRay Communications System Protocol Specification, Version* 2.1 Rev A[1]
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A*

29.1.2 Glossary

This section provides a list of terms used in this chapter.

Table 29-1. List of terms

Term	Definition
BCU	Buffer Control Unit - handles message buffer access
BMIF	Bus Master Interface—provides master access to FlexRay memory area
CC	Communication Controller
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in μT	The actual length of a cycle in μT for the ideal controller (±0 ppm)
EBI	External Bus Interface
FlexRay Memory Area	Memory area to store the physical message buffer payload data, frame header, frame and slot status, and synchronization frame related tables
System Memory	Memory that contains the FlexRay Memory Area
System Bus	Bus that connects the controller and System Memory
FSS	Frame Start Sequence
HIF	Host Interface—provides host access to controller
Host	FlexRay CC host MCU
LUT	Look Up Table—stores message buffer header index value
LRAM	Look Up Table RAM—module internal memory to store message buffer configuration data and data field offsets for individual message buffers and receive shadow buffers
MB	Message Buffer
MBIDX	Message Buffer Index—the position of a header field entry within the header area; if the header area is accessed as an array, this is the same as the array index of the entry
MBNum	Message Buffer Number—position of the message buffer configuration registers within the register map; for example, Message Buffer Number 5 corresponds to the MBCCS5 register
MCU	Microcontroller Unit
μT	Microtick

1. The FlexRay Specifications have been developed for automotive applications.The FlexRay Specifications have been neither developed nor tested for non-automotive applications.

Table 29-1. List of terms (continued)

Term	Definition
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time
PE	Protocol Engine
POC	Protocol Operation Control—each state of the POC is denoted by *POC:state*
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission
sync frame	Null frame or message frame with *Sync Frame Indicator* set to 1
startup frame	Null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 1
normal frame	Null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 0
null frame	Frame with *Null Frame Indicator* set to 0
message frame	Frame with *Null Frame Indicator* set to 1

29.1.3 Color coding

Throughout this chapter, types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants, and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

29.1.4 Overview

The CC is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The CC has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the CC with its surrounding modules is given in Figure 29-1.

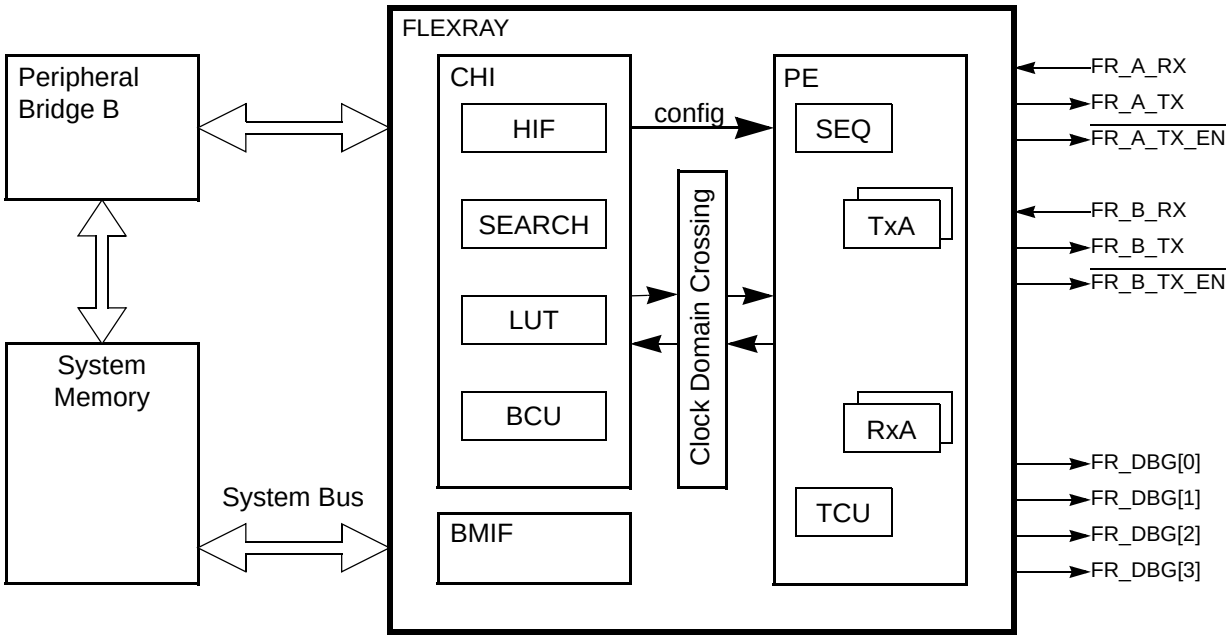


Figure 29-1. FLEXRAY block diagram

The protocol engine has two transmitter units (TxA and TxB) and two receiver units (RxA and RxB) for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information are stored in the FlexRay memory area.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain, and vice versa, to allow for asynchronous PE and CHI clock domains.

The CC stores the frame header and payload data of frames received or of frames to be transmitted in the FlexRay memory area. The application accesses the FlexRay memory area to retrieve and provide the frames to be processed by the CC. In addition to the frame header and payload data, the CC stores the synchronization frame related tables in the FlexRay memory area for application processing.

The FlexRay memory area is located in the system memory of the MCU. The CC has access to the FlexRay memory area via its bus master interface (BMIF). The host provides the start address of the FlexRay memory area within the system memory by programming the System Memory Base Address Register (FR_SYMBADR). All FlexRay memory area related offsets are stored in offset registers. The physical address pointer into the FlexRay memory area is calculated using the offset values and the FlexRay memory base address.

NOTE

The CC does not provide a memory protection scheme for the FlexRay memory area.

29.1.5 Features

The CC provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*-compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A*-compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 64 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status, and payload data stored in dedicated FlexRay memory area
  - Allows for flexible and efficient message buffer implementation
  - Ensures consistent data access by means of buffer locking scheme
  - Allows application to lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0–254 bytes
- Two independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
  - Receive message buffer
  - Transmit message buffer
- Individual message buffer reconfiguration supported
  - Means provided to safely disable individual message buffers
  - Disabled message buffers can be reconfigured
- 2 independent receive FIFOs
  - One receive FIFO per channel
  - Up to 255 entries for each FIFO

  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for the dynamic segment
- Four configurable slot error counters
- Four dedicated slot status indicators
  - Used to observe slots without using receive message buffers
- Measured value indicators for the clock synchronization
  - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory area
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- One absolute timer
- One timer that can be configured to absolute or relative
- SECDED for protocol engine data ram
- SEDDED for CHI lookup table ram

29.1.6 Modes of operation

This section describes the basic operational power modes of the CC.

29.1.6.1 Disabled mode

The CC enters the Disabled mode during hard reset. The CC indicates that it is in the Disabled mode by negating the module enable bit MEN in the Module Configuration Register (FR_MCR).

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled mode* can be accessed for writing as stated in Section 29.5.2, Register descriptions.

The application configures the CC by accessing the configuration bits and fields in the Module Configuration Register (FR_MCR).

29.1.6.1.1 Leave Disabled mode

The CC leaves Disabled mode and enters Normal mode when the application writes a 1 to the module enable bit MEN in the Module Configuration Register (FR_MCR).

NOTE

When the CC is enabled, it cannot be disabled later on.

29.1.6.2 Normal mode

In this mode, the CC is fully functional. The CC indicates that it is in Normal mode by asserting the module enable bit MEN in the Module Configuration Register (FR_MCR).

29.1.6.2.1 Enter Normal mode

This mode is entered when the application requests the CC to leave the Disabled mode. If the Normal mode was entered by leaving the Disabled mode, the application has to perform the protocol initialization described in 29.7.2.2, Protocol Initialization, to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the Module Configuration Register (FR_MCR), the corresponding FlexRay bus driver ports are enabled and driven.

29.1.6.3 Debug mode

When the MCU is in debug mode, the FLEXRAY behavior is unaffected and remains dedicated by the mode of the FLEXRAY.

29.2 External signal description

This section lists and describes the CC signals, connected to external pins. These signals are summarized in Table 29-2 and described in detail in Section 29.2.1, Detailed signal descriptions.

NOTE

The off-chip signals FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ are available on each package option. The availability of the other off-chip signals depends on the package option.

Table 29-2. External signal properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
$\overline{\text{FR\_A\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
$\overline{\text{FR\_B\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

29.2.1 Detailed signal descriptions

This section provides a detailed description of the CC signals, connected to external pins.

29.2.1.1 FR_A_RX—Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

29.2.1.2 FR_A_TX—Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

29.2.1.3 $\overline{\text{FR\_A\_TX\_EN}}$—Transmit Enable Channel A

The $\overline{\text{FR\_A\_TX\_EN}}$ signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel A.

29.2.1.4 FR_B_RX—Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

29.2.1.5 FR_B_TX—Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver.

29.2.1.6 $\overline{\text{FR\_B\_TX\_EN}}$—Transmit Enable Channel B

The $\overline{\text{FR\_B\_TX\_EN}}$ signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel B.

29.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0]—Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to Section 29.6.16, Strobe Signal Support.

29.3 Controller host interface clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. Two constraints apply to the minimum CHI clock frequency.

The first constraint corresponds to the number of utilized message buffers and is specified in Section 29.7.5, Number of Usable Message Buffers.

The second constraint corresponds to the value of the TIMEOUT field in the System Memory Access Time-Out Register (FR_SYMATOR) and is specified in Section 29.7.1.1, Configure System Memory Access Time-Out Register (FR_SYMATOR).

29.4 Protocol engine clocking

The clock for the protocol engine can be generated by two sources. The first source is the external crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the Module Configuration Register (FR_MCR).

29.4.1 Oscillator clocking

If the protocol engine is clocked by the external crystal oscillator, a 40 MHz crystal or CMOS-compatible clock must be connected to the oscillator pins. The crystal or clock must fulfill the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

29.5 Memory map and register description

The CC occupies 8 KB (8192 bytes) of address space starting at the CC's base address defined by the memory map of the MCU.

29.5.1 Memory map

The complete memory map of the CC is shown in Table 29-3. The addresses presented here are the offsets relative to the CC base address that is defined by the MCU address map.

Table 29-3. FlexRay memory map

Offset	Register	Access
Module Configuration and Control		
0x0000	Module Version Register (FR_MVR)	R
0x0002	Module Configuration Register (FR_MCR)	R/W
0x0004	System Memory Base Address High Register (FR_SYMBADHR)	R/W
0x0006	System Memory Base Address Low Register (FR_SYMBADLR)	R/W
0x0008	Strobe Signal Control Register (FR_STBSCR)	R/W
0x000A	Reserved	R
0x000C	Message Buffer Data Size Register (FR_MBDSR)	R/W
0x000E	Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)	R/W
PE Access Registers		
0x0010	PE DRAM Access Register (FR_PEDRAR)	R/W
0x0012	PE DRAM Data Register (FR_PEDRDR)	R/W
Interrupt and Error Handling		
0x0014	Protocol Operation Control Register (FR_POCR)	R/W
0x0016	Global Interrupt Flag and Enable Register (FR_GIFER)	R/W
0x0018	Protocol Interrupt Flag Register 0 (FR_PIFR0)	R/W
0x001A	Protocol Interrupt Flag Register 1 (FR_PIFR1)	R/W
0x001C	Protocol Interrupt Enable Register 0 (FR_PIER0)	R/W
0x001E	Protocol Interrupt Enable Register 1 (FR_PIER1)	R/W
0x0020	CHI Error Flag Register (FR_CHIERFR)	R/W

Table 29-3. FlexRay memory map (continued)

Offset	Register	Access
0x0022	Message Buffer Interrupt Vector Register (FR_MBIVEC)	R
0x0024	Channel A Status Error Counter Register (FR_CASERCR)	R
0x0026	Channel B Status Error Counter Register (FR_CBSERCR)	R
Protocol Status		
0x0028	Protocol Status Register 0 (FR_PSR0)	R
0x002A	Protocol Status Register 1 (FR_PSR1)	R
0x002C	Protocol Status Register 2 (FR_PSR2)	R
0x002E	Protocol Status Register 3 (FR_PSR3)	R/W
0x0030	Macrotick Counter Register (FR_MTCTR)	R
0x0032	Cycle Counter Register (FR_CYCTR)	R
0x0034	Slot Counter Channel A Register (FR_SLTCTAR)	R
0x0036	Slot Counter Channel B Register (FR_SLTCTBR)	R
0x0038	Rate Correction Value Register (FR_RTCORVR)	R
0x003A	Offset Correction Value Register (FR_OFCORVR)	R
0x003C	Combined Interrupt Flag Register (FR_CIFR)	R
0x003E	System Memory Access Time-Out Register (FR_SYMATOR)	R/W
Sync Frame Counter and Tables		
0x0040	Sync Frame Counter Register (FR_SFCNTR)	R
0x0042	Sync Frame Table Offset Register (FR_SFTOR)	R/W
0x0044	Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)	R/W
Sync Frame Filter		
0x0046	Sync Frame ID Rejection Filter Register (FR_SFIDRFR)	R/W
0x0048	Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)	R/W
0x004A	Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)	R/W
Network Management Vector		
0x004C	Network Management Vector Register 0 (FR_NMVR0)	R
0x004E	Network Management Vector Register 1 (FR_NMVR1)	R
0x0050	Network Management Vector Register 2 (FR_NMVR2)	R
0x0052	Network Management Vector Register 3 (FR_NMVR3)	R
0x0054	Network Management Vector Register 4 (FR_NMVR4)	R
0x0056	Network Management Vector Register 5 (FR_NMVR5)	R
0x0058	Network Management Vector Length Register (FR_NMVLR)	R/W

Table 29-3. FlexRay memory map (continued)

Offset	Register	Access
Timer Configuration		
0x005A	Timer Configuration and Control Register (FR_TICCR)	R/W
0x005C	Timer 1 Cycle Set Register (FR_TI1CYSR)	R/W
0x005E	Timer 1 Macrotick Offset Register (FR_TI1MTOR)	R/W
0x0060	Timer 2 Configuration Register 0 (FR_TI2CR0)	R/W
0x0062	Timer 2 Configuration Register 1 (FR_TI2CR1)	R/W
Slot Status Configuration		
0x0064	Slot Status Selection Register (FR_SSSR)	R/W
0x0066	Slot Status Counter Condition Register (FR_SSCCR)	R/W
Slot Status		
0x0068	Slot Status Register 0 (FR_SSR0)	R
0x006A	Slot Status Register 1 (FR_SSR1)	R
0x006C	Slot Status Register 2 (FR_SSR2)	R
0x006E	Slot Status Register 3 (FR_SSR3)	R
0x0070	Slot Status Register 4 (FR_SSR4)	R
0x0072	Slot Status Register 5 (FR_SSR5)	R
0x0074	Slot Status Register 6 (FR_SSR6)	R
0x0076	Slot Status Register 7 (FR_SSR7)	R
0x0078	Slot Status Counter Register 0 (FR_SSCR0)	R
0x007A	Slot Status Counter Register 1 (FR_SSCR1)	R
0x007C	Slot Status Counter Register 2 (FR_SSCR2)	R
0x007E	Slot Status Counter Register 3 (FR_SSCR3)	R
MTS Generation		
0x0080	MTS A Configuration Register (FR_MTSACFR)	R/W
0x0082	MTS B Configuration Register (MTSBCFR)	R/W
Shadow Buffer Configuration		
0x0084	Receive Shadow Buffer Index Register (FR_RSBIR)	R/W
Receive FIFO — Configuration		
0x0086	Receive FIFO Watermark and Selection Register (FR_RFWMSR)	R/W
0x0088	Receive FIFO Start Index Register (FR_RFSIR)	R/W
0x008A	Receive FIFO Depth and Size Register (RFDSR)	R/W

Table 29-3. FlexRay memory map (continued)

Offset	Register	Access
Receive FIFO — Control		
0x008C	Receive FIFO A Read Index Register (FR_RFARIR)	R
0x008E	Receive FIFO B Read Index Register (FR_RFBRIR)	R
Receive FIFO — Filter		
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)	R/W
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)	R/W
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)	R/W
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)	R/W
0x0098	Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)	R/W
0x009A	Receive FIFO Range Filter Control Register (FR_RFRFCTR)	R/W
Dynamic Segment Status		
0x009C	Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)	R
0x009E	Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)	R
Protocol Configuration		
0x00A0 ... 0x00DC	Protocol Configuration Register 0 (FR_PCR0) ... Protocol Configuration Register 30 (FR_PCR30)	R/W
0x00DE ... 0x00E4	Reserved[1]	R
Receive FIFO — Configuration (cont.)		
0x00E6	Receive FIFO Start Data Offset Register (FR_RFSDOR)	R/W
0x00E8	Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)	R/W
0x00EA	Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)	R/W
0x00EC	Receive FIFO Periodic Timer Register (FR_RFPTR)	R/W
Receive FIFO — Control (cont.)		
0x00EE	Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)	R/W

Table 29-3. FlexRay memory map (continued)

Offset	Register	Access
ECC Registers		
0x00F0	ECC Error Interrupt Flag and Enable Register (FR_EEIFER)	R/W
0x00F2	ECC Error Report and Injection Control Register (FR_EERICR)	R/W
0x00F4	ECC Error Report Address Register (FR_EERAR)	R
0x00F6	ECC Error Report Data Register (FR_EERDR)	R
0x00F8	ECC Error Report Code Register (FR_EERCR)	R
0x00FA	ECC Error Injection Address Register (FR_EEIAR)	R/W
0x00FC	ECC Error Injection Data Register (FR_EEIDR)	R/W
0x00FE	ECC Error Injection Code Register (FR_EEICR)	R/W
0x0100 ... 0x07FE	Not Implemented[1]	R
Message Buffers Configuration, Control, Status		
0x0800	Message Buffer Configuration, Control, Status Register 0 (FR_MBCCSR0)	R/W
0x0802	Message Buffer Cycle Counter Filter Register 0 (FR_MBCCFR0)	R/W
0x0804	Message Buffer Frame ID Register 0 (FR_MBFIDR0)	R/W
0x0806	Message Buffer Index Register 0 (FR_MBIDXR0)	R/W
...	...	...
0x09F8	Message Buffer Configuration, Control, Status Register 63 (FR_MBCCSR63)	R/W
0x09FA	Message Buffer Cycle Counter Filter Register 63 (FR_MBCCFR63)	R/W
0x09FC	Message Buffer Frame ID Register 63 (FR_MBFIDR63)	R/W
0x09FE	Message Buffer Index Register 63 (FR_MBIDXR63)	R/W
0x0A00 ... 0x0FFF	Not Implemented[1]	R
0x1000 ... 0x1086	Message Buffer Data Field Offset Register 0 (FR_MBDOR0) ... Message Buffer Data Field Offset Register 67 (FR_MBDOR67)	R/W
0x1088 ... 0x108E	Reserved[1]	R/W
0x1090 ... 0x109A	LRAM ECC Error Test Register 0 (FR_LEETR0) ... LRAM ECC Error Test Register 5 (FR_LEETR5)	R/W
0x109C ... 0x1FFE	Not Implemented[1]	R

[1] See Table 29-4.

29.5.2 Register descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities.

Table 29-4 provides a key for the register figures and register tables.

Table 29-4. Register access conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read and is cleared by writing a one. Writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
—	Not defined after reset and not affected by reset.

29.5.2.1 Register reset

All registers except the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn), Message Buffer Frame ID Registers (FR_MBFIDRn), and Message Buffer Index Registers (FR_MBIDXRn) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and thus are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in Table 29-5.

Table 29-5. Additional register reset conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes RUN command 0101 to the POCCMD field in the Protocol Operation Control Register (FR_POCR).
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit FR_MBCCSR*n*[EDT] while the message buffer is enabled (FR_MBCCSR*n*[EDS] = 1) and the CC grants the disable to the application by clearing the FR_MBCCSR*n*[EDS] bit.

29.5.2.2 Register write access

This section describes the write access restriction terms that apply to all registers.

29.5.2.2.1 Register write access restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in Table 29-6. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.The condition term [A and B] indicates that the register or field can be written to if both conditions are fulfilled.

Table 29-6. Register write access restrictions

Condition	Indication	Description
Any Time	—	No write access restriction.
Disabled mode	FR_MCR[MEN] = 0	Write access only when CC is in Disabled mode.
Normal mode	FR_MCR[MEN] = 1	Write access only when CC is in Normal mode.
POC:config	FR_PSR0[PROTSTATE] = POC:config	Write access only when Protocol is in the POC:config state.
MB_DIS	FR_MBCCSR[EDS] = 0	Write access only when related Message Buffer is disabled.
MB_LCK	FR_MBCCSR*n*[LCKS] = 1	Write access only when related Message Buffer is locked.
IDL	FR_EEIRICR[BSY] = 0	Write access only when ECC configuration is idle.

29.5.2.2.2 Register write access requirements

All registers can be accessed with 8-bit, 16-bit, and 32-bit wide operations.

For some of the registers, a write access that is at least 16 bits wide is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected. If an 8-bit wide write access is performed to any of these registers, this access is ignored without notification.

29.5.2.2.3 Internal register access

The following memory-mapped registers are used to access multiple internal registers.

- Strobe Signal Control Register (FR_STBSCR)
- Slot Status Selection Register (FR_SSSR)
- Slot Status Counter Condition Register (FR_SSCCR)
- Receive Shadow Buffer Index Register (FR_RSBIR)

Each of these memory-mapped registers provides an SEL field and a WMD bit. The SEL field selects the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

29.5.2.3 Module Version Register (FR_MVR)

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHIVER								PEVER							
W																
Reset	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0

Figure 29-2. Module Version Register (FR_MVR)

This register provides the CC version number. The module version number is derived from the CHI version number and the PE version number.

Table 29-7. FR_MVR field descriptions

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the controller host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

29.5.2.4 Module Configuration Register (FR_MCR)

Base + 0x0002 Write: MEN, SBFF, SCM, CHB, CHA, ECCE, FUM, FAM, CLKSEL, BITRATE: Disabled mode
SFFE: Disabled mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	ECCE	R*	FUM	FAM	0	CLK SEL	BITRATE			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-3. Module Configuration Register (FR_MCR)

This register defines the global configuration of the CC.

Table 29-8. FR_MCR field descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the CC is in the Disabled mode. The application requests the CC to leave the Disabled mode by writing 1 to this bit. Before leaving the Disabled mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, and BITRATE values. For details, see Section 29.1.6, Modes of operation. 0 Write: ignored, CC disable not possible. Read: CC disabled. 1 Write: enable CC. Read: CC enabled. **Note:** If the CC is enabled it cannot be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the CC in case of a system bus failure. 0 Continue normal operation. 1 Transition to freeze mode.
SCM	**Single Channel Device mode** — This control bit defines the channel device mode of the CC as described in Section 29.6.10, Channel Device Modes. 0 CC works in dual channel device mode. 1 CC works in single channel device mode.

Table 29-8. FR_MCR field descriptions (continued)

Field	Description
CHB CHA	**Channel Enable** — Protocol-related parameter: *pChannels* The semantics of these control bits depends on the channel device mode controlled by the SCM bit and is given in Table 29-9.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see Section 29.6.15, Sync Frame Filtering. 0 Synchronization frame filtering disabled. 1 Synchronization frame filtering enabled.
ECCE	**ECC Functionality Enable** — This bit controls the ECC memory error detection functionality. For details see Section 29.6.24, Memory Content Error Detection. 0 ECC functionality (injection, detection, reporting, response) disabled. 1 ECC functionality enabled.
FUM	**FIFO Update mode** — This bit controls the FIFO update behavior when the interrupt flags FR_GIFER[FAFAIF] and FR_GIFER[FAFBIF] are written by the application (see Section 29.6.9.8, FIFO Update). 0 FIFOA/FIFOB is updated on writing 1 to FR_GIFER[FAFAIF] /FR_GIFER[FAFBIF]. 1 FIFOA/FIFOB) is *not* updated on writing 1 to FR_GIFER[FAFAIF]/FR_GIFER[FAFBIF].
FAM	**FIFO Address mode** — This bit controls the location of the system memory base address for the FIFOs (see Section 29.6.9.2, FIFO Configuration). 0 FIFO Base Address located in System Memory Base Address Register (FR_SYMBADR). 1 FIFO Base Address located in Receive FIFO System Memory Base Address Register (FR_RFSYMBADR).
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by external crystal oscillator. 1 PE clock source is generated by on-chip PLL.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the FlexRay Bus Bit Rate. 000 10.0 Mbit/sec 001 5.0 Mbit/sec 010 2.5 Mbit/sec 011 8.0 Mbit/sec 100 Reserved 101 Reserved 110 Reserved 111 Reserved

Table 29-9. FlexRay channel selection

SCM	CHB	CHA	Description
Dual Channel Device modes			
0	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel B
Single Channel Device mode			
1	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ driven by CC — connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ not driven by CC
	1	1	Reserved

29.5.2.5 System Memory Base Address Register (FR_SYMBADR)

Base + 0x0004 Write: Disabled mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SMBA[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-4. System Memory Base Address High Register (FR_SYMBADHR)

Base + 0x0006 Write: Disabled mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-5. System Memory Base Address Low Register (FR_SYMBADLR)

NOTE

The system memory base address must be set before the CC is enabled.

The system memory base address registers define the base address of the FlexRay memory area within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 29-10. FR_SYMBADR field descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the individual message buffers and sync frame table. This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defined as a byte address.

29.5.2.6 Strobe Signal Control Register (FR_STBSCR)

Base + 0x0008 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SEL				0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-6. Strobe Signal Control Register (FR_STBSCR)

This register is used to assign the individual protocol timing-related strobe signals given in Table 29-12 to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed information, including timing, refer to Section 29.6.16, Strobe Signal Support.

NOTE

In single-channel device mode, channel B-related strobe signals are undefined and should not be assigned to the strobe ports.

Table 29-11. FR_STBSCR field descriptions

Field	Description
WMD	**Write mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in Table 29-12 to be enabled or disabled and assigned to one of the four strobe ports given in Table 29-12.

Table 29-11. FR_STBSCR field descriptions (continued)

Field	Description
ENB	**Strobe Signal Enable** — The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 Assign selected signal to FR_DBG[0]. 01 Assign selected signal to FR_DBG[1]. 10 Assign selected signal to FR_DBG[2]. 11 Assign selected signal to FR_DBG[3].

Table 29-12. Strobe signal mapping

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
0	0x0	Arm	—	value	+1	MT start
1	0x1	MT	—	value	+1	MT start
2	0x2	Cycle start	—	pulse	0	MT start
3	0x3	Minislot start	—	pulse	0	MT start
4	0x4	Slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	Receive data after glitch filtering	A	value	+4	FR_A_RX
7	0x7		B			FR_B_RX
8	0x8	Channel idle indicator	A	level	+5	FR_A_RX
9	0x9		B			FR_B_RX
10	0xA	Syntax error detected	A	pulse	+4	FR_A_RX
11	0xB		B			FR_B_RX
12	0xC	Content error detected	A	level	+4	FR_A_RX
13	0xD		B			FR_B_RX
14	0xE	Receive FIFO almost-full interrupt signals	A	value	n.a.	RX FIFO A Almost Full Interrupt
15	0xF		B			RX FIFO B Almost Full Interrupt

1 Given in PE clock cycles

29.5.2.7 Message Buffer Data Size Register (FR_MBDSR)

Base + 0x000C — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-7. Message Buffer Data Size Register (FR_MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The CC provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 29-13. FR_MBDSR field descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

29.5.2.8 Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

Base + 0x000E — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	LAST_MB_SEG1						0	0	LAST_MB_UTIL					
W																
Reset	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1

Figure 29-8. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 29-14. FR_MBSSUTR field descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers FR_MBCCSR*n*, FR_MBCCFR*n*, FR_MBFIDR*n*, FR_MBIDXR*n* with $n \leq$ LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1 + 1 individual message buffers. **Note:** The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers FR_MBCCSR*n*, FR_MBCCFR*n*, FR_MBFIDR*n*, FR_MBIDXR*n* with LAST_MB_SEG1 < *n* < 64. **Note:** If LAST_MB_SEG1 = 63 then all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of the individual message buffer that was used last. The message buffer search engine examines all individual message buffers that have a message buffer number $n \leq$ LAST_MB_UTIL. **Note:** If LAST_MB_UTIL = LAST_MB_SEG1, all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

29.5.2.9 PE DRAM Access Register (FR_PEDRAR)

Base + 0x0010 — 16-bit write access required — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INST				ADDR											DAD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-9. PE DRAM Access Register (FR_PEDRAR)

This register is used to trigger write and read operations on the PE data memory (PE DRAM). These operations are used for memory error injection and memory error observation.

Each write access to this register initiates a read or write operation on the PE DRAM. The access done status bit DAD is cleared after the write access and is set if the PE DRAM access has been finished.

In case of a PE DRAM write access, the data provided in PE DRAM Data Register (FR_PEDRDR) is written into the PE DRAM, read back from the PE DRAM, and stored into the PE DRAM Data Register (FR_PEDRDR).

In case of a PE DRAM read access, the requested data is read from PE DRAM and stored into the PE DRAM Data Register (FR_PEDRDR).

For a detailed description refer to Section 29.6.24, Memory Content Error Detection.

Table 29-15. FR_PEDRAR field descriptions

Field	Description
INST	**PE DRAM Access Instruction** — This field defines the operation to be executed on the PE DRAM. 0011 PE DRAM write: Write FR_PEDRDR[DATA] to PE DRAM address ADDR (16-bit). 0101 PE DRAM read: Read Data from PE DRAM address ADDR (16-bit) into FR_PEDRDR[DATA]. Other values are reserved.
ADDR	**PE DRAM Access Address** — This field defines the address in the PE DRAM to be written to or read from.
DAD	**PE DRAM Access Done** — This status bit is cleared when the application has written to this register and is set when the PE DRAM access has finished. 0 PE DRAM access running. 1 PE DRAM access done.

29.5.2.10 PE DRAM Data Register (FR_PEDRDR)

Base + 0x0012 — 16-bit write access required — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-10. PE DRAM Data Register (FR_PEDRDR)

This register provides the data to be written to or read from the PE DRAM by the access initiated by write access to the PE DRAM Access Register (FR_PEDRAR).

29.5.2.11 Protocol Operation Control Register (FR_POCR)

Base + 0x0014 — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-11. Protocol Operation Control Register (FR_POCR)

The application uses this register to issue

- Protocol control commands
- External clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see Section 29.7.6, Protocol Control Command Execution.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to Section 29.6.11, External Clock Synchronization.

Table 29-16. FR_POCR field descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field is used to trigger the application of the external offset correction value defined in the Protocol Configuration Register 29 (FR_PCR29). 00 Do not apply external offset correction value. 01 Reserved 10 Subtract external offset correction value. 11 Add external offset correction value.
ERC_AP	**External Rate Correction Application** — This field is used to trigger application of the external rate correction value defined in the Protocol Configuration Register 21 (FR_PCR21). 00 Do not apply external rate correction value. 01 Reserved 10 Subtract external rate correction value. 11 Add external rate correction value.
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The CC sets this status bit when the application has issued a protocol control command via the POCCMD field. The CC clears this status bit when a protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the CC ignores this command, sets the protocol command ignored error flag PCMI_EF in the CHI Error Flag Register (FR_CHIERFR), and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The CC sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed[1] transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Reserved

[1] Delayed means on completion of current communication cycle.

29.5.2.12 Global Interrupt Flag and Enable Register (FR_GIFER)

Base + 0x0016 — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUP IF	FAFB IF	FAFA IF	RBIF	TBIF	MIE	PRIE	CHIE	WUP IE	FAFB IE	FAFA IE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-12. Global Interrupt Flag and Enable Register (FR_GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in Figure 29-157. For more details on interrupt generation, see Section 29.6.20, Interrupt Support. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 29-17. FR_GIFER field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This interrupt flag is set if at least one of the other interrupt flags in this register and the related interrupt enable bit are set. 0 No interrupt flag and related interrupt enable bit are set. 1 At least one of the other interrupt flags in this register and the related interrupt bit are set.
PRIF	**Protocol Interrupt Flag** — This interrupt flag is set if at least one of the individual flags in the Protocol Interrupt Flag Register 0 (FR_PIFR0) and Protocol Interrupt Flag Register 1 (FR_PIFR1) and the related interrupt enable bit are set. 0 No individual protocol interrupt flag and related interrupt enable bit are set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable bit are set.
CHIF	**CHI Interrupt Flag** — This interrupt flag is set if at least one of the error flags in the CHI Error Flag Register (FR_CHIERFR) and the CHI error interrupt enable bit FR_GIFER[CHIE] are set. 0 All CHI error flags are equal to 0 or the CHI error interrupt is disabled. 1 At least one CHI error flag and the CHI error interrupt enable are set.
WUPIF	**Wakeup Interrupt Flag** — This interrupt flag is set when the CC has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the Protocol Status Register 3 (FR_PSR3). 0 No Wakeup symbol received on FlexRay bus. 1 Wakeup symbol received on FlexRay bus.
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs: • The current number of FIFO B entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (FR_RFWMSR), and the CC writes a received message into the FIFO B, or • The current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (FR_RFPTR) expires. 0 No such event. 1 FIFO B almost full event has occurred.

Table 29-17. FR_GIFER field descriptions (continued)

Field	Description
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs: • The current number of FIFO A entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (FR_RFWMSR), and the CC writes a received message into the FIFO A, or • The current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (FR_RFPTR) expires. 0 No such event. 1 FIFO A almost full event has occurred.
RBIF	**Receive Message Buffer Interrupt Flag** — This interrupt flag is set if, for at least one of the individual receive message buffers (FR_MBCCSR*n*[MTD] = 0), both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding **Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)** are asserted. The application cannot clear this interrupt flag directly — instead it is cleared by the CC when all of the interrupt flags (MBIF) of the individual receive message buffers are cleared by the application or if the application has cleared the related interrupt enables bit (MBIE). 0 None of the individual receive message buffers has the MBIF and MBIE flag set. 1 At least one individual receive message buffer has the MBIF and MBIE flag set.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if, for at least one of the individual message buffers (FR_MBCCSR*n*[MTD] = 1), both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding **Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)** are equal to 1. The application cannot clear this interrupt flag directly. Instead, this interrupt flag is cleared by the CC when either all of the individual interrupt flags (MBIF) of the individual transmit message buffers are cleared by the application, or the application has cleared the related interrupt enables bit (MBIE). 0 None of the individual transmit message buffers has the MBIF and MBIE flags set. 1 At least one individual transmit message buffer has the MBIF and MBIE flags set.
MIE	**Module Interrupt Enable** — This bit controls whether the Module Interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
PRIE	**Protocol Interrupt Enable** — This bit controls whether the Protocol Interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
CHIE	**CHI Interrupt Enable** — This bit controls whether the CHI Interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
WUPIE	**Wakeup Interrupt Enable** — This bit controls whether the Wakeup Interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This bit controls whether the RX FIFO B Almost Full Interrupt line is asserted when the FAFBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
FAFAIE	**Receive FIFO Channel A Almost Full Interrupt Enable** — This bit controls whether the RX FIFO A Almost Full Interrupt line is asserted when the FAFAIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.

Table 29-17. FR_GIFER field descriptions (continued)

Field	Description
RBIE	**Receive Message Buffer Interrupt Enable** — This bit controls whether the Receive Message Buffer Interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
TBIE	**Transmit Message Buffer Interrupt Enable** — This bit controls whether the Transmit Message Buffer Interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.

29.5.2.13 Protocol Interrupt Flag Register 0 (FR_PIFR0)

Base + 0x0018 — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-13. Protocol Interrupt Flag Register 0 (FR_PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 29-18. FR_PIFR0 field descriptions

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: • *pLatestTx* violation, as described in the MAC process of the FlexRay protocol • Transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the Protocol Configuration Register 14 (FR_PCR14) and Protocol Configuration Register 15 (FR_PCR15) when the CONFIG_COMPLETE command was sent by the application via the Protocol Operation Control Register (FR_POCR). If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.

Table 29-18. FR_PIFR0 field descriptions (continued)

Field	Description
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the Protocol Configuration Register 3 (FR_PCR3). 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event. 1 Insufficient number of measurements for rate correction detected.
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded their configured thresholds as given by the *offset_correction_out* field in the Protocol Configuration Register 9 (FR_PCR9) and the *rate_correction_out* field in the Protocol Configuration Register 14 (FR_PCR14). 0 No such event. 1 Offset or rate correction limit reached.
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the Protocol Configuration Register 30 (FR_PCR30). 0 No such event. 1 More than *node_sync_max* sync frames detected. **Note:** Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.

Table 29-18. FR_PIFR0 field descriptions (continued)

Field	Description
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event. 1 Timer 1 has reached its time limit.
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event. 1 Communication cycle started.

29.5.2.14 Protocol Interrupt Flag Register 1 (FR_PIFR1)

Base + 0x001A — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-14. Protocol Interrupt Flag Register 1 (FR_PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 29-19. FR_PIFR1 field descriptions

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the Protocol Status Register 0 (FR_PSR0) is changed by the CC. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command issued via the POCCMD field of the Protocol Operation Control Register (FR_POCR), and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see Section 29.7.6, Protocol Control Command Execution. 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the CC has detected a communication failure between the PE and the CHI. 0 No such event. 1 Protocol engine communication failure detected.

Table 29-19. FR_PIFR1 field descriptions (continued)

Field	Description
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the Protocol Status Register 0 (FR_PSR0) has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding Slot Status Counter Registers (FR_SSCR0–FR_SSCR3) is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the FlexRay memory area for the even cycle. 0 No such event. 1 Sync frame measurement table written.
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the FlexRay memory area for the odd cycle. 0 No such event. 1 Sync frame measurement table written.

29.5.2.15 Protocol Interrupt Enable Register 0 (FR_PIER0)

Base + 0x001C — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-15. Protocol Interrupt Enable Register 0 (FR_PIER0)

This register defines whether or not the individual interrupt flags defined in the Protocol Interrupt Flag Register 0 (FR_PIFR0) can generate a protocol interrupt request.

Table 29-20. FR_PIER0 field descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 Interrupt request generation disabled. 1 interrupt request generation enabled.
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.

Table 29-20. FR_PIER0 field descriptions (continued)

Field	Description
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
TBVB_IE	**Transmission across Boundary on Channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
TBVA_IE	**Transmission across Boundary on Channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.

29.5.2.16 Protocol Interrupt Enable Register 1 (FR_PIER1)

Base + 0x001E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PECF_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-16. Protocol Interrupt Enable Register 1 (FR_PIER1)

This register defines whether or not the individual interrupt flags defined in Protocol Interrupt Flag Register 1 (FR_PIFR1) can generate a protocol interrupt request.

Table 29-21. FR_PIER1 field descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.

29.5.2.17 CHI Error Flag Register (FR_CHIERFR)

Base + 0x0020 — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	0	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-17. CHI Error Flag Register (FR_CHIERFR)

This register holds the CHI-related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 29-22. FR_CHIERFR field descriptions

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event. 1 Frame lost on channel B detected.
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error. 1 Frame lost on channel A detected.
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the Protocol Operation Control Register (FR_POCR) while the BSY flag is equal to 1. In this case the command is ignored by the CC and is lost. 0 No such error. 1 POC command ignored.
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error. 1 FIFO overrun on channel B has been detected.
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error. 1 FIFO overrun on channel A has been detected.
MBS_EF	**Message Buffer Search Error Flag** — This flag is set if at least one of the following events occurs: • The message buffer search engine is still running while the next search must be started due to the FlexRay protocol timing. • A message buffer index greater than 67 is detected in the FR_MBIDXR[MBIDX] field of a found message buffer or in one of the FR_RSBIR[RSBIDX] fields. Refer to Section 29.6.7.4, Message Buffer Search Error, for details. 0 No such event. 1 Search engine active while search start occurs or illegal message buffer index detected.
MBU_EF	**Message Buffer Utilization Error Flag** — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR). If the application writes to a FR_MBCCSR*n* register with *n* > LAST_MB_UTIL, the CC ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the CHI Error Flag Register (FR_CHIERFR). 0 No such event. 1 Non-utilized message buffer enabled.

Table 29-22. FR_CHIERFR field descriptions (continued)

Field	Description
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the CC due to internal operations. In that case, the CC does not grant the lock to the application. The application must issue the lock request again. 0 No such error. 1 Lock error detected.
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see Section 29.6.19.1.2, System Bus Access Timeout). 0 No such event. 1 System bus access not finished in time.
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred. 1 Frame ID error occurred.
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment, which is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the Protocol Configuration Register 24 (FR_PCR24). 0 No such error occurred. 1 Dynamic payload length error occurred.
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the static segment is different from the payload length for the static segment, which is configured in the corresponding protocol configuration register field payload_length_static in the Protocol Configuration Register 19 (FR_PCR19). 0 No such error occurred. 1 Static payload length error occurred.
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the Network Management Vector Length Register (FR_NMVLR). In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred. 1 Network management length error occurred.
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see Network Management Vector Registers (FR_NMVR0–FR_NMVR5)) are not updated. 0 No such error occurred. 1 Network management frame error occurred.
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the CC (see Section 29.6.19.1.1, System Bus Illegal Address Access). 0 No such event. 1 Illegal system bus address accessed.

29.5.2.18 Message Buffer Interrupt Vector Register (FR_MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	TBIVEC						0	0	RBIVEC					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-18. Message Buffer Interrupt Vector Register (FR_MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 29-23. FR_MBIVEC field descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer that has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

29.5.2.19 Channel A Status Error Counter Register (FR_CASERCR)

Base + 0x0024 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-19. Channel A Status Error Counter Register (FR_CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol-related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see Section 29.6.18, Slot Status Monitoring.

Table 29-24. FR_CASERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

29.5.2.20 Channel B Status Error Counter Register (FR_CBSERCR)

Base + 0x0026 — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-20. Channel B Status Error Counter Register (FR_CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol-related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see Section 29.6.18, Slot Status Monitoring.

Table 29-25. FR_CBSERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

29.5.2.21 Protocol Status Register 0 (FR_PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-21. Protocol Status Register 0 (FR_PSR0)

This register provides information about the current protocol status.

Table 29-26. FR_PSR0 field descriptions

Field	Description
ERRMODE	**Error Mode** — Protocol-related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 Reserved
SLOTMODE	**Slot Mode** — Protocol-related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 Reserved

Table 29-26. FR_PSR0 field descriptions (continued)

Field	Description
PROTSTATE	**Protocol State** — Protocol-related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*
STARTUP STATE	**Startup State** — Protocol-related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 Reserved 0001 Reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integrationi listen* 0110 Reserved 0111 *POC:initialize schedule* 1000 Reserved 1001 Reserved 1010 *POC:coldstart consistency check* 1011 Reserved 1100 Reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — Protocol-related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 Reserved

29.5.2.22 Protocol Status Register 1 (FR_PSR1)

Base + 0x002A — Additional Reset: CSAA, CSP, CPN: RUN Command — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-22. Protocol Status Register 1 (FR_PSR1)

Table 29-27. FR_PSR1 field descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — Protocol-related event: 'set coldstart abort indicator in CHI'. This flag is set when the CC has aborted a cold start attempt. 0 No such event. 1 Cold start attempt aborted.
CSP	**Leading Cold Start Path** — This status bit is set when the CC has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event. 1 *POC:normal active* reached from *POC:startup* state via leading cold start path.
REMCSAT	**Remaining Coldstart Attempts** — Protocol-related variable: *vRemainingColdstartAttempts*. This field provides the number of remaining cold start attempts that the CC will execute.
CPN	**Leading Cold Start Path Noise** — Protocol-related variable: *vPOC!ColdstartNoise* This status bit is set if the CC has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the CC was starting up the cluster. 0 No such event. 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path.
HHR	**Host Halt Request Pending** — Protocol-related variable: *vPOC!CHIHaltRequest* This status bit is set when CC receives the HALT command from the application via the Protocol Operation Control Register (FR_POCR). The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event. 1 HALT command received.
FRZ	**Freeze Occurred** — Protocol-related variable: *vPOC!Freeze* This status bit is set when the CC has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event. 1 Immediate halt due to FREEZE or internal error condition.
APTAC	**Allow Passive to Active Counter** — Protocol-related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the Protocol Configuration Register 12 (FR_PCR12).

29.5.2.23 Protocol Status Register 2 (FR_PSR2)

Base + 0x002C — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-23. Protocol Status Register 2 (FR_PSR2)

This register provides a snapshot of status information about the Network Idle Time (NIT), the symbol window, and the clock synchronization. The NIT-related status bits NBVB, NSEB, NBVA, and NSEA are updated by the CC after the end of the NIT and before the end of the first slot of the next communication cycle. The symbol window-related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA

are updated by the CC after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window-related status bits remain in their reset state. The clock synchronization-related CLKCORRFAILCNT is updated by the CC after the end of the static segment and before the end of the current communication cycle.

Table 29-28. FR_PSR2 field descriptions

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — Protocol-related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event. 1 Media activity at boundaries detected.
NSEB	**NIT Syntax Error on Channel B** — Protocol-related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event. 1 Syntax error detected.
STCB	**Symbol Window Transmit Conflict on Channel B** — Protocol-related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event. 1 Transmission conflict detected.
SBVB	**Symbol Window Boundary Violation on Channel B** — Protocol-related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event. 1 Media activity at boundaries detected.
SSEB	**Symbol Window Syntax Error on Channel B** — Protocol-related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event. 1 Syntax error detected.
MTB	**Media Access Test Symbol MTS Received on Channel B** — Protocol-related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event. 1 MTS symbol received.
NBVA	**NIT Boundary Violation on Channel A** — Protocol-related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event. 1 Media activity at boundaries detected.
NSEA	**NIT Syntax Error on Channel A** — Protocol-related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event. 1 Syntax error detected.

Table 29-28. FR_PSR2 field descriptions (continued)

Field	Description
STCA	**Symbol Window Transmit Conflict on Channel A** — Protocol-related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflict during the symbol window on channel A. 0 No such event. 1 Transmission conflict detected.
SBVA	**Symbol Window Boundary Violation on Channel A** — Protocol-related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event. 1 Media activity at boundaries detected.
SSEA	**Symbol Window Syntax Error on Channel A** — Protocol-related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event. 1 Syntax error detected.
MTA	**Media Access Test Symbol MTS Received on Channel A** — Protocol-related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 0 No such event. 1 MTS symbol received.
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — Protocol-related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the Protocol Configuration Register 8 (FR_PCR8). The CC resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

29.5.2.24 Protocol Status Register 3 (FR_PSR3)

Base + 0x002E — Additional Reset: RUN Command — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-24. Protocol Status Register 3 (FR_PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 29-29. FR_PSR3 field descriptions

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received. 1 Wakeup symbol received.
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected. 1 Boundary violation detected.
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected. 1 Additional communication detected.
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected. 1 Content error detected.
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected. 1 Syntax errors detected.
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 0 No syntactically valid frames received. 1 At least one syntactically valid frame received.
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received. 1 Wakeup symbol received.
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected. 1 Boundary violation detected.
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected. 1 Additional communication detected.
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected. 1 Content error detected.

Table 29-29. FR_PSR3 field descriptions (continued)

Field	Description
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected. 1 Syntax errors detected.
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received. 1 At least one syntactically valid frame received.

29.5.2.25 Macrotick Counter Register (FR_MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-25. Macrotick Counter Register (FR_MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 29-30. FR_MTCTR field descriptions

Field	Description
MTCT	**Macrotick Counter** — Protocol-related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

29.5.2.26 Cycle Counter Register (FR_CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-26. Cycle Counter Register (FR_CYCTR)

This register provides the number of the current communication cycle.

Table 29-31. FR_CYCTR field descriptions

Field	Description
CYCCNT	**Cycle Counter** — Protocol-related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

29.5.2.27 Slot Counter Channel A Register (FR_SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-27. Slot Counter Channel A Register (FR_SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 29-32. FR_SLTCTAR field descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — Protocol-related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

29.5.2.28 Slot Counter Channel B Register (FR_SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-28. Slot Counter Channel B Register (FR_SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 29-33. FR_SLTCTBR field descriptions

Field	Description
SLOTCNTB	**Slot Counter Value for Channel B** — Protocol-related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

29.5.2.29 Rate Correction Value Register (FR_RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RATECORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-29. Rate Correction Value Register (FR_RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT of each odd numbered communication cycle.

Table 29-34. FR_RTCORVR field descriptions

Field	Description
RATECORR	**Rate Correction Value** — Protocol-related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the Protocol Configuration Register 13 (FR_PCR13), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (FR_PIFR0). **Note:** If the CC was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

29.5.2.30 Offset Correction Value Register (FR_OFCORVR)

Base + 0x003A — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OFFSETCORR															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-30. Offset Correction Value Register (FR_OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT.

Table 29-35. FR_OFCORVR field descriptions

Field	Description
OFFSETCORR	**Offset Correction Value** — Protocol-related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the Protocol Configuration Register 29 (FR_PCR29), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (FR_PIFR0). **Note:** If the CC was not able to calculate a new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

29.5.2.31 Combined Interrupt Flag Register (FR_CIFR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUP IF	FAFB IF	FAFA IF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-31. Combined Interrupt Flag Register (FR_CIFR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in Figure 29-159. The individual interrupt flags WUPIF, FAFBIF, and FAFAIF are copies of corresponding flags in the Global Interrupt Flag and Enable Register (FR_GIFER) and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the Global Interrupt Flag and Enable Register (FR_GIFER).

NOTE

The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 29-36. FR_CIFR field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted. 1 At least one interrupt source has its interrupt flag asserted.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (FR_PIFR0) or Protocol Interrupt Flag Register 1 (FR_PIFR1) is equal to 1. 0 All individual protocol interrupt flags are equal to 0. 1 At least one of the individual protocol interrupt flags is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (FR_CHIERFR) is equal to 1. 0 All CHI error flags are equal to 0. 1 At least one CHI error flag is equal to 1.
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as FR_GIFER[WUPIF].
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFBIF].
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFAIF].
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (FR_MBCCSR*n*[MTD] = 0) the interrupt flag MBIF in the corresponding **Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)** is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffer has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual transmit message buffers (FR_MBCCSR*n*[MTD] = 1) the interrupt flag MBIF in the corresponding **Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)** is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffer has the MBIF flag asserted.

29.5.2.32 System Memory Access Time-Out Register (FR_SYMATOR)

Base + 0x003E — Write: Disabled mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TIMEOUT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 29-32. System Memory Access Time-Out Register (FR_SYMATOR)

Table 29-37. FR_SYMATOR field descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines when a system bus access timout is detected. For a detailed description see Section 29.7.1.1, Configure System Memory Access Time-Out Register (FR_SYMATOR), and Section 29.6.19.1.2, System Bus Access Timeout.

29.5.2.33 Sync Frame Counter Register (FR_SFCNTR)

Base + 0x0040 — Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-33. Sync Frame Counter Register (FR_SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

NOTE

If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the CC will not update the fields SFEVB and SFEVA.

If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the CC will not update the values SFODB and SFODA.

Table 29-38. FR_SFCNTR field descriptions

Field	Description
SFEVB	**Sync Frames Channel B**, even cycle — Protocol-related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	**Sync Frames Channel A**, even cycle — Protocol-related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

Table 29-38. FR_SFCNTR field descriptions (continued)

Field	Description
SFODB	**Sync Frames Channel B**, odd cycle — Protocol-related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	**Sync Frames Channel A**, odd cycle — Protocol-related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

29.5.2.34 Sync Frame Table Offset Register (FR_SFTOR)

Base + 0x0042 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFT_OFFSET[15:1]															0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-34. Sync Frame Table Offset Register (FR_SFTOR)

This register defines the FlexRay memory-area-related offset for sync frame tables. For more details, see Section 29.6.12, Sync Frame ID and Sync Frame Deviation Tables.

Table 29-39. FR_SFTOR field description

Field	Description
SFT_OFFSET	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the FlexRay memory area. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

29.5.2.35 Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

Base + 0x0044 — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDV EN	SID EN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-35. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see Section 29.6.12, Sync Frame ID and Sync Frame Deviation Tables.

Table 29-40. FR_SFTCCSR field descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect. 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect. 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing). 1 Tables are valid (consistent).
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing). 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the CC writes continuously or only one pair of Sync Frame Tables into the FlexRay memory area. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the CC writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FlexRay memory area. In this case, the CC clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the CC writes continuously the enabled Sync Frame Tables into the FlexRay memory area. 0 Continuously write pairs of enabled Sync Frame Tables into FlexRay memory area. 1 Write only one pair of enabled Sync Frame Tables into FlexRay memory area.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the CC to write the Sync Frame Deviation Tables into the FlexRay memory area. 0 Do not write Sync Frame Deviation Tables. 1 Write Sync Frame Deviation Tables into FlexRay memory area. **Note:** If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the CC to write the Sync Frame ID Tables into the FlexRay memory area. 0 Do not write Sync Frame ID Tables. 1 Write Sync Frame ID Tables into FlexRay memory area.

29.5.2.36 Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

Base + 0x0046 — 16-bit write access required — Write: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-36. Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the CC accepts the sync frame in the current cycle.

Table 29-41. FR_SFIDRFR field descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see Section 29.6.15.2, Sync Frame Rejection Filtering.

29.5.2.37 Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

Base + 0x0048 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FVAL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-37. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see Section 29.6.15, Sync Frame Filtering.

Table 29-42. FR_SFIDAFVR field descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

29.5.2.38 Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

Base + 0x004A — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FMSK									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-38. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see Section 29.6.15.1, Sync Frame Acceptance Filtering.

Table 29-43. FR_SFIDAFMR field descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

29.5.2.39 Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Base + 0x004C (FR_NMVR0)
Base + 0x004E (FR_NMVR1)
Base + 0x0050 (FR_NMVR2)
Base + 0x0052 (FR_NMVR3)
Base + 0x0054 (FR_NMVR4)
Base + 0x0056 (FR_NMVR5)

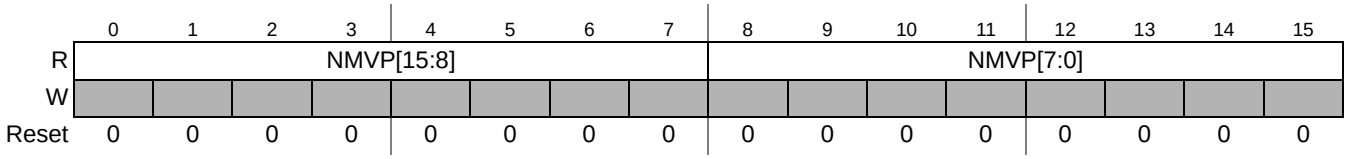


Figure 29-39. Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the Network Management Vector Length Register (FR_NMVLR). If FR_NMVLR is programmed with a value that is less than twelve bytes, the remaining bytes of the Network Management Vector Registers (FR_NMVR0–FR_NMVR5), which are not used for the Network Management Vector accumulating, will remain zero.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 29-44. NMVR[0:5] field descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the Network Management Vector Registers (FR_NMVR0–FR_NMVR5) and the receive message buffer payload bytes in NMV[0:11] is depicted in **Table 29-45**.

Table 29-45. Mapping of NMVR*n* to the received payload bytes NMV*n*

NMVR*n* register	NMV*n* received payload
FR_NMVR0[NMVP[15:8]]	NMV0
FR_NMVR0[NMVP[7:0]]	NMV1
FR_NMVR1[NMVP[15:8]]	NMV2
FR_NMVR1[NMVP[7:0]]	NMV3
...	
FR_NMVR5[NMVP[15:8]]	NMV10
FR_NMVR5[NMVP[7:0]]	NMV11

29.5.2.40 Network Management Vector Length Register (FR_NMVLR)

Base + 0x0058 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-40. Network Management Vector Length Register (FR_NMVLR)

This register defines the length of the network management vector in bytes.

Table 29-46. FR_NMVLR field descriptions

Field	Description
NMVL	**Network Management Vector Length** — Protocol-related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

29.5.2.41 Timer Configuration and Control Register (FR_TICCR)

Base + 0x005A — Write: T2_CFG: *POC:config*; T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_CFG	T2_REP	0	0	0	T2ST	0	0	0	T1_REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-41. Timer Configuration and Control Register (FR_TICCR)

This register is used to configure and control the two timers T1 and T2. For timer details, see Section 29.6.17, Timer Support. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 29-47. FR_TICCR field descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the timebase mode of Timer T2. 0 Timer T2 is absolute timer. 1 Timer T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 Timer T2 is non-repetitive. 1 Timer T2 is repetitive.
T2SP	**Timer T2 Stop** — This trigger bit is used to stop timer T2. 0 No effect. 1 Stop timer T2.
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 No effect. 1 Start timer T2.

Table 29-47. FR_TICCR field descriptions (continued)

Field	Description
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 Timer T2 is idle. 1 Timer T2 is running.
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 Timer T1 is non-repetitive. 1 Timer T1 is repetitive.
T1SP	**Timer T1 Stop** — This trigger bit is used to stop timer T1. 0 No effect. 1 Stop timer T1.
T1TR	**Timer T1 Trigger** — This trigger bit is used to start timer T1. 0 No effect. 1 Start timer T1.
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 Timer T1 is idle. 1 Timer T1 is running.

NOTE

Both timers are deactivated immediately when the protocol enters a state different from *POC:normal active* or *POC:normal passive*.

29.5.2.42 Timer 1 Cycle Set Register (FR_TI1CYSR)

Base + 0x005C — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-42. Timer 1 Cycle Set Register (FR_TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to Section 29.6.17.1, Absolute Timer T1.

Table 29-48. FR_TI1CYSR field descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

29.5.2.43 Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Base + 0x005E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-43. Timer 1 Macrotick Offset Register (FR_TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to Section 29.6.17.1, Absolute Timer T1.

Table 29-49. FR_TI1MTOR field descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

29.5.2.44 Timer 2 Configuration Register 0 (FR_TI2CR0)

Base + 0x0060 — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*		T2_CYC_VAL						R*		T2_CYC_MSK					
W																
R	T2_MTCNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-44. Timer 2 Configuration Register 0 (FR_TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (FR_TICCR). For a detailed description of timer T2, refer to Section 29.6.17.2, Absolute / Relative Timer T2.

Table 29-50. FR_TI2CR0 field descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

29.5.2.45 Timer 2 Configuration Register 1 (FR_TI2CR1)

Base + 0x0062 — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*				T2_MTOFFSET											
W																
R	T2_MTCNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-45. Timer 2 Configuration Register 1 (FR_TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (FR_TICCR). For a detailed description of timer T2, refer to Section 29.6.17.2, Absolute / Relative Timer T2.

Table 29-51. FR_TI2CR1 field descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

29.5.2.46 Slot Status Selection Register (FR_SSSR)

Base + 0x0064 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	SLOTNUMBER										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-46. Slot Status Selection Register (FR_SSSR)

This register is used to access the four internal non-memory mapped slot status selection registers FR_SSSR0–FR_SSSR3. Each internal register selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding Slot Status Registers (FR_SSR0–FR_SSR7) according to Table 29-53. For a detailed description of slot status monitoring, refer to Section 29.6.18, Slot Status Monitoring.

Table 29-52. FR_SSSR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 Select FR_SSSR0. 01 Select FR_SSSR1. 10 Select FR_SSSR2. 11 Select FR_SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. **Note:** If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 29-53. Mapping between FR_SSSR*n* and FR_SSR*n*

Internal slot status selection register	Write the Slot Status of the slot selected by FR_SSSRn for each			
	Even communication cycle		Odd communication cycle	
	For channel B to	For channel A to	For channel B to	For channel A to
FR_SSSR0	FR_SSR0[15:8]	FR_SSR0[7:0]	FR_SSR1[15:8]	FR_SSR1[7:0]
FR_SSSR1	FR_SSR2[15:8]	FR_SSR2[7:0]	FR_SSR3[15:8]	FR_SSR3[7:0]
FR_SSSR2	FR_SSR4[15:8]	FR_SSR4[7:0]	FR_SSR5[15:8]	FR_SSR5[7:0]
FR_SSSR3	FR_SSR6[15:8]	FR_SSR6[7:0]	FR_SSR7[15:8]	FR_SSR7[7:0]

29.5.2.47 Slot Status Counter Condition Register (FR_SSCCR)

Base + 0x0066 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-47. Slot Status Counter Condition Register (FR_SSCCR)

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers FR_SSCCR0 to FR_SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding Slot Status Counter Registers (FR_SSCR0–FR_SSCR3). The correspondence is given in Table 29-55. For a detailed description of slot status counters, refer to Section 29.6.18.4, Slot Status Counter Registers.

Table 29-54. FR_SSCCR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 Select FR_SSCCR0. 01 Select FR_SSCCR1. 10 Select FR_SSCCR2. 11 Select FR_SSCCR3.
CNTCFG	**Counter Configuration** — This bit field controls the channel-related incrementing of the slot status counter. 00 Increment by 1 if condition is fulfilled on channel A. 01 Increment by 1 if condition is fulfilled on channel B. 10 Increment by 1 if condition is fulfilled on at least one channel. 11 Increment by 2 if condition is fulfilled on both channels. Increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.

Table 29-54. FR_SSCCR field descriptions (continued)

Field	Description
SUF	**Startup Frame Restriction** — This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUSMASK[3:0]	**Slot Status Mask** — This bit field is used to enable the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 29-55. Mapping between internal FR_SSCCR*n* and FR_SSCR*n*

Condition register	Condition defined for register
FR_SSCCR0	FR_SSCR0
FR_SSCCR1	FR_SSCR1
FR_SSCCR2	FR_SSCR2
FR_SSCCR3	FR_SSCR3

29.5.2.48 Slot Status Registers (FR_SSR0–FR_SSR7)

Base + 0x0068 (FR_SSR0)
Base + 0x006A (FR_SSR1)
Base + 0x006C (FR_SSR2)
Base + 0x006E (FR_SSR3)
Base + 0x0070 (FR_SSR4)
Base + 0x0072 (FR_SSR5)
Base + 0x0074 (FR_SSR6)
Base + 0x0076 (FR_SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-48. Slot Status Registers (FR_SSR0–FR_SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the Slot Status Selection Register (FR_SSSR). Each register is updated after the end of the corresponding slot as shown in Figure 29-155. The register bits are directly related to the protocol variables and described in more detail in Section 29.6.18, Slot Status Monitoring.

Table 29-56. FR_SSR0–FR_SSR7 field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — Protocol-related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — Protocol-related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — Protocol-related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — Protocol-related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — Protocol-related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — Protocol-related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — Protocol-related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — Protocol-related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1

Table 29-56. FR_SSR0–FR_SSR7 field descriptions (continued)

Field	Description
BVA	**Boundary Violation on Channel A** — Protocol-related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — Protocol-related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

29.5.2.49 Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Base + 0x0078 (FR_SSCR0)
Base + 0x007A (FR_SSCR1)
Base + 0x007C (FR_SSCR2)
Base + 0x007E (FR_SSCR3)

Additional Reset: RUN Command

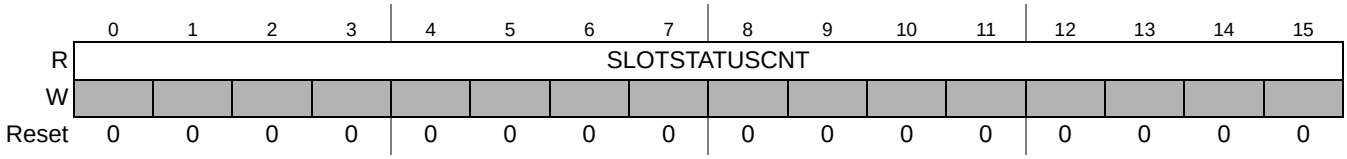


Figure 29-49. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register FR_SSCCR*n*, which can be programmed by using the Slot Status Counter Condition Register (FR_SSCCR). For more details, see Section 29.6.18.4, Slot Status Counter Registers.

NOTE

If the counter has reached its maximum value 0xFFFF and is in the multicycle mode (FR_SSCCR*n*[MCY] = 1), the counter is not reset to 0x0000. The application can reset the counter by clearing the FR_SSCCR*n*[MCY] bit and waiting for the next cycle start, when the CC clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 29-57. FR_SSCR0–FR_SSCR3 field descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

29.5.2.50 MTS A Configuration Register (FR_MTSACFR)

Base + 0x0080

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-50. MTS A Configuration Register (FR_MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see Section 29.6.13, MTS Generation.

Table 29-58. FR_MTSACFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	**Cycle Counter Mask** — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

29.5.2.51 MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-51. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see Section 29.6.13, MTS Generation.

Table 29-59. MTSBCFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	**Cycle Counter Mask** — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

29.5.2.52 Receive Shadow Buffer Index Register (FR_RSBIR)

Base + 0x0084 — 16-bit write access required — Write: WMD, SEL: Any Time; RSBIDX: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	0	$RSBIDX_{A1}$/$RSBIDX_{A2}$						
W	WMD									$RSBIDX_{B1}$/$RSBIDX_{B2}$						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-52. Receive Shadow Buffer Index Register (FR_RSBIR)

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to Section 29.6.6.3.5, Receive Shadow Buffers Concept.

Table 29-60. FR_RSBIR field descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 Update SEL and RSBIDX field on register write. 1 Update only SEL field on register write.
SEL	**Selector** — This field is used to select the internal receive shadow buffer index register for access. 00 FR_RSBIR_A1 — Receive shadow buffer index register for channel A, segment 1. 01 FR_RSBIR_A2 — Receive shadow buffer index register for channel A, segment 2. 10 FR_RSBIR_B1 — Receive shadow buffer index register for channel B, segment 1. 11 FR_RSBIR_B2 — Receive shadow buffer index register for channel B, segment 2.
$RSBIDX_{A1}$ $RSBIDX_{A2}$ $RSBIDX_{B1}$ $RSBIDX_{B2}$	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The CC uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory area. The CC will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. CC: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index. Legal Values are $0 \le i \le 67$. Illegal values will be detected during the message buffer search.

29.5.2.53 Receive FIFO Start Data Offset Register (FR_RFSDOR)

Base + 0x00E6 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SDO_A/SDO_B															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-53. Receive FIFO Start Data Offset Register (FR_RFSDOR)

Table 29-61. FR_RFSDOR field descriptions

Field	Description
SDO_A SDO_B	**Start Data Field Offset** — This field defines the data field offset of the header field of the first message buffer of the selected FIFO. The CC uses the value of the SDO field to determine the physical location of the receiver FIFO's first message buffer header field. For configuration constraints see Section 29.7.1.2, Configure Data Field Offsets.

NOTE

Since all data fields of the FIFO are of equal length and are located at subsequent system memory addresses, the content of the FR_RFSDOR register corresponds to the start address of payload area of the selected FIFO.

29.5.2.54 Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)

Base + 0x00E8 — Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-54. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)

Base + 0x00EA — Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-55. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)

These registers define the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 29-62. FR_RFSYMBADR field descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defined as a byte address.

29.5.2.55 Receive FIFO Periodic Timer Register (FR_RFPTR)

Base + 0x00EC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PTD													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-56. Receive FIFO Periodic Timer Register (FR_RFPTR)

This register holds periodic timer duration for the periodic FIFO timer. The periodic timer applies to both FIFOs (see Section 29.6.9.3, FIFO Periodic Timer).

Table 29-63. FR_RFPTR field descriptions

Field	Description
PTD	**Periodic Timer Duration** — This value defines the periodic timer duration in terms of macroticks. 0000 Timer stays expired. 3FFF Timer never expires. Other Timer expires after specified number of macroticks, expires, and is restarted at each cycle start.

29.5.2.56 Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Base + 0x0086 — Write: *WM_A/WM_B: POC:config, SEL: Anytime*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WM_A/WM_B								0	0	0	0	0	0	0	SEL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-57. Receive FIFO Watermark and Selection Register (FR_RFWMSR)

This register is used to:

- Select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in Table 29-64.
- Define the watermark for the selected FIFO.

Table 29-64. SEL controlled receiver FIFO registers

Register
Receive FIFO Start Index Register (FR_RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)
Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Table 29-65. FR_RFWMSR field descriptions

Field	Description
WM_A WM_B	**Watermark** — This field defines the watermark value for the selected FIFO. This value is used to control the generation of the almost full interrupt flags.
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected. 1 Receiver FIFO for channel B selected.

29.5.2.57 Receive FIFO Start Index Register (FR_RFSIR)

Base + 0x0088 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	$SIDX_A/SIDX_B$									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-58. Receive FIFO Start Index Register (FR_RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 29-66. FR_RFSIR field descriptions

Field	Description
$SIDX_A$ $SIDX_B$	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected FIFO. The CC uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

29.5.2.58 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	$FIFO\_DEPTH_A/FIFO\_DEPTH_B$								0	$ENTRY\_SIZE_A/ENTRY\_SIZE_B$						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-59. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, that is, the number of entries and the size of each entry.

Table 29-67. RFDSR field descriptions

Field	Description
$FIFO\_DEPTH_A$ $FIFO\_DEPTH_B$	**FIFO Depth** — This field defines the depth of the selected FIFO, i.e., the number of entries. **Note:** If the FIFO_DEPTH is configured to 0, FR_RFFIDRFMR[FIDRFMSK] must also be configured to 0 to ensure that no frames are received into the FIFO.
$ENTRY\_SIZE_A$ $ENTRY\_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

29.5.2.59 Receive FIFO A Read Index Register (FR_RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-60. Receive FIFO A Read Index Register (FR_RFARIR)

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 29-68. FR_RFARIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

NOTE

If the FIFO is empty, the RDIDX field points to a physical message buffer with invalid content.

29.5.2.60 Receive FIFO B Read Index Register (FR_RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-61. Receive FIFO B Read Index Register (FR_RFBRIR)

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 29-69. FR_RFBRIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

NOTE

If the FIFO is empty, the RDIDX field points to a physical message buffer with invalid content.

29.5.2.61 Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Base + 0x00EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FLB								FLA							
W	PCB								PCA							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-62. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

This register provides the current fill level of the two receiver FIFOs and is used to pop a number of entries from the FIFOs.

Table 29-70. FR_RFFLPCR field descriptions

Field	Description
FLB	**Fill Level FIFO B** — This field provides the current number of entries in the FIFO B.
FLA	**Fill Level FIFO A**— This field provides the current number of entries in the FIFO A.
PCB	**Pop Count FIFO B** — This field defines the number of entries to be removed from FIFO B.
PCA	**Pop Count FIFO A**— This field defines the number of entries to be removed from FIFO A.

NOTE

If the pop count value PCA/PCB is greater than the current FIFO fill level FLB/FLA, then the FIFO is empty after the update. No notification is given that a number different than the required number of entries was removed.

29.5.2.62 Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Base + 0x0090 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$MIDAFVAL_A$/$MIDAFVAL_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-63. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 29.6.9.9, FIFO Filtering.

Table 29-71. FR_RFMIDAFVR field descriptions

Field	Description
$MIDAFVAL_A$ $MIDAFVAL_B$	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

29.5.2.63 Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

Base + 0x0092 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$MIDAFMSK_A$/$MIDAFMSK_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-64. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 29.6.9.9, FIFO Filtering.

Table 29-72. FR_RFMIDAFMR field descriptions

Field	Description
$MIDAFMSK_A$ $MIDAFMSK_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

29.5.2.64 Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	$FIDRFVAL_A$/$FIDRFVAL_B$										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-65. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 29.6.9.9, FIFO Filtering.

Table 29-73. FR_RFFIDRFVR field descriptions

Field	Description
$FIDRFVAL_A$ $FIDRFVAL_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

29.5.2.65 Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	$FIDRFMSK_A$/$FIDRFMSK_B$										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-66. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 29.6.9.9, FIFO Filtering.

Table 29-74. FR_RFFIDRFMR field descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

29.5.2.66 Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Base + 0x0098 — 16-bit write access required — Write: WMD, IBD, SEL: Any Time; SID: *POC:config*

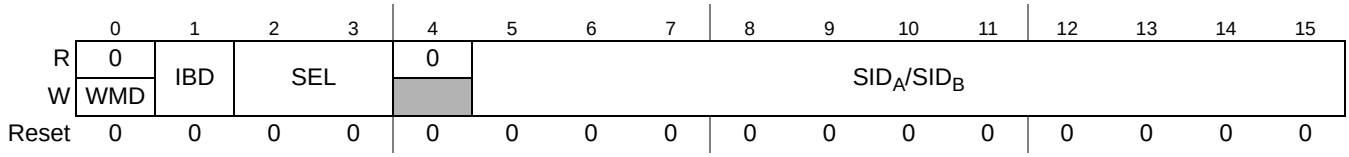


Figure 29-67. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected FIFO. For details on frame ID range filter see Section 29.6.9.9, FIFO Filtering.

Table 29-75. FR_RFRFCFR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 Program lower interval boundary. 1 Program upper interval boundary.
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 Select frame ID range filter 0. 01 Select frame ID range filter 1. 10 Select frame ID range filter 2. 11 Select frame ID range filter 3.
SID_A SID_B	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

29.5.2.67 Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Base + 0x009A — Write: Anytime

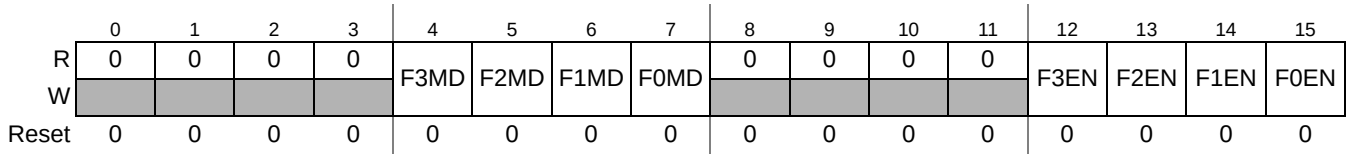


Figure 29-68. Receive FIFO Range Filter Control Register (FR_RFRFCTR)

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 29-76. FR_RFRFCTR field descriptions

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 Range filter 3 runs as acceptance filter. 1 Range filter 3 runs as rejection filter.
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 Range filter 2 runs as acceptance filter. 1 Range filter 2 runs as rejection filter.
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 Range filter 1 runs as acceptance filter. 1 Range filter 1 runs as rejection filter.
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 Range filter 0 runs as acceptance filter. 1 Range filter 0 runs as rejection filter.
F3EN	**Range Filter 3 Enable** — This control bit is used to enable and disable the frame ID range filter 3. 0 Range filter 3 disabled. 1 Range filter 3 enabled.
F2EN	**Range Filter 2 Enable** — This control bit is used to enable and disable the frame ID range filter 2. 0 Range filter 2 disabled. 1 Range filter 2 enabled.
F1EN	**Range Filter 1 Enable** — This control bit is used to enable and disable the frame ID range filter 1. 0 Range filter 1 disabled. 1 Range filter 1 enabled.
F0EN	**Range Filter 0 Enable** — This control bit is used to enable and disable the frame ID range filter 0. 0 Range filter 0 disabled. 1 Range filter 0 enabled.

29.5.2.68 Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-69. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 29-77. FR_LDTXSLAR field descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — Protocol-related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

29.5.2.69 Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-70. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 29-78. FR_LDTXSLBR field descriptions

Field	Description
LASTDYNTXSLOTB	**Last Dynamic Transmission Slot Channel B** — Protocol-related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

29.5.2.70 Protocol Configuration Registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in Table 29-79. For more details about the FlexRay-related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

Table 29-79. Protocol configuration register fields

Name	Description[1]	Min	Max	Unit	FR_PCR
coldstart_attempts	*gColdstartAttempts*			number	3
action_point_offset	*gdActionPointOffset* – 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* – 1			*gdBit*	4
dynamic_slot_idle_phase	*gdDynamicSlotIdlePhase*			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* – 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* – 1			MT	2
static_slot_length	*gdStaticSlot*			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* – 1			MT	13
symbol_window_exists	*gdSymbolWindow* ≠ 0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* – 1			MT	6
tss_transmitter	*gdTSSTransmitter*			*gdBit*	5
wakeup_symbol_rx_idle	*gdWakeupSymbolRxIdle*			*gdBit*	5
wakeup_symbol_rx_low	*gdWakeupSymbolRxLow*			*gdBit*	3
wakeup_symbol_rx_window	*gdWakeupSymbolRxWindow*			*gdBit*	4

Table 29-79. Protocol configuration register fields (continued)

Name	Description[1]	Min	Max	Unit	FR_PCR
wakeup_symbol_tx_idle	*gdWakeupSymbolTxIdle*			*gdBit*	8
wakeup_symbol_tx_low	*gdWakeupSymbolTxLow*			*gdBit*	5
noise_listen_timeout	(*gListenNoise* × *pdListenTimeout*) – 1			µT	16/17
macro_initial_offset_a	*pMacroInitialOffset[A]*			MT	6
macro_initial_offset_b	*pMacroInitialOffset[B]*			MT	16
macro_per_cycle	*gMacroPerCycle*			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	*gMaxWithoutClockCorrectionFatal*			cyclepairs	8
max_without_clock_correction_passive	*gMaxWithoutClockCorrectionPassive*			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	*gNumberOfStaticSlots*			static slot	2
offset_correction_start	*gOffsetCorrectionStart*			MT	11
payload_length_static	*gPayloadLengthStatic*			2-bytes	19
max_payload_length_dynamic	*pPayloadLengthDynMax*			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	*pAllowHaltDueToClock*			bool	26
allow_passive_to_active	*pAllowPassiveToActive*			cyclepairs	12
cluster_drift_damping	*pClusterDriftDamping*			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	*pKeySlotId*			number	18
key_slot_used_for_startup	*pKeySlotUsedForStartup*			bool	11
key_slot_used_for_sync	*pKeySlotUsedForSync*			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	*gSyncNodeMax*			number	30
micro_initial_offset_a	*pMicroInitialOffset[A]*			µT	20
micro_initial_offset_b	*pMicroInitialOffset[B]*			µT	20
micro_per_cycle	*pMicroPerCycle*			µT	22/23
micro_per_cycle_min	*pMicroPerCycle - pdMaxDrift*			µT	24/25
micro_per_cycle_max	*pMicroPerCycle + pdMaxDrift*			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	*pOffsetCorrectionOut*			µT	9

Table 29-79. Protocol configuration register fields (continued)

Name	Description[1]	Min	Max	Unit	FR_PCR
rate_correction_out	*pRateCorrectionOut*			µT	14
single_slot_enabled	*pSingleSlotEnabled*			bool	10
wakeup_channel	*pWakeupChannel*	see Table 29-80			10
wakeup_pattern	*pWakeupPattern*			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	*pExternOffsetCorrection*			µT	29
extern_rate_correction	*pExternRateCorrection*			µT	21

[1] See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 29-80. Wakeup channel selection

wakeup_channel	Wakeup channel
0	A
1	B

29.5.2.70.1 Protocol Configuration Register 0 (FR_PCR0)

Base + 0x00A0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	action_point_offset								static_slot_length							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-71. Protocol Configuration Register 0 (FR_PCR0)

29.5.2.70.2 Protocol Configuration Register 1 (FR_PCR1)

Base + 0x00A2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	macro_after_first_static_slot													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-72. Protocol Configuration Register 1 (FR_PCR1)

29.5.2.70.3 Protocol Configuration Register 2 (FR_PCR2)

Base + 0x00A4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	minislot_after_action_point								number_of_static_slots							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-73. Protocol Configuration Register 2 (FR_PCR2)

29.5.2.70.4 Protocol Configuration Register 3 (FR_PCR3)

Base + 0x00A6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	wakeup_symbol_rx_low						minislot_action_point_offset[4:0]						coldstart_attempts			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-74. Protocol Configuration Register 3 (FR_PCR3)

29.5.2.70.5 Protocol Configuration Register 4 (FR_PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	cas_rx_low_max								wakeup_symbol_rx_window							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-75. Protocol Configuration Register 4 (FR_PCR4)

29.5.2.70.6 Protocol Configuration Register 5 (FR_PCR5)

Base + 0x00AA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	tss_transmitter				wakeup_symbol_tx_low						wakeup_symbol_rx_idle					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-76. Protocol Configuration Register 5 (FR_PCR5)

29.5.2.70.7 Protocol Configuration Register 6 (FR_PCR6)

Base + 0x00AC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point							macro_initial_offset_a							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-77. Protocol Configuration Register 6 (FR_PCR6)

29.5.2.70.8 Protocol Configuration Register 7 (FR_PCR7)

Base + 0x00AE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	decoding_correction_b								micro_per_macro_nom_half							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-78. Protocol Configuration Register 7 (FR_PCR7)

29.5.2.70.9 Protocol Configuration Register 8 (FR_PCR8)

Base + 0x00B0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	max_without_clock_correction_fatal				max_without_clock_correction_passive				wakeup_symbol_tx_idle							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-79. Protocol Configuration Register 8 (FR_PCR8)

29.5.2.70.10 Protocol Configuration Register 9 (FR_PCR9)

Base + 0x00B2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	mini slot_ exists	symbol_ window_ exists	offset_correction_out													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-80. Protocol Configuration Register 9 (FR_PCR9)

29.5.2.70.11 Protocol Configuration Register 10 (FR_PCR10)

Base + 0x00B4 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	single_slot_ enabled	wakeup_ channel	macro_per_cycle													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-81. Protocol Configuration Register 10 (FR_PCR10)

29.5.2.70.12 Protocol Configuration Register 11 (FR_PCR11)

Base + 0x00B6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	key_slot_used_for_startup	key_slot_used_for_sync	offset_correction_start													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-82. Protocol Configuration Register 11 (FR_PCR11)

29.5.2.70.13 Protocol Configuration Register 12 (FR_PCR12)

Base + 0x00B8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	allow_passive_to_active				key_slot_header_crc											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-83. Protocol Configuration Register 12 (FR_PCR12)

29.5.2.70.14 Protocol Configuration Register 13 (FR_PCR13)

Base + 0x00BA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	first_minislot_action_point_offset						static_slot_after_action_point									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-84. Protocol Configuration Register 13 (FR_PCR13)

29.5.2.70.15 Protocol Configuration Register 14 (FR_PCR14)

Base + 0x00BC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	rate_correction_out											listen_timeout[20:16]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-85. Protocol Configuration Register 14 (FR_PCR14)

29.5.2.70.16 Protocol Configuration Register 15 (FR_PCR15)

Base + 0x00BE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	listen_timeout[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-86. Protocol Configuration Register 15 (FR_PCR15)

29.5.2.70.17 Protocol Configuration Register 16 (FR_PCR16)

Base + 0x00C0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b								noise_listen_timeout[24:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-87. Protocol Configuration Register 16 (FR_PCR16)

29.5.2.70.18 Protocol Configuration Register 17 (FR_PCR17)

Base + 0x00C2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-88. Protocol Configuration Register 17 (FR_PCR17)

29.5.2.70.19 Protocol Configuration Register 18 (FR_PCR18)

Base + 0x00C4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_pattern								key_slot_id							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-89. Protocol Configuration Register 18 (FR_PCR18)

29.5.2.70.20 Protocol Configuration Register 19 (FR_PCR19)

Base + 0x00C6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_a								payload_length_static							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-90. Protocol Configuration Register 19 (FR_PCR19)

29.5.2.70.21 Protocol Configuration Register 20 (FR_PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_initial_offset_b								micro_initial_offset_a							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-91. Protocol Configuration Register 20 (FR_PCR20)

29.5.2.70.22 Protocol Configuration Register 21 (FR_PCR21)

Base + 0x00CA — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	extern_rate_correction				latest_tx											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-92. Protocol Configuration Register 21 (FR_PCR21)

29.5.2.70.23 Protocol Configuration Register 22 (FR_PCR22)

Base + 0x00CC — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*				comp_accepted_startup_range_a								micro_per_cycle[19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-93. Protocol Configuration Register 22 (FR_PCR22)

29.5.2.70.24 Protocol Configuration Register 23 (FR_PCR23)

Base + 0x00CE — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-94. Protocol Configuration Register 23 (FR_PCR23)

29.5.2.70.25 Protocol Configuration Register 24 (FR_PCR24)

Base + 0x00D0 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cluster_drift_damping				max_payload_length_dynamic								micro_per_cycle_min[19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-95. Protocol Configuration Register 24 (FR_PCR24)

29.5.2.70.26 Protocol Configuration Register 25 (FR_PCR25)

Base + 0x00D2 — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-96. Protocol Configuration Register 25 (FR_PCR25)

29.5.2.70.27 Protocol Configuration Register 26 (FR_PCR26)

Base + 0x00D4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	allow_halt_due_to_clock				comp_accepted_startup_range_b								micro_per_cycle_max [19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-97. Protocol Configuration Register 26 (FR_PCR26)

29.5.2.70.28 Protocol Configuration Register 27 (FR_PCR27)

Base + 0x00D6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	micro_per_cycle_max[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-98. Protocol Configuration Register 27 (FR_PCR27)

29.5.2.70.29 Protocol Configuration Register 28 (FR_PCR28)

Base + 0x00D8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	dynamic_slot_idle_phase				macro_after_offset_correction											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-99. Protocol Configuration Register 28 (FR_PCR28)

29.5.2.70.30 Protocol Configuration Register 29 (FR_PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_offset_correction				minislots_max											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-100. Protocol Configuration Register 29 (FR_PCR29)

29.5.2.70.31 Protocol Configuration Register 30 (FR_PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-101. Protocol Configuration Register 30 (FR_PCR30)

29.5.2.71 ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Base + 0x00F0 — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRNE_OF	LRCE_OF	DRNE_OF	DRCE_OF	LRNE_IF	LRCE_IF	DRNE_IF	DRCE_IF	0	0	0	0	LRNE_IE	LRCE_IE	DRNE_IE	DRCE_IE
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-102. ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

This register provides the means to control the ECC-related interrupt request lines and provides the corresponding interrupt flags. The interrupt flags are cleared by writing 1, which resets the corresponding report registers. For a detailed description see Section 29.6.24.2, Memory Error Reporting.

Table 29-81. FR_EEIFER field descriptions

Field	Description
	Error Overflow Flags
LRNE_OF	**LRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events occurs: a) Memory errors are detected *but not corrected* on CHI LRAM and interrupt flag LRNE_IF is already 1. b) Memory errors are detected *but not corrected* on at least two banks of CHI LRAM. 0 No such event. 1 Non-Corrected error overflow detected on CHI LRAM.
LRCE_OF	**LRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events occurs: a) Memory errors are detected *and corrected* on CHI LRAM and interrupt flag LRCE_IF is already 1. b) Memory errors are detected *and corrected* on at least two banks of CHI LRAM. 0 No such event. 1 Corrected error overflow detected on CHI LRAM. **Note:** Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_OF	**DRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events occurs: a) Memory errors are detected *but not corrected* on PE DRAM and interrupt flag DRNE_IF is already 1. b) Memory errors are detected *but not corrected* on at least two banks of the PE DRAM. 0 No Such event. 1 Non-Corrected error overflow detected on PE DRAM.
DRCE_OF	**DRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events occurs: a) Memory errors are detected *and corrected* on PE DRAM and interrupt flag DRCE_IF is already 1. b) Memory errors are detected *and corrected* on at least two banks of PE DRAM. 0 No such event. 1 Corrected error overflow detected on PE DRAM.

Table 29-81. FR_EEIFER field descriptions (continued)

Field	Description
Error Interrupt Flags	
LRNE_IF	**LRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on the CHI LRAM. 0 No such event. 1 Non-Corrected error detected on CHI LRAM.
LRCE_IF	**LRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on the CHI LRAM. 0 No such event. 1 Corrected error detected on CHI LRAM. **Note:** Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_IF	**DRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on PE DRAM. 0 No such event. 1 Non-Corrected error detected on PE DRAM.
DRCE_IF	**DRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on PE DRAM. 0 No such event. 1 Corrected error detected on PE DRAM.
Error Interrupt Enables	
LRNE_IE	**LRAM Non-Corrected Error Interrupt Enable** — This flag controls whether the LRAM Non-Corrected Error Interrupt line is asserted when the LRNE_IF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
LRCE_IE	**LRAM Corrected Error Interrupt Enable** — This flag controls whether the LRAM Corrected Error Interrupt line is asserted when the LRCE_IF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
DRNE_IE	**DRAM Non-Corrected Error Interrupt Enable** — This flag controls whether the DRAM Non-Corrected Error Interrupt line is asserted when the DRNE_IF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.
DRCE_IE	**DRAM Corrected Error Interrupt Enable** — This flag controls whether the DRAM Corrected Error Interrupt line is asserted when the DRCE_IF flag is set. 0 Disable interrupt line. 1 Enable interrupt line.

29.5.2.72 ECC Error Report and Injection Control Register (FR_EERICR)

Base + 0x00F2

Write: ERS: Anytime
ERM, EIM, EIE: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	0	0	0	0	ERS		0	0	0	ERM	0	0	EIM	EIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-103. ECC Error Report and Injection Control Register (FR_EERICR)

This register configures the error injection and error reporting and provides the selector for the content of the report registers.

Table 29-82. FR_EERICR field descriptions

Field	Description
BSY	**Register Update Busy**— This field indicates the current state of the ECC configuration update and controls the register write access condition IDL specified in Section 29.5.2.2, Register write access. 0 ECC configuration is idle. 1 ECC configuration is running.
ERS	**Error Report Select** — This field selects the content of the ECC Error reporting registers. 00 Show PE DRAM non-corrected error information. 01 Show PE DRAM corrected error information. 10 Show CHI LRAM non-corrected error information. 11 Show CHI LRAM corrected error information.
ERM	**Error Report Mode** — This bit configures the type of data written into the internal error report registers on the detection of a memory error. 0 Store data and code as delivered by ECC decoding logic. 1 Store data and code as read from the memory.
EIM	**Error Injection Mode** — This bit configures the ECC error injection mode. 0 Use FR_EEIDR[DATA] and FR_EEICR[CODE] as XOR distortion pattern for error injection. 1 Use FR_EEIDR[DATA] and FR_EEICR[CODE] as write value for error injection.
EIE	**Error Injection Enable** — This bit configures the ECC error injection on the memories. 0 Error injection disabled. 1 Error injection enabled.

29.5.2.73 ECC Error Report Address Register (FR_EERAR)

Base + 0x00F4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MID	BANK			ADDR											
W																
Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-104. ECC Error Report Address Register (FR_EERAR)

This register provides the memory identifier, bank, and address for which the memory error is reported.

Table 29-83. FR_EERAR field descriptions

Field	Description
MID	**Memory Identifier** — This flag provides the memory instance for which the memory error is reported. 0 PE DRAM 1 CHI LRAM

Table 29-83. FR_EERAR field descriptions

Field	Description
BANK	**Memory Bank** — This field provides the bank for which the memory error is reported. 111 Reset value, indicates no error found after reset. For MID = 0 000 PE DRAM [7:0] 001 PE DRAM [15:8] Others — not used For MID = 1: Refer to Table 29-84 for the assignment of the LRAM banks.
ADDR	**Memory Address** — This field provides the address of the failing memory location.

Table 29-84. LRAM bank value for MID = 1

BANK	Register		
000	FR_MBCCFR(2*n*)	FR_MBDOR(6*n*)	FR_LEETR0
001	FR_MBFIDR(2*n*)	FR_MBDOR(6*n* + 1)	FR_LEETR1
010	FR_MBIDXR(2*n*)	FR_MBDOR(6*n* + 2)	FR_LEETR2
011	FR_MBCCFR(2*n* + 1)	FR_MBDOR(6*n* + 3)	FR_LEETR3
100	FR_MBFIDR(2*n* + 1)	FR_MBDOR(6*n* + 4)	FR_LEETR4
101	FR_MBIDXR(2*n* + 1)	FR_MBDOR(6*n* + 5)	FR_LEETR5
110	Not Used	Not Used	Not Used
111			

29.5.2.74 ECC Error Report Data Register (FR_EERDR)

Base + 0x00F6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-105. ECC Error Report Data Register (FR_EERDR)

This register provides the data-related information of the reported memory read access. The assignment of the bits depends on the selected memory and memory bank as shown in Table 29-86.

Table 29-85. FR_EERDR field descriptions

Field	Description
DATA	**Data** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM = 0: ECC Data, shows data as generated by the ECC decoding logic. ERM = 1: Memory Data, shows data as read from the memory.

Table 29-86. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field

MEM	BANK	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PE DRAM	0									PE DRAM[7:0]							
PE DRAM	1									PE DRAM[15:8]							
CHI LRAM	0	FR_MBCCFR(2*n*)															
CHI LRAM	0	FR_MBDOR(6*n*)															
CHI LRAM	0	FR_LEETR0															
CHI LRAM	1						FR_MBFIDR(2*n*)[FID]										
CHI LRAM	1	FR_MBDOR(6*n* + 1)															
CHI LRAM	1	FR_LEETR1															
CHI LRAM	2										FR_MBIDXR(2*n*)[MBIDX]						
CHI LRAM	2	FR_MBDOR(6*n* + 2)															
CHI LRAM	2	FR_LEETR2															
CHI LRAM	3	FR_MBCCFR(2*n* + 1)															
CHI LRAM	3	FR_MBDOR(6*n* + 3)															
CHI LRAM	3	FR_LEETR3															
CHI LRAM	4						FR_MBFIDR(2*n* + 1)[FID]										
CHI LRAM	4	FR_MBDOR(6*n* + 4)															
CHI LRAM	4	FR_LEETR4															
CHI LRAM	5										FR_MBIDXR(2*n* + 1)[MBIDX]						
CHI LRAM	5	FR_MBDOR(6*n* + 5)															
CHI LRAM	5	FR_LEETR5															

29.5.2.75 ECC Error Report Code Register (FR_EERCR)

Base + 0x00F8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-106. ECC Error Report Code Register (FR_EERCR)

This register provides the ECC-related information of the reported memory read access.

Table 29-87. FR_EERSR field descriptions

Field	Description
CODE	**Code** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM = 0: Syndrome. Shows the ECC syndrome generated by the ECC decoding logic. The coding of the PE DRAM syndrome is shown in Section 29.6.24.2.2, PE DRAM Syndrome. The coding of the CHI LRAM syndrome is shown in Section 29.6.24.2.4, CHI LRAM Syndrome. ERM = 1: Checkbits. Shows the ECC checkbits read from the memory.

29.5.2.76 ECC Error Injection Address Register (FR_EEIAR)

Base + 0x00FA — Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MID			BANK					ADDR							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-107. ECC Error Injection Address Register (FR_EEIAR)

This register defines the memory module, bank, and address where the ECC error has to be injected.

Table 29-88. FR_EEIAR field descriptions

Field	Description
MID	**Memory Identifier** — This flag defines the memory instance for ECC error injection. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field defines the memory bank for ECC error injection. For MID = 0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] Other values are reserved. For MID = 1: Refer to Table 29-84 for the assignment of the LRAM banks.
ADDR	**Memory Address** — This flag defines the memory address for ECC error injection.

29.5.2.77 ECC Error Injection Data Register (FR_EEIDR)

Base + 0x00FC — Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-108. ECC Error Injection Data Register (FR_EEIDR)

This register defines the data distortion pattern for the error injection write. The number of valid bits depends on the selected memory and memory bank as shown in Table 29-86.

Table 29-89. FR_EEIDR field descriptions

Field	Description
DATA	**Data** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM = 0: This field defines the XOR distortion pattern for the data written into the memory. EIM = 1: This field defines the data to be written into the memory.

NOTE

The effect of the error injected depends on the LRAM content at the address accessed and on the module internal usage of the data. Refer to Section 29.6.24.3, Memory Error Response, for details.

29.5.2.78 ECC Error Injection Code Register (FR_EEICR)

Base + 0x00FE — Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-109. ECC Error Injection Code Register (FR_EEICR)

This register defines the ECC code distortion pattern for the error injection write.

Table 29-90. FR_EEICR field descriptions

Field	Description
CODE	**Code** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM = 0: This field defines the XOR distortion pattern for the ECC checkbits written into the memory. EIM = 1: This field defines the ECC checkbits written into the memory.

29.5.2.79 Message Buffer Configuration, Control, Status Registers (FR_MBCCSR*n*)

Base + 0x0800 (FR_MBCCSR0)
Base + 0x0808 (FR_MBCCSR1)
...
Base + 0x09F8 (FR_MBCCSR63)

Write: MTD: *POC:cnfig* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-110. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in Section 29.6.6, Individual message buffer functional description.

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 29-91. FR_MBCCSR*n* field descriptions

Field	Description
Message Buffer Configuration	
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer. 1 Transmit message buffer.
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates whether the transmit message buffer data is ready for transmission. 0 Message buffer data not ready for transmission. 1 Message buffer data ready for transmission.
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value of the EDS status bit. 0 No effect. 1 Message buffer enable or disable is triggered.
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit and writes 0 to the EDT bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect. 1 Message buffer lock or unlock is triggered.
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.
Message Buffer Status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame header and message buffer data field not updated. 1 Frame header and message buffer data field updated.
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 Receive message buffer contains no valid frame data / message is transmitted for the first time. 1 Receive message buffer contains valid frame data / message will be transferred again.
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event. 1 Slot status field updated or transmit message buffer just enabled.

29.5.2.80 Message Buffer Cycle Counter Filter Registers (FR_MBCCFR*n*)

Base + 0x0802 (FR_MBCCFR0) 16-bit write access required Write: *POC:config* or MB_DIS
Base + 0x080A (FR_MBCCFR1)
...
Base + 0x09FA (FR_MBCCFR63)

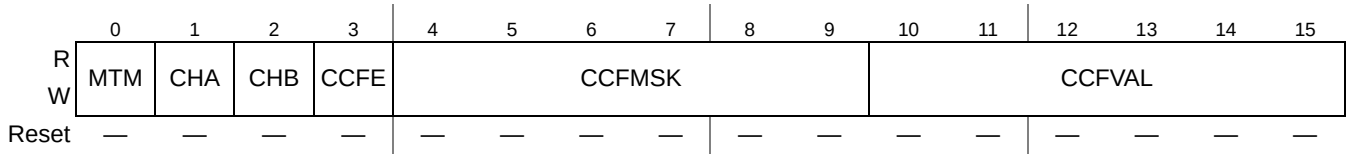


Figure 29-111. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to Section 29.6.7.1, Message Buffer Cycle Counter Filtering.

Table 29-92. FR_MBCCFR*n* field descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode. 1 State transmission mode.
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to Table 29-93.
CCFE	**Cycle Counter Filtering Enable** — This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled. 1 Cycle counter filtering enabled.
CCFMSK	**Cycle Counter Filtering Mask** — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 29-93. Channel assignment description

CHA	CHB	Transmit message buffer		Receive message buffer	
		Static segment	Dynamic segment	Static segment	Dynamic segment
1	1	Transmit on both channel A and channel B	Transmit on channel A only	Store first valid frame received on either channel A or channel B	Store first valid frame received on channel A, ignore channel B
0	1	Transmit on channel B	Transmit on channel B	Store first valid frame received on channel B	Store first valid frame received on channel B
1	0	Transmit on channel A	Transmit on channel A	Store first valid frame received on channel A	Store first valid frame received on channel A
0	0	No frame transmission	No frame transmission	No frame stored	No frame stored

NOTE

If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

29.5.2.81 Message Buffer Frame ID Registers (FR_MBFIDR*n*)

Base + 0x0804 (FR_MBFIDR0) 16-bit write access required Write: *POC:config* or MB_DIS
Base + 0x080C (FR_MBFIDR1)
...
Base + 0x09FC (FR_MBFIDR63)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—

Figure 29-112. Message Buffer Frame ID Registers (FR_MBFIDRn)

Table 29-94. FR_MBFIDR*n* field descriptions

Field	Description
FID	**Frame ID** — The semantics of this field depends on the message buffer transfer type. • *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. • *Transmit Message Buffer:* This field is used to determine the slot in which the message in this message buffer should be transmitted.

29.5.2.82 Message Buffer Index Registers (FR_MBIDXR*n*)

Base + 0x0806 (FR_MBIDXR0) 16-bit write access required Write: *POC:config* or MB_DIS
Base + 0x080E (FR_MBIDXR1)
...
Base + 0x09FE (FR_MBIDXR63)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	MBIDX						
W																
Reset	0	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—

Figure 29-113. Message Buffer Index Registers (FR_MBIDXRn)

Table 29-95. FR_MBIDXR*n* field descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The CC updates this register after frame reception or transmission. Legal Values are $0 \leq i \leq 67$. Illegal values will be detected during the message buffer search.

29.5.2.83 Message Buffer Data Field Offset Registers (FR_MBDOR*n*)

Base + 0x1000 (FR_MBDOR0) 16-bit write access required Write: *POC:config*
Base + 0x1002 (FR_MBDOR1)
...
Base + 0x1086 (FR_MBDOR67)

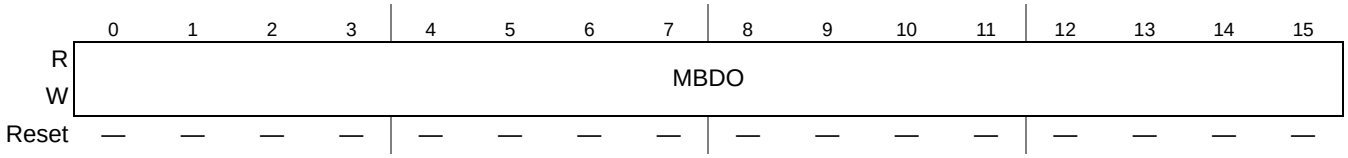


Figure 29-114. Message Buffer Data Field Offset Registers (FR_MBDORn)

Table 29-96. FR_MBDOR*n* field descriptions

Field	Description
MBDO	**Message Buffer Data Field Offset** — This field provides the data field offset belonging to a particular Message Buffer Index. For configuration constraints see Section 29.7.1.2, Configure Data Field Offsets.

29.5.2.84 LRAM ECC Error Test Registers (FR_LEETR*n*)

Base + 0x1090 (FR_LEETR0) 16-bit write access required Write: Anytime
...
Base + 0x109A(FR_LEETR5)

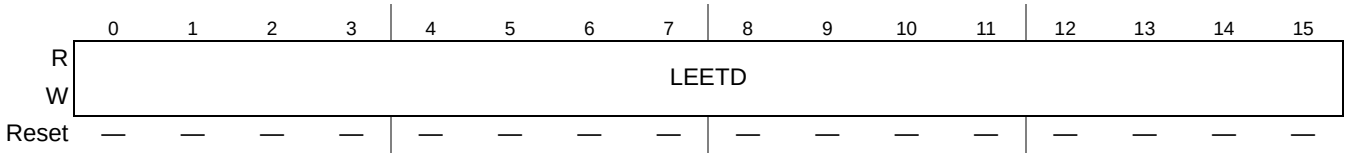


Figure 29-115. LRAM ECC Error Test Registers (FR_LEETRn)

Table 29-97. FR_LEETR*n* field descriptions

Field	Description
LEETD	**LRAM ECC Error Test Data** — This field contains the LRAM data belonging to the test register located in LRAM Bank n.

29.6 Functional description

This section provides a detailed description of the functionality implemented in the CC.

29.6.1 Message buffer concept

The CC uses a data structure called a *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data is located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in Section 29.6.3, Message buffer types. The physical message buffer is located in the FlexRay memory area and is described in Section 29.6.2, Physical message buffer.

29.6.2 Physical message buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the FlexRay memory area.The structure of a physical message buffer is depicted in Figure 29-116.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header* and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

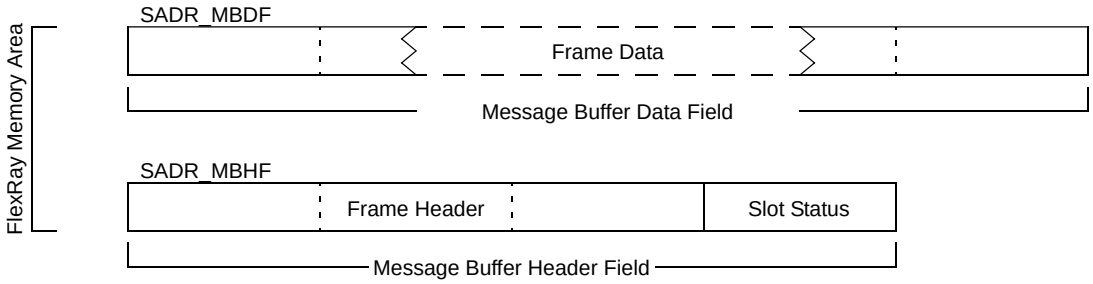


Figure 29-116. Physical message buffer structure

29.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the FlexRay memory area and occupies eight bytes. It contains the frame header and the slot status. Its structure is shown in Figure 29-116. The physical start address SADR_MBHF of the message buffer header field must be 16-bit aligned.

29.6.2.1.1 Frame Header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the usage and the content of the frame header is provided in Section 29.6.5.2.1, Frame Header Description.

29.6.2.1.2 Slot Status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in Section 29.6.5.2.2, Slot Status Description.

29.6.2.2 Message Buffer Data Field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum

length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in Section 29.6.3, Message buffer types.

29.6.3 Message buffer types

The CC provides three different types of message buffers:

- Individual message buffers
- Receive shadow buffers
- Receive FIFO buffers

For each message buffer type, the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data is described in the following sections.

29.6.3.1 Individual Message Buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The CC supports three types of individual message buffers, which are described in Section 29.6.6, Individual message buffer functional description.

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FlexRay memory area, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in Figure 29-117.

Each individual message buffer has a message buffer number *n* assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers FR_MBCCSR*n*, FR_MBCCFR*n*, FR_MBFIDR*n*, and FR_MBIDXR*n*.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the Message Buffer Index Registers (FR_MBIDXRn). The start address SADR_MBHF of the related message buffer header field in the FlexRay memory area is determined according to Equation 29-1.

$$\text{SADR\_MBHF} = (\text{FR\_MBIDXR}n[\text{MBIDX}] \times 8) + \text{SMBA} \qquad \textit{Eqn. 29-1}$$

The data field belonging to a particular physical message buffer is characterized by the data field offset. For each physical message buffer with MBIDXi the FR_MBDORi contains the offset of the corresponding message buffer data field with respect to the CC FlexRay memory area base address as provided by SMBA field in the System Memory Base Address Register (FR_SYMBADR).

The data field offset is used to determine the start address SADR_MBDF of the corresponding message buffer data field in the FlexRay memory area according to Equation 29-2.

$$\text{SADR\_MBDF} = [\text{Data Field Offset}] + \text{SMBA} \qquad \textit{Eqn. 29-2}$$

Instances of FR_MBDOR*n* are stored in the module internal memory LRAM. Refer to Section 29.7.2.3, CHI LRAM Initialization, for the setup of the data field offset values.

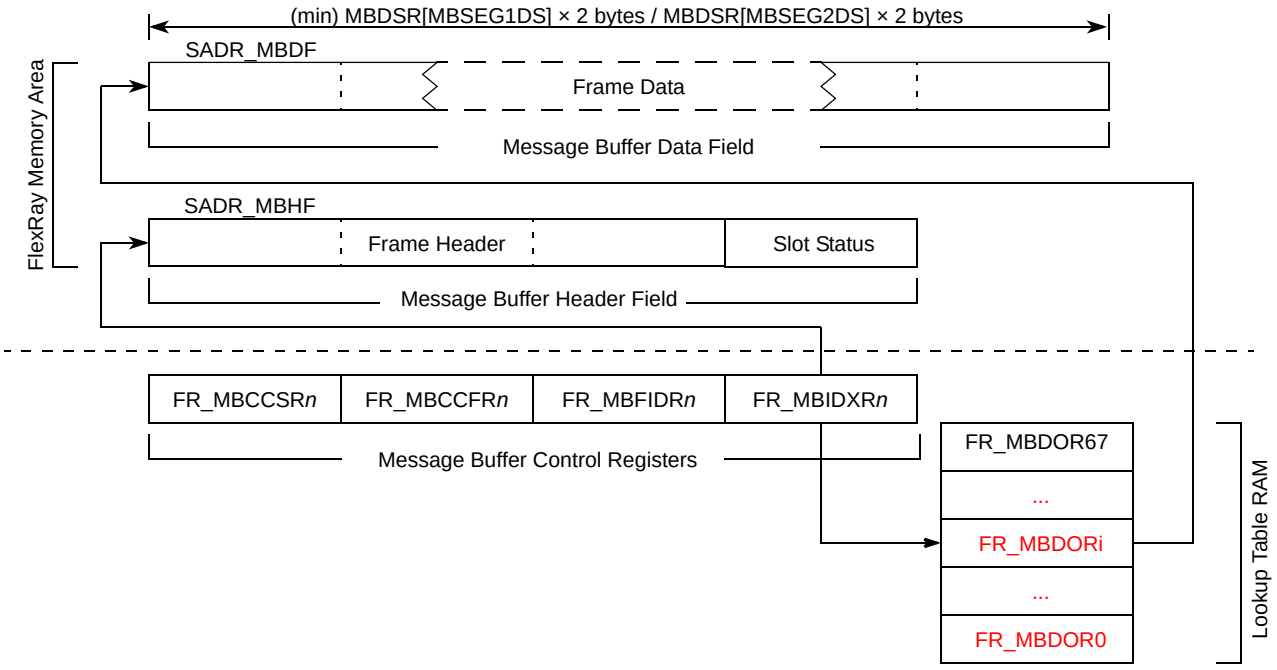


Figure 29-117. Individual message buffer structure

29.6.3.1.1 Individual Message Buffer Segments

The set of the individual message buffers can be split up into two message buffer segments using the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR). All individual message buffers with a message buffer number $n \leq$ FR_MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number $n >$ FR_MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- All physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length.
- The minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 × FR_MBDSR[MBSEG1DS] bytes.
- The minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 × FR_MBDSR[MBSEG2DS] bytes.

29.6.3.2 Receive Shadow Buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The CC provides four receive shadow buffers: one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FlexRay memory area and the receive shadow buffer control registers located in dedicated registers. The structure

of a receive shadow buffer is shown in Figure 29-118. The four internal shadow buffer control registers can be accessed by the Receive Shadow Buffer Index Register (FR_RSBIR).

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the Receive Shadow Buffer Index Register (FR_RSBIR). The start address SADR_MBHF of the related message buffer header field in the FlexRay memory area is determined according to Equation 29-3.

$$SADR\_MBHF = (FR\_RSBIR[RSBIDX] \times 8) + SMBA \qquad \textit{Eqn. 29-3}$$

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in Receive Shadow Buffer Index Register (FR_RSBIR).

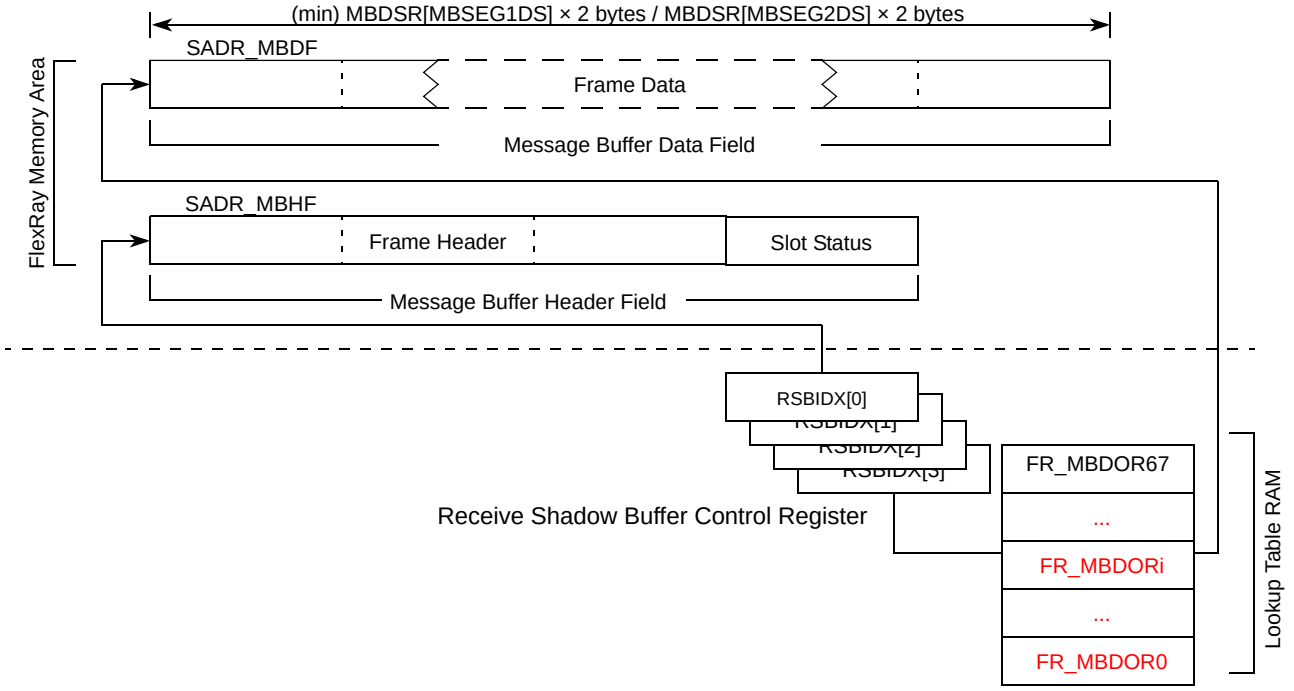


Figure 29-118. Receive shadow buffer structure

29.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The CC provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FlexRay memory area and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in Figure 29-119.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the Receive FIFO Start Index Register (FR_RFSIR), the Receive FIFO Depth and Size Register (RFDSR), and the Receive FIFO A Read Index Register (FR_RFARIR) / Receive FIFO B Read Index Register (FR_RFBRIR).

The system memory base address SMBA valid for the receive FIFOs is defined by the system memory base address register selected by the FIFO address mode bit FR_MCR[FAM]; refer to Section 29.5.2.4, Module Configuration Register (FR_MCR).

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO is determined according to Equation 29-4.

$$SADR\_MBHF[1] = (8 \times FR\_RFSIR[SIDX]) + SMBA \qquad \textit{Eqn. 29-4}$$

The start byte address SADR_MBHF[*n*] of the last message buffer header field that belongs to the receive FIFO in the FlexRay memory area is determined according to Equation 29-5.

$$SADR\_MBHF[n] = (8 \times (FR\_RFSIR[SIDX] + RFDSR[FIFO\_DEPTH])) + SMBA \qquad \textit{Eqn. 29-5}$$

The required information to access the current entry of the FIFO is given in the following registers:

- The registers Receive FIFO A Read Index Register (FR_RFARIR) and Receive FIFO B Read Index Register (FR_RFBRIR) provide the index of the physical message buffer belonging to the current entry.

  The data field offset belonging to the current FIFO entry RF_DFO[X] must be calculated using the current read index *i* according to the following formula:

$$RF\_DFO[X] = FR\_RFSDOR[X] + (FR\_RFDSR[X][ENTRY\_SIZE] \times 2) \times i - FR\_RFSIDX[X]) \qquad \textit{Eqn. 29-6}$$

NOTE

The current read index loops up starting at the number given in the FR_RD[A/B]RDIDX register for the required number of entries.

Refer to Section 29.6.9.8, FIFO Update for details about updating the FIFO read pointer.

All message buffer header fields assigned to a receive FIFO are within a contiguous region defined by FR_RFSIR[SIDX] and RFDSR[FIFO_DEPTH].

The data sections of all FIFO entries within one receive FIFO are of the same length defined by RFDSR[FIFO_SIZE].

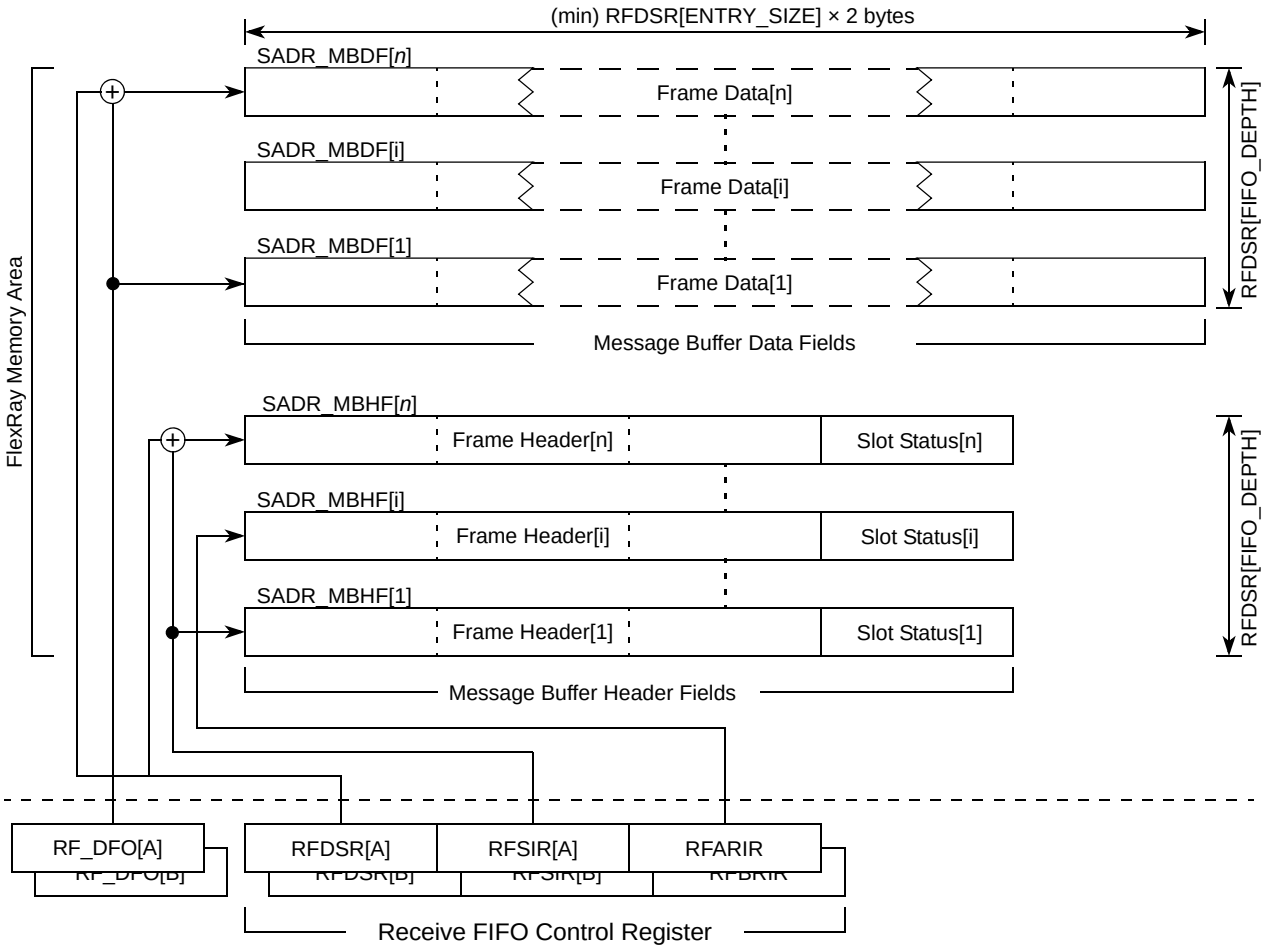


Figure 29-119. Receive FIFO structure

NOTE

The actual values of the data field offsets RF_DFO[A/B] need to be calculated according to Equation 29-6. They are not stored in a register.

29.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

29.6.3.4.1 Individual Message Buffer Configuration Data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

Common Configuration Data

The set of common configuration data for individual message buffers is located in the following registers.

- Message Buffer Data Size Register (FR_MBDSR)
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see Section 29.6.3.1.1, Individual Message Buffer Segments.

Specific Configuration Data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)
  The MTD bit configures the message buffer type.
- Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- Message Buffer Frame ID Registers (FR_MBFIDRn)
  For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- Message Buffer Index Registers (FR_MBIDXRn)
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

29.6.3.5 Individual Message Buffer Control Data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn).

29.6.3.6 Receive Shadow Buffer Configuration Data

Before frame reception by the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data is provided by the Receive Shadow Buffer Index Register (FR_RSBIR). For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full CC control.

29.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

29.6.3.7.1 Receive FIFO Configuration Data

The CC provides two functional independent receive FIFOs, one per channel. The FIFOs have a common subset of configuration data:

- Receive FIFO Periodic Timer Register (FR_RFPTR)

Each FIFO has its own set of configuration data. The configuration data is located in the following registers:

- Receive FIFO Watermark and Selection Register (FR_RFWMSR)
- Receive FIFO Start Index Register (FR_RFSIR)
- Receive FIFO Start Data Offset Register (FR_RFSDOR)
- Receive FIFO Depth and Size Register (RFDSR)
- Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
- Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
- Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
- Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
- Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

29.6.3.7.2 Receive FIFO Control Data

The application can access the FIFOs at any time using the control bits in the following registers:

- Global Interrupt Flag and Enable Register (FR_GIFER)
- Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

29.6.3.7.3 Receive FIFO Status Data

The current status of the receive FIFO is provided in the following register:

- Global Interrupt Flag and Enable Register (FR_GIFER)
- Receive FIFO A Read Index Register (FR_RFARIR)
- Receive FIFO B Read Index Register (FR_RFBRIR)
- Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

29.6.4 Flexray Memory Area Layout

The CC supports a wide range of possible layouts for the FlexRay memory area. Two basic layout modes can be selected by the FIFO address mode bit FR_MCR[FAM].

29.6.4.1 Flexray Memory Area Layout (FR_MCR[FAM] = 0)

Figure 29-120 shows an example layout for the FIFO address mode FR_MCR[FAM]=0. In this mode, the following set of rules applies to the layout of the FlexRay memory area:

- The FlexRay memory area is one contiguous region.
- The FlexRay memory area size is maximum 64 KB.
- The FlexRay memory area starts at a 16 byte boundary.

The FlexRay memory area contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*.

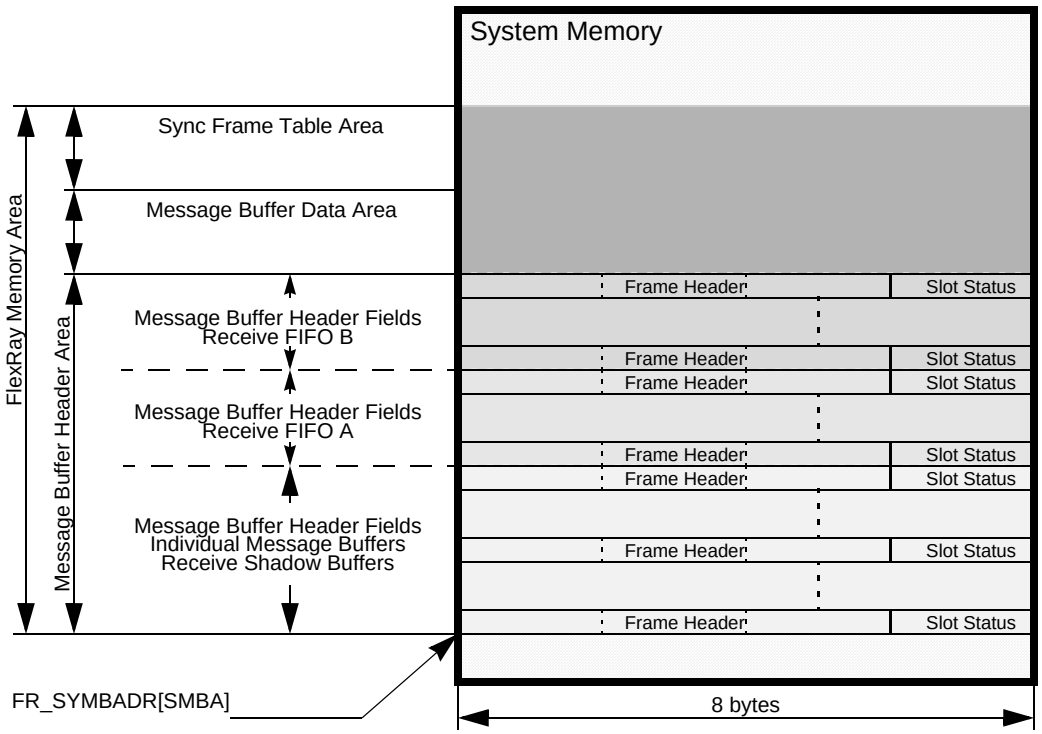


Figure 29-120. Example of FlexRay memory area layout (FR_MCR[FAM] = 0)

29.6.4.2 FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

Figure 29-121 shows an example layout for the FIFO address mode FR_MCR[FAM]=1. The following set of rules applies to the layout of the FlexRay memory area:

- The FlexRay memory area consists of two contiguous regions.
- The size of each region is maximum 64 KB.
- Each region starts at a 16-byte boundary.

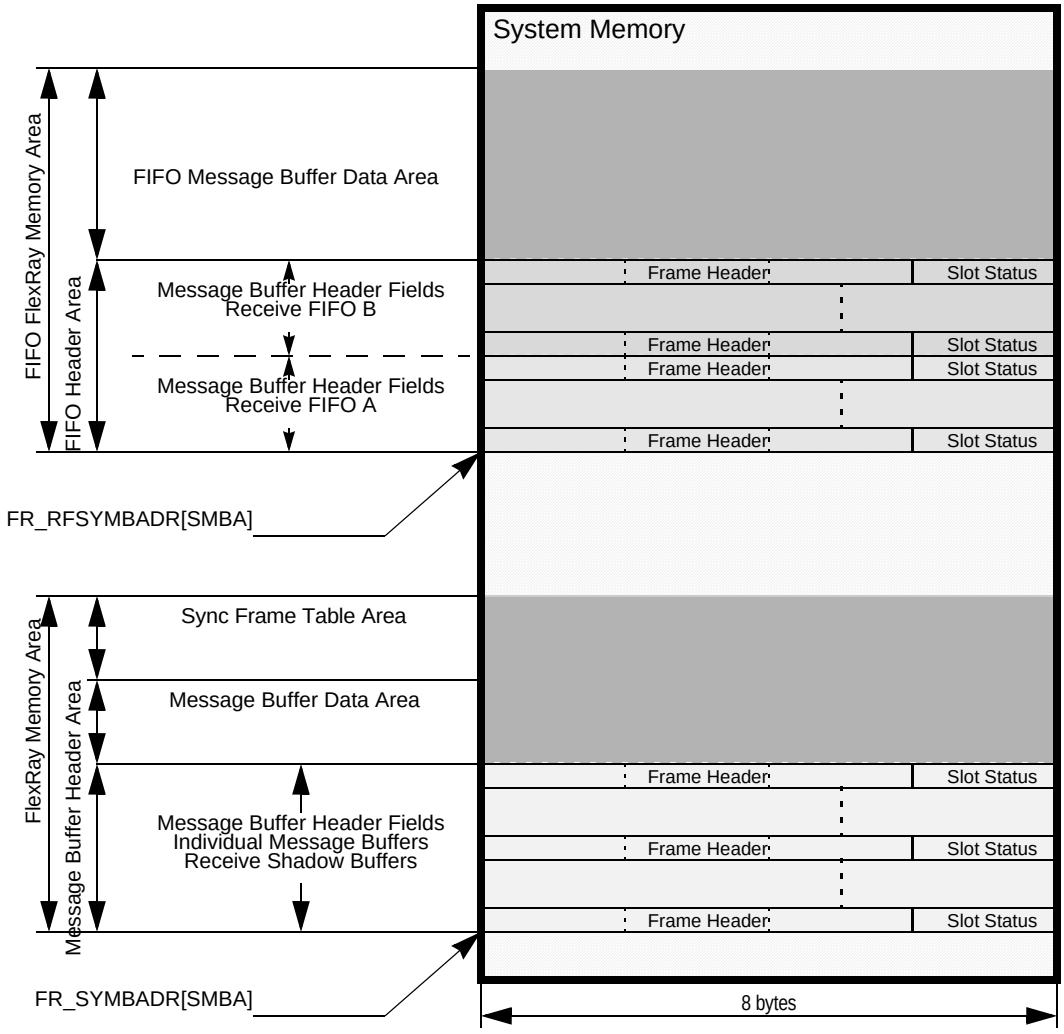


Figure 29-121. Example of FlexRay memory area layout (FR_MCR[FAM] = 1)

29.6.4.3 Message Buffer Header Area (FR_MCR[FAM] = 0)

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start byte address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 29-7.

$$SADR\_MBHF = (i \times 8) + FR\_SYMBADR[SMBA];\ (0 \le i \le 67) \qquad \textit{Eqn. 29-7}$$

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 29-8.

$$SADR\_MBHF = (i \times 8) + FR\_SYMBADR[SMBA];\ (0 \le i \le 1023) \qquad \textit{Eqn. 29-8}$$

3. The message buffer header fields for each FIFO must be a contiguous area.

29.6.4.4 Message Buffer Header Area (FR_MCR[FAM] = 1)

The message buffer header area contains all message buffer header fields of the physical message buffers for the individual message buffers and receive shadow buffers. The following rules apply to the message buffer header fields for the two types of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 29-9.

$$SADR\_MBHF = (i \times 8) + FR\_SYMBADR[SMBA];\ (0 \le i \le 67) \qquad \textit{Eqn. 29-9}$$

29.6.4.5 FIFO Message Buffer Header Area (FR_MCR[FAM] = 1)

The FIFO message buffer header area contains all message buffer header fields of the physical message buffers for the FIFO. The following rules apply to the FIFO message buffer header fields.

1. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 29-10.

$$SADR\_MBHF = (i \times 8) + FR\_RFSYMBADR[SMBA];\ (0 \le i \le 1023) \qquad \textit{Eqn. 29-10}$$

2. The message buffer header fields for each FIFO have to be a contiguous area.

29.6.4.6 Message Buffer Data Area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

29.6.4.7 Sync Frame Table Area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to Section 29.6.12, Sync Frame ID and Sync Frame Deviation Tables, for the description of the sync frame table area.

29.6.5 Physical Message Buffer Description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field, and the message buffer data field.

29.6.5.1 Message Buffer Protection and Data Consistency

The physical message buffers are located in the FlexRay memory area. The CC provides no means to protect the FlexRay memory area from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

29.6.5.2 Message Buffer Header field description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in Section 29.6.2.1, Message Buffer Header Field. Each message buffer header field consists of two sections: the frame header section and the slot status section.

29.6.5.2.1 Frame Header Description

Frame Header Content

The semantics and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received, regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in Figure 29-122. A detailed description is given in Table 29-99.

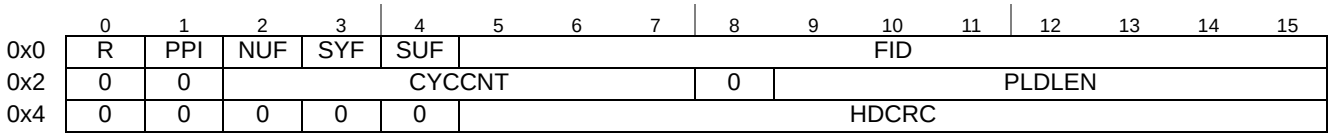


Figure 29-122. Frame header structure (receive message buffer and Receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in Figure 29-123. A detailed description is given in Table 29-100. The checks that will be performed are described in Section , Frame Header Checks.

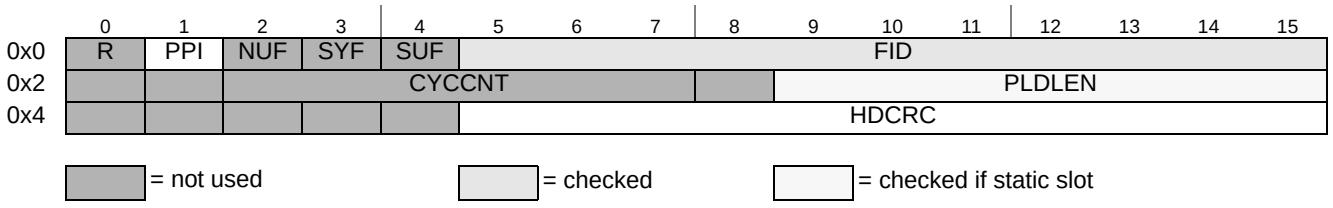


Figure 29-123. Frame header structure (Transmit message buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in Figure 29-124.

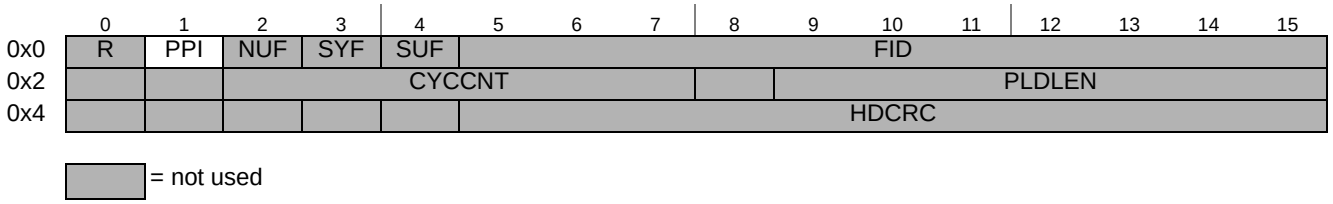


Figure 29-124. Frame header structure (Transmit message buffer for key slot)

Frame Header Access

The frame header is located in the FlexRay memory area. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in Table 29-98. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 29-98. Frame header write access constraints (transmit message buffer)

Field	Static segment	Dynamic segment
FID	*POC:config* or MB_DIS	
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or	
	—	MB_LCK

Frame Header Checks

As shown in Figure 29-123 and Figure 29-124, not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding Message Buffer Frame ID Registers (FR_MBFIDRn). If the CC detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the CHI Error Flag Register (FR_CHIERFR). The value of the FID field will be ignored and replaced by the value provided in the Message Buffer Frame ID Registers (FR_MBFIDRn).

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the Protocol Configuration Register 19 (FR_PCR19). If this is not fulfilled, the static payload length error flag SPL_EF in the CHI Error Flag Register (FR_CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the Protocol Configuration Register 24 (FR_PCR24). If this is not fulfilled, the dynamic payload length error flag DPL_EF in the CHI Error Flag Register (FR_CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 29-99. Frame header field descriptions (receive message buffer and receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer.
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 29-100. Frame header field descriptions (transmit message buffer)

Field	Description
R	**Reserved Bit** — This bit is not used. The value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used. The value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SYF	**Sync Frame Indicator** — This bit is not used. The value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
SUF	**Startup Frame Indicator** — This bit is not used. The value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*
FID	**Frame ID** — This field is checked as described in Frame Header Checks.
CYCCNT	**Cycle Count** — This field is not used. The value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.

Table 29-100. Frame header field descriptions (transmit message buffer) (continued)

Field	Description
PLDLEN	**Payload Length** — This field is checked and used as described in Frame Header Checks.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

29.6.5.2.2 Slot Status Description

The slot status is a read-only structure for the application and a write-only structure for the CC. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

Receive Message Buffer and Receive FIFO Slot Status Description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by Table 29-101.

Table 29-101. Receive message buffer slot status content

Receive message buffer type	Slot status content
Individual receive message buffer assigned to both channels FR_MBCCFR*n*[CHA] = 1 and FR_MBCCFR*n*[CHB] = 1	See Figure 29-125
Individual receive bessage Buffer assigned to channel A FR_MBCCFR*n*[CHA] = 1 and FR_MBCCFR*n*[CHB] = 0	See Figure 29-126
Individual receive message Buffer assigned to channel B FR_MBCCFR*n*[CHA] = 0 and FR_MBCCFR*n*[CHB] = 1	See Figure 29-127
Receive FIFO Channel A message buffer	See Figure 29-126
Receive FIFO Channel B message buffer	See Figure 29-127

The meaning of the bits in the slot status structure is explained in Table 29-102.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-125. Receive message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-126. Receive message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-127. Receive message buffer slot status structure (ChB)

Table 29-102. Receive message buffer slot status field description

Field	Description
	Common message buffer status bits
VFB	**Valid Frame on Channel B** — Protocol-related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — Protocol-related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — Protocol-related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — Protocol-related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 First valid frame received on channel A, or no valid frame received at all. 1 First valid frame received on channel B.
VFA	**Valid Frame on Channel A** — Protocol-related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — Protocol-related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1

Table 29-102. Receive message buffer slot status field description (continued)

Field	Description
CEA	**Content Error on Channel A** — Protocol-related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — Protocol-related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1

Transmit Message Buffer Slot Status Description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by Table 29-103.

Table 29-103. Transmit message buffer slot status content

Transmit message buffer type	Slot status content
Individual transmit message buffer assigned to both channels FR_MBCCFR*n*[CHA] = 1 and FR_MBCCFR*n*[CHB] = 1	see Figure 29-128
Individual transmit message buffer assigned to channel A FR_MBCCFR*n*[CHA] = 1 and FR_MBCCFR*n*[CHB] = 0	see Figure 29-129
Individual transmit message buffer assigned to channel B FR_MBCCFR*n*[CHA] = 0 and FR_MBCCFR*n*[CHB] = 1	see Figure 29-130

The meaning of the bits in the slot status structure is described in Table 29-102.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-128. Transmit message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-129. Transmit message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 29-130. Transmit message buffer slot status structure (ChB)

Table 29-104. Transmit message buffer slot status structure field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — Protocol-related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — Protocol-related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — Protocol-related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — Protocol-related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — Protocol-related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — Protocol-related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — Protocol-related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — Protocol-related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — Protocol-related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — Protocol-related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — Protocol-related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — Protocol-related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

29.6.5.3 Message Buffer Data field description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in Table 29-105. The structure of the message buffer data field is given in Figure 29-131.

Table 29-105. Message buffer data field minimum length

Physical message buffer assigned to	Minimum length defined by
Individual Message Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive FIFO for channel A	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 0)
Receive FIFO for channel B	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 1)

NOTE

The CC will not access any locations outside the message buffer data field boundaries given by Table 29-105.

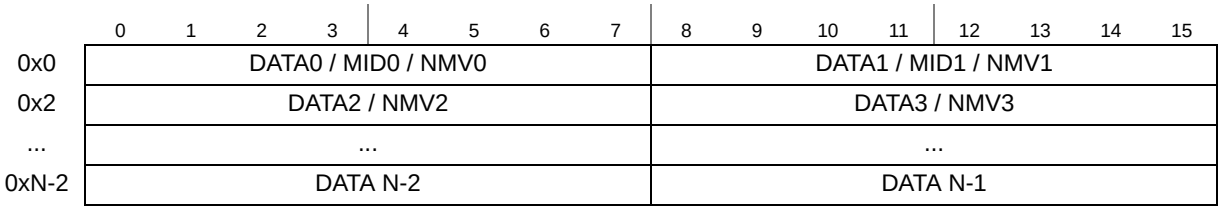


Figure 29-131. Message buffer data field structure

The message buffer data field is located in the FlexRay memory area; thus, the CC has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

29.6.5.3.1 Message Buffer Data Field Read Access

For transmit message buffers, the CC will not modify the content of the message buffer data field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the Message Buffer Index Registers (FR_MBIDXRn). While the message buffer is locked, the CC will not update the message buffer data field.

For receive FIFOs, the application can read the message buffer indicated by the Receive FIFO A Read Index Register (FR_RFARIR) or the Receive FIFO B Read Index Register (FR_RFBRIR) when the related fill levels in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) indicate a non-empty FIFO.

29.6.5.3.2 Message Buffer Data Field Write Access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in Table 29-106.

Table 29-106. Frame data write access constraints

Field	CC/MB state
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK

Table 29-107. Frame data field descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	**Message Identifier** — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	**Network Management Vector** — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. **Note:** The MID and NMV bytes replace the corresponding DATA bytes.

29.6.6 Individual message buffer functional description

The CC provides these basic types of individual message buffers:

- Transmit message buffers
- Receive message buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in Section 29.6.3.4.1, Individual Message Buffer Configuration Data.

29.6.6.1 Individual Message Buffer Configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the FlexRay memory area. The second step is the programming of the message buffer configuration registers, which is described in this section.

29.6.6.1.1 Common Configuration Data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the Message Buffer Data Size Register (FR_MBDSR).

Depending on the current receive functionality of the CC, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the Receive Shadow Buffer Index Register (FR_RSBIR).

29.6.6.1.2 Specific Configuration Data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- The protocol is in the *POC:config* state, or
- The message buffer is disabled (FR_MBCCSR*n*[EDS] = 0)

The individual message buffer type is defined by the MTD and MBT bits in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) as given in Table 29-108.

Table 29-108. Individual message buffer types

FR_MBCCSR*n*		Individual message buffer description
MTD	MBT	
0	0	Receive Message Buffer
0	1	Reserved
1	0	Transmit Message Buffer
1	1	Reserved

The message buffer specific configuration data are

1. MTD bits in Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn).
2. All fields and bits in Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn).
3. All fields and bits in Message Buffer Frame ID Registers (FR_MBFIDRn).
4. All fields and bits in Message Buffer Index Registers (FR_MBIDXRn).

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions Section 29.6.6.2, Transmit Message Buffers, and Section 29.6.6.3, Receive Message Buffers.

29.6.6.2 Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A transmit message buffer is used by the application to provide message data to the CC that will be transmitted over the FlexRay Bus. The CC uses the transmit message buffers to provide information about the transmission process and status information about the slot in which the message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a transmit message buffer by the following settings:

- FR_MBCCSR*n*[MBT] = 0 (single-buffered message buffer)
- FR_MBCCSR*n*[MTD] = 1 (transmit message buffer)

29.6.6.2.1 Access Regions

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the access to the data, control, and status bits of a message buffer. The access regions for transmit message buffers are depicted in Figure 29-132. A description of the regions is given in Table 29-109. If a region is active as indicated in Table 29-110, the access scheme given for that region applies to the message buffer.

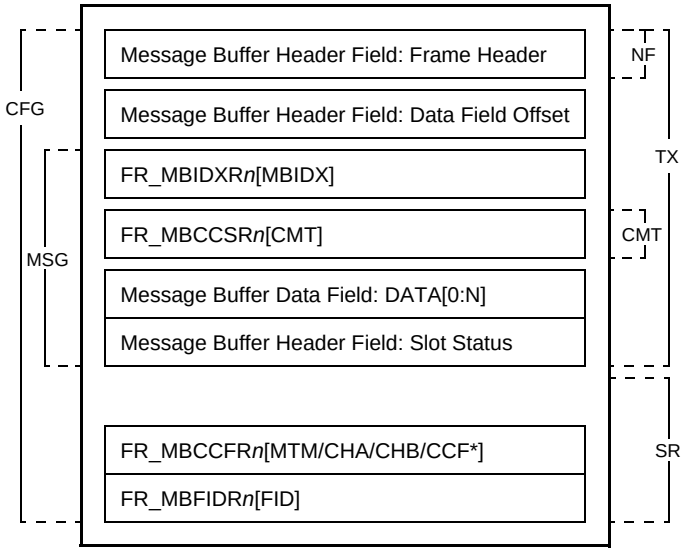


Figure 29-132. Transmit message buffer access regions

Table 29-109. Transmit message buffer access regions description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Data and Slot Status Access
NF	—	read-only	Message Header Access for Null Frame Transmission
TX	—	read/write	Message Transmission and Slot Status Update
CM	—	read-only	Message Buffer Validation

Table 29-109. Transmit message buffer access regions description (continued)

Region	Access from		Region used for
	Application	Module	
SR	—	read-only	Message Buffer Search

The trigger bits FR_MBCCSR*n*[EDT] and FR_MBCCSR*n*[LCKT], and the interrupt enable bit FR_MBCCSR*n*[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSR*n*[EDS] and FR_MBCCSR*n*[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSR*n*[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC clear access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in Figure 29-133. A description of the states is given in Table 29-110, which also provides the access scheme for the access regions.

The status bits FR_MBCCSR*n*[EDS] and FR_MBCCSR*n*[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

29.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

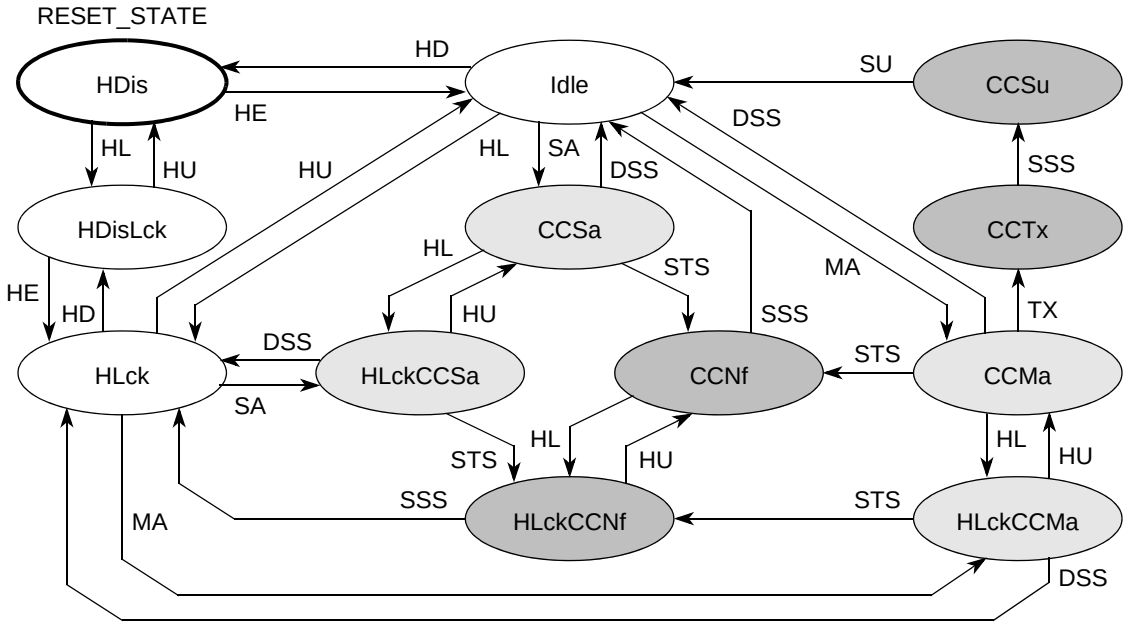


Figure 29-133. Transmit message buffer states

Table 29-110. Transmit message buffer state description

State	FR_MBCCSR*n*		Access region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	CM, SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	**Locked** — Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	—	—	**Slot Assigned** — Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	—	**Locked and Slot Assigned** — Applications access to data, control, and status. Message buffer assigned to next static slot
CCNf	1	0	—	NF	**Null Frame Transmission** — Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	**Locked and Null Frame Transmission** — Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	—	CM	**Message Available** — Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	—	**Locked and Message Available** — Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	—	TX	**Message Transmission** — Message buffer data transmit. Payload data from buffer transmitted.
CCSu	1	0	—	TX	**Status Update** — Message buffer status update. Update of status flags, the slot status field, and the header index.

29.6.6.2.3 Message Buffer Transitions

Application Transitions

The application transitions can be triggered by the application using the commands described in Table 29-111. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands are issued by writing 1 to the trigger bit FR_MBCCSR*n*[EDT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSR*n*[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition

has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node and configured and enabled message buffers in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit.

To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. This should enable the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands are issued by writing 1 to the trigger bit FR_MBCCSR*n*[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSR*n*[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (FR_CHIERFR) is set.

Table 29-111. Transmit message buffer application transitions

Transition	Command	Condition	Description
HE	FR_MBCCSR*n*[EDT]:= 1	FR_MBCCSR*n*[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSR*n*[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSR*n*[LCKT]:= 1	FR_MBCCSR*n*[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSR*n*[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the CC are described in Table 29-112. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 29-112. Transmit message buffer module transitions

Transition	Condition	Description
SA	Slot match and static slot	Slot Assigned — Message buffer is assigned to next static slot.
MA	Slot match and CycleCounter match	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
TX	Slot start and FR_MBCCSR*n*[CMT] = 1	Transmission Slot Start — Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	Status updated	Status Updated — Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	Static slot start	Static Slot Start — Start of static slot.
DSS	Dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start — Start of dynamic slot or symbol window or NIT.
SSS	Slot start or symbol window start or NIT start	Slot or Segment Start — Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of Table 29-113, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of Table 29-113.

Table 29-113. Transmit message buffer transition priorities

State	Priorities	Description
Module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
Module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

29.6.6.2.4 Transmit Message Setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). The physical access to the message buffer data field is described in Section 29.6.3.1, Individual Message Buffers.

As indicated by Table 29-110, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in Table 29-111. The state change is indicated through the FR_MBCCSR*n*[EDS] and FR_MBCCSR*n*[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the FR_MBCCSR*n*[DVAL] flag is negated.

29.6.6.2.5 Message Transmission

As a result of the message buffer search described in Section 29.6.7, Individual Message Buffer Search, the CC triggers the message available transition MA for up to two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The CC transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. The message buffer is in the message available state CCMa.
2. The message data is still valid (FR_MBCCSR*n*[CMT] = 1).

In this case, the CC triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in Figure 29-134. In this example, the message buffer with message buffer number *n* is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data is valid (FR_MBCCSR*n*[CMT] = 1).

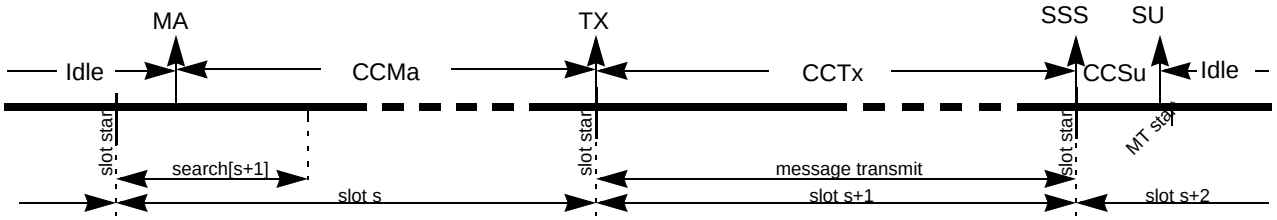


Figure 29-134. Message transmission timing

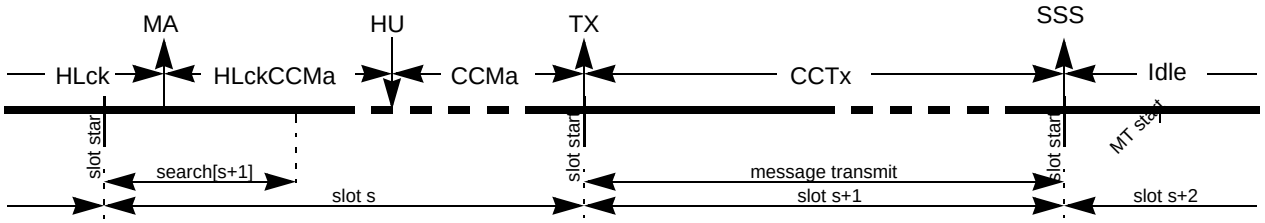


Figure 29-135. Message transmission from HLck state with unlock

The amount of message data read from the FlexRay memory area and transferred to the FlexRay bus is determined by these three items:

1. The message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).
2. The message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (FR_MBDSR).
3. The value of the PLDLEN field in the message buffer header field, as described in Section 29.6.5.2.1, Frame Header Description.

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 × MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN ≤ MBSEG1DS, the CC reads and transfers 2 × PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

29.6.6.2.6 Null Frame Transmission

A static slot with slot number S is assigned to the CC for channel A, if at least one transmit message buffer is configured with the FR_MBFIDR*n*[FID] set to S and FR_MBCCFR*n*[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A if this slot is assigned to the CC for channel A, and all transmit message buffers with FR_MBFIDR*n*[FID] = s and FR_MBCCFR*n*[CHA] = 1 are either not committed, i.e., FR_MBCCSR*n*[CMT] = 0, or locked by the application (FR_MBCCSR*n*[LCKS] = 1), or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in Section 29.6.7, Individual Message Buffer Search, the CC triggers the slot assigned transition SA for up to two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in Figure 29-136.

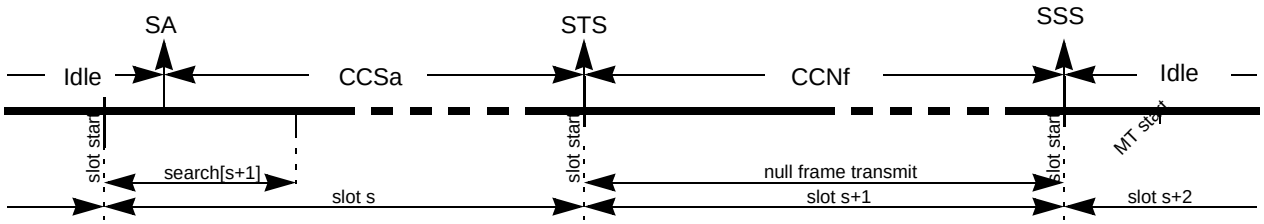


Figure 29-136. Null frame transmission from idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in Figure 29-137.

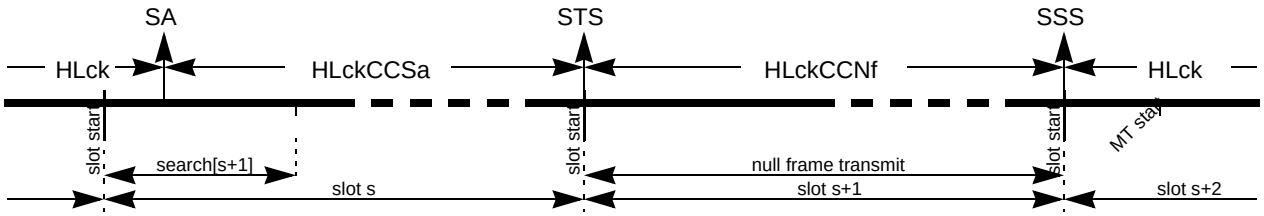


Figure 29-137. Null frame transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 29-138.

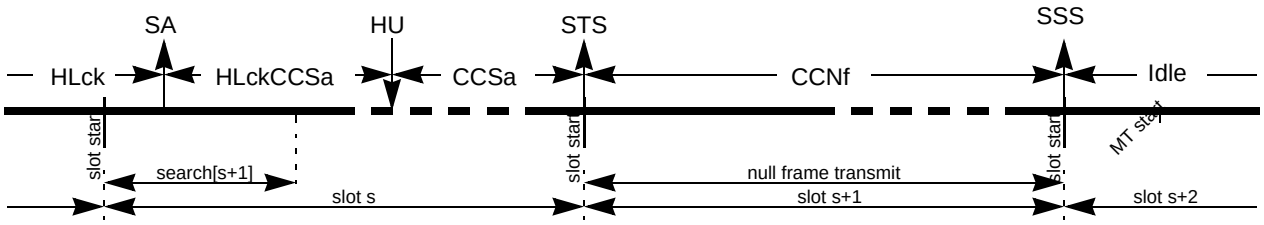


Figure 29-138. Null frame transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 29-139.

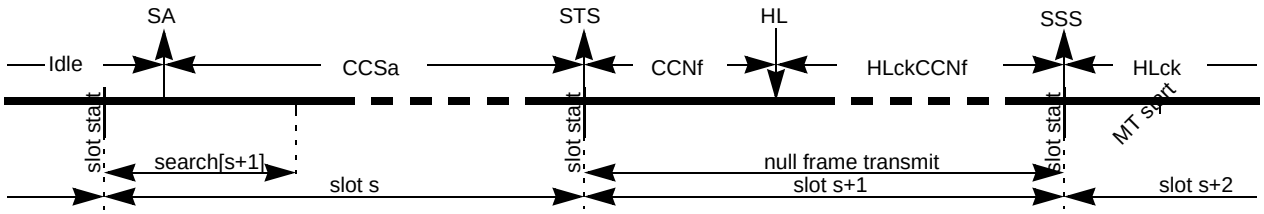


Figure 29-139. Null frame transmission from idle state with locking

29.6.6.2.7 Message Buffer Status Update

After the end of each slot, the PE generates the slot status vector. Depending on the status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

Message Buffer Status Update after Complete Message Transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were sent to the FlexRay bus. In this case, the CC updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the CC sets the message buffer interrupt flag FR_MBCCSR*n*[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag FR_MBCCFR*n*[MTM], the CC changes the commit flag FR_MBCCSR*n*[CMT] and the valid flag FR_MBCCSR*n*[DVAL]. If the FR_MBCCFR*n*[MTM] flag is negated, the message buffer is in the *event transmission mode*. In this case, each committed message is transmitted only once. The commit flag FR_MBCCSR*n*[CMT] is cleared with the SU transition. If the FR_MBCCFR*n*[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The CC will not clear the FR_MBCCSR*n*[CMT] flag at the end of transmission and will set the valid flag FR_MBCCSR*n*[DVAL] to indicate that the message will be transmitted again.

Message Buffer Status Update after Incomplete Message Transmission

The term “incomplete message transmission” refers to the fact that not all payload data that should be transmitted were sent to the FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

- The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
- The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those errors occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the Protocol Interrupt Flag Register 1 (FR_PIFR1).

In either of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Message Buffer Status Update after Null Frame Transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

29.6.6.3 Receive Message Buffers

The section provides a detailed description of the functionality of the receive message buffers. If receive message buffers are used it is necessary to configure the related receive shadow buffer as described in Section 29.6.3.2, Receive Shadow Buffers.

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The CC uses the receive message buffer to provide the following data to the application:

- Message data received
- Information about the reception process
- Status information about the slot in which the message was received

An individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings:

- FR_MBCCSR*n*[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in Figure 29-140. A description of the regions is given in Table 29-114. If a region is active as indicated in Table 29-115, the access scheme given for that region applies to the message buffer.

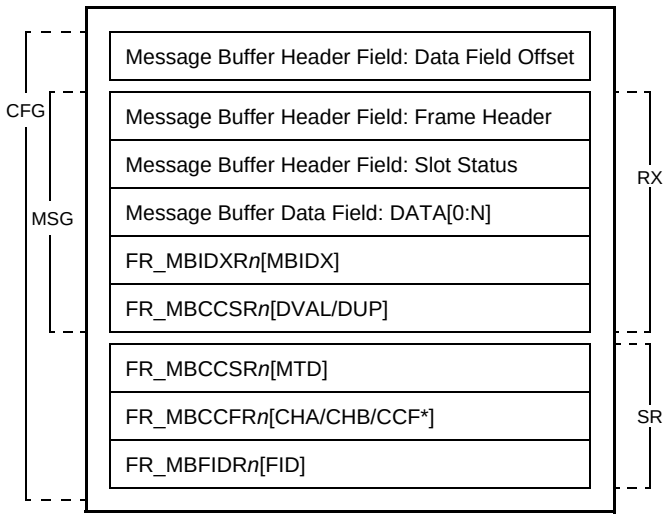


Figure 29-140. Receive message buffer access regions

Table 29-114. Receive message buffer access region description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	—	Message Data, Header, and Status Access
RX	—	write-only	Message Reception and Status Update
SR	—	read-only	Message Buffer Search Data

The trigger bits FR_MBCCSR*n*[EDT] and FR_MBCCSR*n*[LCKT] and the interrupt enable bit FR_MBCCSR*n*[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSR*n*[EDS] and FR_MBCCSR*n*[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSR*n*[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC set access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in Figure 29-141. A description of the message buffer states is given in Table 29-110, which also provides the access scheme for the access regions.

The status bits FR_MBCCSR*n*[EDS] and FR_MBCCSR*n*[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

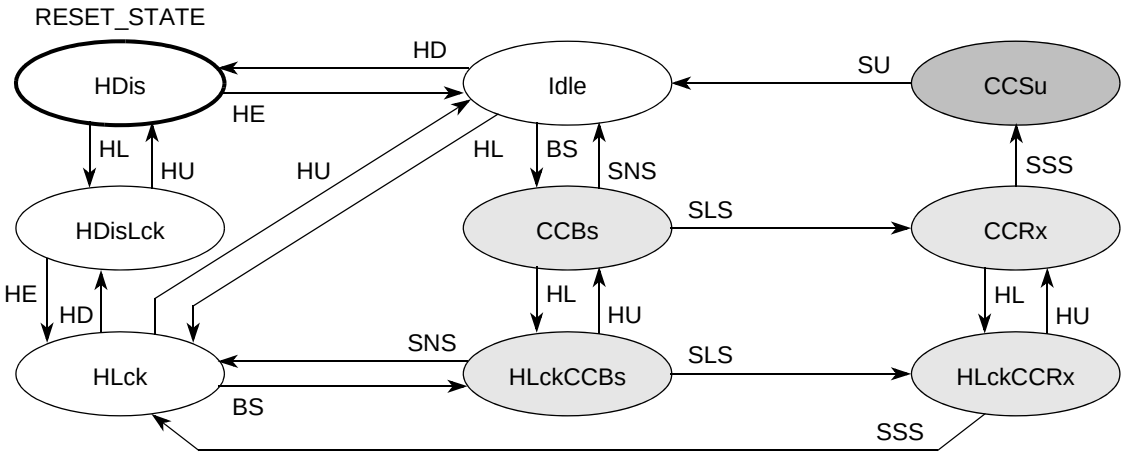


Figure 29-141. Receive message buffer states

Table 29-115. Receive Message buffer states and access

State	FR_MBCCSR*n*		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	—	**Locked** — Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	—	—	**Buffer Subscribed** — Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	—	**Locked and Buffer Subscribed** — Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	—	—	**Message Receive** — Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	—	**Locked and Message Receive** — Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	—	RX	**Status Update** — Message buffer status update. Update of status flags, the slot status field, and the header index.

29.6.6.3.1 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application use the commands described in Table 29-116. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands are issued by writing 1 to the trigger bit FR_MBCCSR*n*[EDT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSR*n*[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node and configured and enabled message buffers in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit.

To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. This should enable the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands are issued by writing 1 to the trigger bit FR_MBCCSR*n*[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSR*n*[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and the message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (FR_CHIERFR) is set.

Table 29-116. Receive message buffer application transitions

Transition	Host command	Condition	Description
HE	FR_MBCCSR*n*[EDT]:= 1	FR_MBCCSR*n*[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSR*n*[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSR*n*[LCKT]:= 1	FR_MBCCSR*n*[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSR*n*[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the CC are described in Table 29-117. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 29-117. Receive message buffer module transitions

Transition	Condition	Description
BS	Slot match and CycleCounter match	Buffer Subscribed — The message buffer filter matches next slot and cycle.
SLS	Slot start	Slot Start — Start of either Static Slot or Dynamic Slot.
SNS	Symbol window start or NIT start	Symbol Window or NIT Start — Start of either Symbol Window or NIT.
SSS	Slot start or symbol window start or NIT start	Slot or Segment Start — Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	Status updated	Status Updated — Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in Table 29-118, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 29-118. Receive message buffer transition priorities

State	Priorities	Description
Module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

29.6.6.3.2 Message Reception

As a result of the message buffer search, the CC changes the state of up to two enabled receive message buffers from either idle state Idle or locked state HLck to either the subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffer is assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module triggers the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data is written into the shadow buffers. For details on receive shadow buffers, see Section 29.6.6.3.5, Receive Shadow Buffers Concept. The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

29.6.6.3.3 Message Buffer Update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data is lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the CHI Error Flag Register (FR_CHIERFR).

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE, the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in Table 29-119.

Table 29-119. Receive message buffer update

vSS!ValidFrame	*vRF!Header!NFIndicator*	Update description
1	1	Valid non-null frame received. • Message buffer data field updated. • Frame header field updated. • Slot status field updated. • DUP:= 1 • DVAL:= 1 • MBIF:= 1
1	0	Valid null frame received. • Message buffer data field *not* updated. • Frame header field *not* updated. • Slot status field updated. • DUP:= 0 • DVAL *not* changed • MBIF:= 1
0	x	No valid frame received. • Message buffer data field not updated. • Frame header field not updated. • Slot status field updated. • DUP:= 0 • DVAL *not* changed. • MBIF:= 1, if the slot was not an empty dynamic slot. **Note:** An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

NOTE

If the number of the last slot in the current communication cycle on a given channel is *n*, then all receive message buffers assigned to this channel with FR_MBFIDR*n*[FID] > *n* will not be updated at all.

When the receive message buffer update has finished, the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example diagram for a receive message buffer timing and state change for a normal frame reception is given in Figure 29-142.

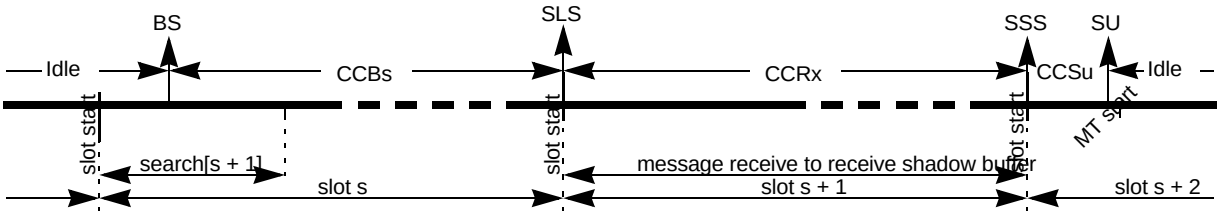


Figure 29-142. Message reception timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following items:

1. The message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).
2. The message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (FR_MBDSR).
3. The number of bytes received over the FlexRay bus.

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2 × FR_MBDSR.MBSEG1DS, the CC writes only 2 × FR_MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2 × FR_MBDSR.MBSEG1DS, the CC writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with FR_MBDSR.MBSEG2DS.

29.6.6.3.4 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in Section 29.6.3.1, Individual Message Buffers.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the CC will not change the content of the message buffer.

29.6.6.3.5 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in Section 29.6.3.2, Receive Shadow Buffers.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see Table 29-119), the CC writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the Message Buffer Index Registers (FR_MBIDXRn) with the content of the corresponding internal shadow buffer index register. In all other cases, the CC writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FlexRay memory area located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FlexRay memory area accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the Message Buffer Index Registers (FR_MBIDXRn). Instead, the index of the message buffer header field must be fetched from the Message Buffer Index Registers (FR_MBIDXRn).

29.6.7 Individual Message Buffer Search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines, for each enabled channel, if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm that is invoked at the following protocol-related events:

- NIT start
- Slot start in the static segment
- Minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n + 1*.

In general, the message buffer search for the next slot *n* considers only message buffers that are:

- Enabled (FR_MBCCSR*n*[EDS] = 1)
- Matches the next slot *n* (FR_MBFIDR*n*[FID] = *n*)

In addition, for the static segment only those message buffers are considered that match the condition of at least one row of Table 29-120. For the dynamic segment, only those message buffers are considered that match the condition of at least one row of Table 29-121. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers, the message buffers selected are those with the highest priority according to Table 29-120 for the static segment and according to Table 29-121 for the dynamic segment.

Table 29-120. Message buffer search priority (static segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	Transmit buffer, matches cycle count, not locked and committed	MA
1	1	—	0	1	Transmit buffer, matches cycle count, not committed	SA
	1	1	—	1	Transmit buffer, matches cycle count, locked	SA
2	1	—	—	—	Transmit buffer	SA
3	0	0	n/a	1	Receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	Receive buffer, matches cycle count, locked	SB

[1] Cycle Counter Filter Match, see Section 29.6.7.1, Message Buffer Cycle Counter Filtering.

Table 29-121. Message buffer search priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	Transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	Receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	Receive buffer, matches cycle count, locked	SB

1 Cycle Counter Filter Match, see Section 29.6.7.1, Message Buffer Cycle Counter Filtering.

If there are multiple message buffers with highest priority, the message buffer with the lowest message buffer number is selected. All message buffers that have the highest priority must have a consistent channel assignment as specified in Section 29.6.7.2, Message Buffer Channel Assignment Consistency.

Depending on the message buffer channel assignment, the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in Section 29.6.3.1, Individual Message Buffers.

29.6.7.1 Message Buffer Cycle Counter Filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn). This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled (CCFE = 0), this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

$$\text{MBCCFR}n\text{[CCFE]} = 0 \qquad \textit{Eqn. 29-11}$$

$$\text{CYCCNT \& MBCCFR}n\text{[CCFMSK]} = \text{MBCCFR}n\text{[CCFVAL] \& MBCCFR}n\text{[CCFMSK]} \qquad \textit{Eqn. 29-12}$$

29.6.7.2 Message Buffer Channel Assignment Consistency

The message buffer channel assignment given by the CHA and CHB bits in the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn) defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if FR_MBCCFR*n*[CHA] = 1 and transmits or receives on channel B if FR_MBCCFR*n*[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not

defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in Figure 29-143.

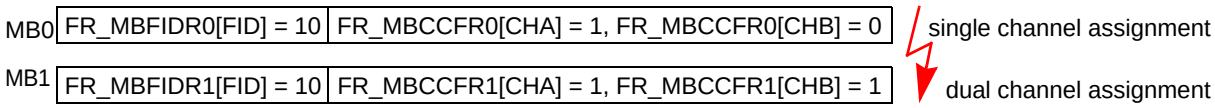


Figure 29-143. Inconsistent channel assignment

29.6.7.3 Node Related Slot Multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if transmit and receive message buffers are configured for the same slot.

According to Table 29-121 the transmit buffer is only found if the cycle counter filter matches and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

29.6.7.4 Message Buffer Search Error

Two kinds of errors, “Message Buffer Search Start while Running” and “Illegal Message Buffer Index Found,” may occur during message buffer search[1].

29.6.7.4.1 Message Buffer Search Start while Running

If the message buffer search is running in slot $n - 1$ and the next message buffer search start event occurs due to the start of slot n, the message buffer search engine is stopped and the Message Buffer Search Error Flag MBS_EF is set in the CHI Error Flag Register (FR_CHIERFR). As a result of this stop, no individual message buffer is identified for transmission or reception in slot n. Additionally, the search engine will not be started in slot n, and consequently no individual message buffer is identified for transmission or reception in slot $n + 1$.

A message buffer search error occurs only if the CHI frequency is too slow to allow the search through all message buffers to be completed within the NIT or a minislot.

For more details of minimum required CHI frequency see Section 29.7.5, Number of Usable Message Buffers.

29.6.7.4.2 Illegal Message Buffer Index Found

If the message buffer search has finished the message buffer search in slot $n-1$, it retrieves the data offset values for the found message buffers and the receive shadow buffers. If one of these message buffers contains an illegal message buffer index, the Message Buffer Search Error Flag MBS_EF is set in the CHI Error Flag Register (FR_CHIERFR) and no individual message buffer is identified for transmission or reception in slot n. The legal message buffer index values for the individual and receive shadow buffers are specified in Section 29.5.2.52, Receive Shadow Buffer Index Register (FR_RSBIR) and Section 29.5.2.82, Message Buffer Index Registers (FR_MBIDXRn).

1. The FIFO reception is not affected by the search errors. Additionally, if no rx buffer has been found due to a search error, the received frame is considered for FIFO reception.

29.6.8 Individual Message Buffer Reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on Specific Configuration Data. The common configuration data given in the section on Specific Configuration Data cannot be reconfigured when the protocol is out of the *POC:config* state.

29.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, these reconfiguration schemes are provided:

- Basic Type Not Changed (RC1)
- Buffer Type Not Changed (RC2)

29.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if the message buffer transfer direction bit FR_MBCCSR*n*[MTD] is not changed. This type of reconfiguration is denoted by RC1 in Figure 29-144. Transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state.

29.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer. This type of reconfiguration is denoted by RC2 in Figure 29-144. It applies to transmit and receive message buffers. Transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

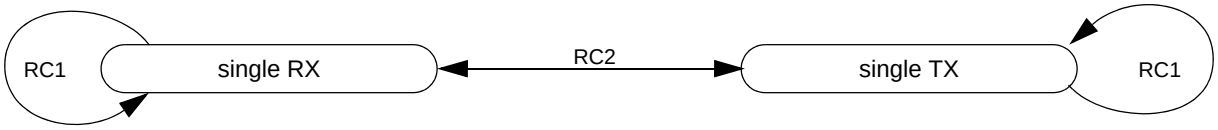


Figure 29-144. Message buffer reconfiguration scheme

29.6.9 Receive FIFOs

This section describes the two receive FIFOs.

29.6.9.1 Overview

The two receive FIFOs implement the queued message buffer concept defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One FIFO is assigned to channel A, the other FIFO is assigned to channel B. Both FIFOs work completely independently of each other.

The message buffer structure of each FIFO is described in Section 29.6.3.3, Receive FIFO. The area in the FlexRay memory area for each of the two FIFOs is characterized by:

- The FIFO system memory base address

- The index of the first FIFO entry given by Receive FIFO Start Index Register (FR_RFSIR)
- The data field offset of the data field belonging to the first FIFO entry given by Receive FIFO Start Data Offset Register (FR_RFSDOR)
- The number of FIFO entries and the length of each FIFO entry as given by Receive FIFO Depth and Size Register (RFDSR)

29.6.9.2 FIFO Configuration

The FIFOs can be configured for two different locations of the system memory base address via the FIFO address mode bit FAM in the Module Configuration Register (FR_MCR).

29.6.9.2.1 Single System Memory Base Address Mode

This mode is configured when the FIFO address mode flag FR_MCR[FAM] is set to 0. In this mode, the location of the system memory base address for the FIFO buffers is System Memory Base Address Register (FR_SYMBADR).

29.6.9.2.2 Dual System Memory Base Address Mode

This mode is configured when the FIFO address mode flag FR_MCR[FAM] is set to 1. In this mode, the location of the system memory base address for the FIFO buffers is Receive FIFO System Memory Base Address Register (FR_RFSYMBADR).

The FIFO control and configuration data is given in Section 29.6.3.7, Receive FIFO Control and Configuration Data. The configuration of the FIFOs consists of two steps.

The first step is the allocation of the required amount of FlexRay memory area for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see Section 29.6.4, Flexray Memory Area Layout.

The second step is the programming of the configuration data register while the PE is in *POC:config*.

The following steps configure the layout of the FIFO:

1. Configure the FIFO update and address modes in Module Configuration Register (FR_MCR).
2. Configure the FIFO system memory base address.
3. Configure the Receive FIFO Start Index Register (FR_RFSIR) with the first message buffer header index that belongs to the FIFO.
4. Configure the Receive FIFO Start Data Offset Register (FR_RFSDOR) with the data field offset of the data field belonging to the first message buffer that belongs to the FIFO.
5. Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO entry size.
6. Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO depth.
7. Configure the FIFO filters.

29.6.9.3 FIFO Periodic Timer

The FIFO periodic timer is used to generate a FIFO almost-full interrupt at a certain point in time, if the almost-full watermark is not reached but the FIFO is not empty. This can be used to prevent frames from getting stuck in the FIFO for a long time.

The FIFO periodic timer is configured via the Receive FIFO Periodic Timer Register (FR_RFPTR). If the periodic timer duration FR_RFPTR[PTD] is configured to 0x0000, the periodic timer is continuously expired. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x3FFF, the periodic timer never expires. If the periodic timer is configured to a value *ptd*, greater than 0x0000 and smaller than 0x3FFF, the periodic timer expires and is restarted at the start of every communication cycle, and expires and is restarted after *ptd* macroticks have elapsed.

29.6.9.4 FIFO Reception

The FIFO reception is a CC internal operation.

A message frame reception is directed into the FIFO if no individual message buffer is assigned for transmission or subscribed for reception for the current slot. When this is the case, the FIFO filter path shown in Figure 29-145 is activated.

If the FIFO filter path indicates that the received frame has to be appended to the FIFO and the FIFO is not full, the CC writes the received frame header into the message buffer header field indicated by the CC internal FIFO write index. The frame payload data is written into the corresponding message buffer data field. If the status of the received frame indicates a valid non-null frame, then the slot status information is written into the message buffer header field, the CC internal FIFO write index is updated by one, and the FIFO fill level FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is incremented.If the status of the received frame indicates an invalid or null frame, the frame is not appended to the FIFO.

29.6.9.5 FIFO Almost-Full Interrupt Generation

If the FIFO fill level FLA (FLB) is updated after a frame reception and exceeds the FIFO watermark level WM, i.e., FLA > WM_A (FLB > WM_B), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.If the periodic timer expires, and FIFOA (FIFOB) is not empty, i.e., FLA > 0 (FLB > 0), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.

29.6.9.6 FIFO Overflow Error Generation

If the FIFOA (FIFOB) is full, i.e., FLA = $FIFO\_DEPTH_A$ (FLB = $FIFO\_DEPTH_B$) and the conditions for a FIFO reception as described in Section 29.6.9.4, FIFO Reception, are fulfilled, then the FIFO overflow error flag FR_CHIERFR[FOVA_EF] (FR_CHIERFR[FOVB_EF]) is asserted.

29.6.9.7 FIFO Message Access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level given in the fields FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is greater than 0. The Receive FIFO A

Read Index Register (FR_RFARIR) and the Receive FIFO B Read Index Register (FR_RFBRIR) point to a message buffer with valid content and the oldest frames stored in the FIFO.The respective read data field offsets can be calculated according to Equation 29-6.

If the FIFO fill level FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is 0, than the FIFOA (FIFOB) contains no valid messages and the corresponding read index register Receive FIFO A Read Index Register (FR_RFARIR) or Receive FIFO B Read Index Register (FR_RFBRIR) point to a message buffer with invalid content. In this case the application must not read data from this FIFO.

To access the oldest message in the FIFOA (FIFOB), the application first reads the read index RDIDX out of the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)). This read index points to the message buffer header field of the oldest message buffer that contains valid received message data. The data field offset belonging to this message buffer must be calculated by the application according to Equation 29-6. The application can access the message data as described in Section 29.6.3.3, Receive FIFO. When the application has read the message buffer data and status information, it can update the FIFO as described in Section 29.6.9.8, FIFO Update.

29.6.9.8 FIFO Update

The application updates the FIFOA (FIFOB) by writing a pop count value *pc* different from 0 to the PCA (PCB) field in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR).

As a result of the this operation, the CC removes the oldest *pc* entries from FIFOA (FIFOB).

If the specified pop count value *pc* is greater than the current fill level *fl* provided in FLA (FLB) field, then only *fl* entries are removed from the FIFOA (FIFOB). The remaining *fl-pc* requested pop operations are discarded without any notification. In this case FIFOA (FIFOB) is empty after the update operation.

The read index in the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)) is incremented by the number of removed items. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index defined in Receive FIFO Start Index Register (FR_RFSIR) automatically.

29.6.9.8.1 FIFO Interrupt Flag Update

The FIFO Interrupt Flag Update mode is configured when the FIFO update mode flag FR_MCR[FUM] is set to 0. In this mode, FIFOA (FIFOB) will be updated by one entry when the interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is written with 1 by the application.

If the FIFO is empty, the update request is ignored without any notification.

The read index in the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)) is incremented by 1 if the FIFO was not empty. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index automatically.

29.6.9.9 FIFO Filtering

The FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The CC provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in Figure 29-145.

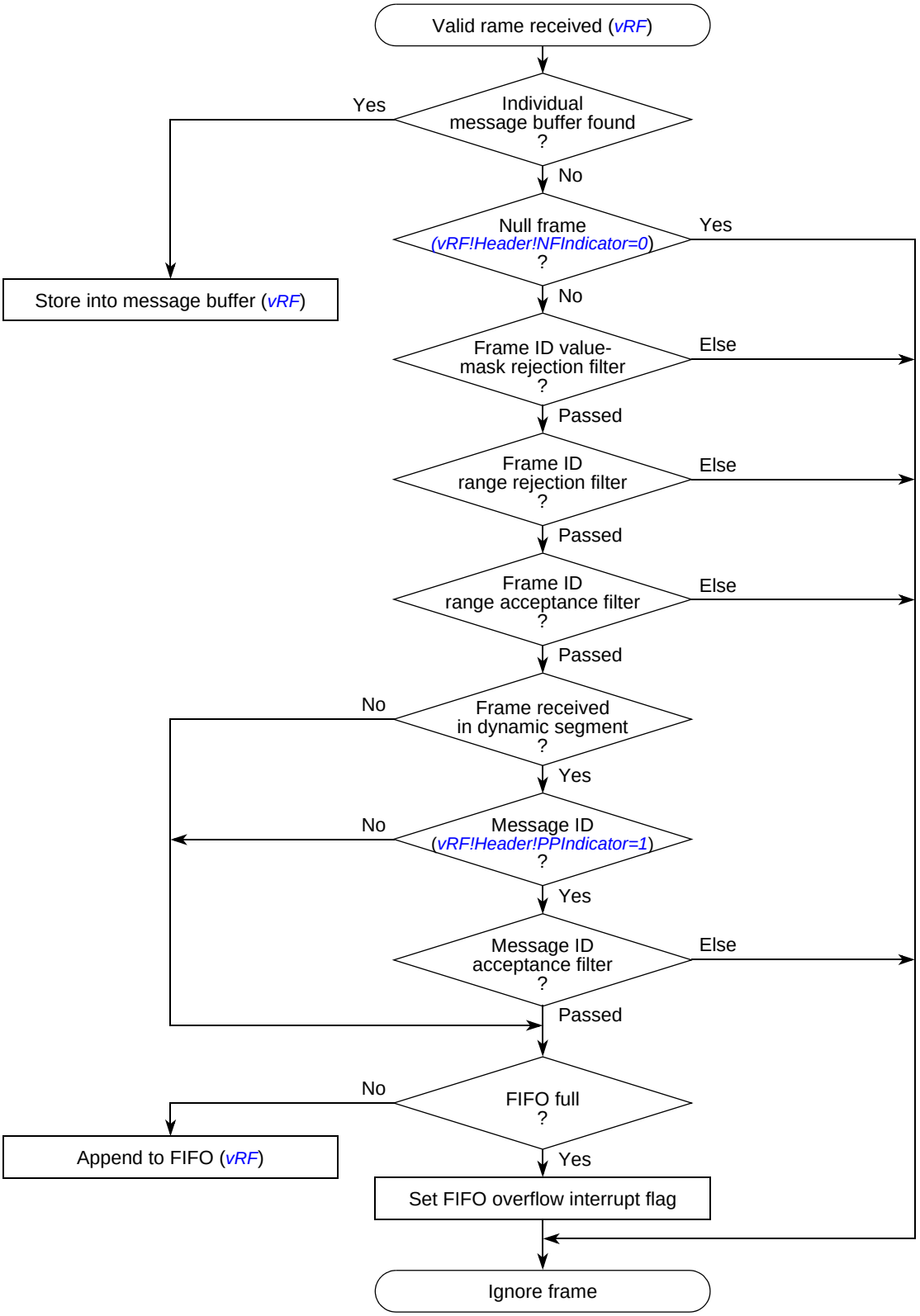


Figure 29-145. Received frame FIFO filter path

A received frame passes the FIFO filtering if it has passed all three types of filter.

29.6.9.9.1 RX FIFO Frame ID Value-Mask Rejection Filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR) and the Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR). Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e., is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if Equation 29-13 is fulfilled.

$$\text{FID \& FR\_RFFIDRFMR[FIDRFMSK]} \neq \text{FR\_RFFIDRFVR[FIDRFVAL] \& FR\_RFFIDRFMR[FIDRFMSK]} \qquad \textit{Eqn. 29-13}$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- FR_RFFIDRFVR[FIDRFVAL]:= 0x000 and FR_RFFIDRFMR[FIDRFMSK]:= 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- FR_RFFIDRFMR[FIDRFMSK]:= 0x000

Using the settings above, Equation 29-13 can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

29.6.9.9.2 RX FIFO Frame ID Range Rejection Filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (FR_RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e., FR_RFRFCTR[FiMD] = 1 and FR_RFRFCTR[FiEN] = 1, Equation 29-14 is fulfilled.

$$\text{FID} < \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 0}}] \quad \text{or} \quad \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 1}}] < \text{FID} \qquad \textit{Eqn. 29-14}$$

Consequently, all frames with a frame ID that fulfills Equation 29-15 for at least one of the enabled rejection filters will be rejected and thus not pass.

$$\text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 0}}] \leq \text{FID} \leq \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 1}}] \qquad \textit{Eqn. 29-15}$$

29.6.9.9.3 RX FIFO Frame ID Range Acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (FR_RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e., FR_RFRFCTR[FiMD] = 0 and FR_RFRFCTR[FiEN] = 1, Equation 29-16 is fulfilled.

$$FR\_RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq FR\_RFRFCFR_{SEL}[SID_{IBD=1}] \qquad \textit{Eqn. 29-16}$$

29.6.9.9.4 RX FIFO Message ID Acceptance Filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR) and the Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR). This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if Equation 29-17 is fulfilled.

$$MID \,\&\, FR\_RFMIDAFMR[MIDAFMSK] = FR\_RFMIDAFMR[MIDAFVAL] \,\&\, FR\_RFMIDAFMR[MIDAFMSK] \qquad \textit{Eqn. 29-17}$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- FR_RFMIDAFMR[MIDAFMSK]:= 0x000

Using the settings above, Equation 29-17 is always fulfilled and all frames will pass.

29.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the CC.

29.6.10.1 Dual Channel Device Mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX and $\overline{\text{FR\_A\_TX\_EN}}$ is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and $\overline{\text{FR\_B\_TX\_EN}}$ is connected to the physical bus channel B. The dual channel system is shown in Figure 29-146.

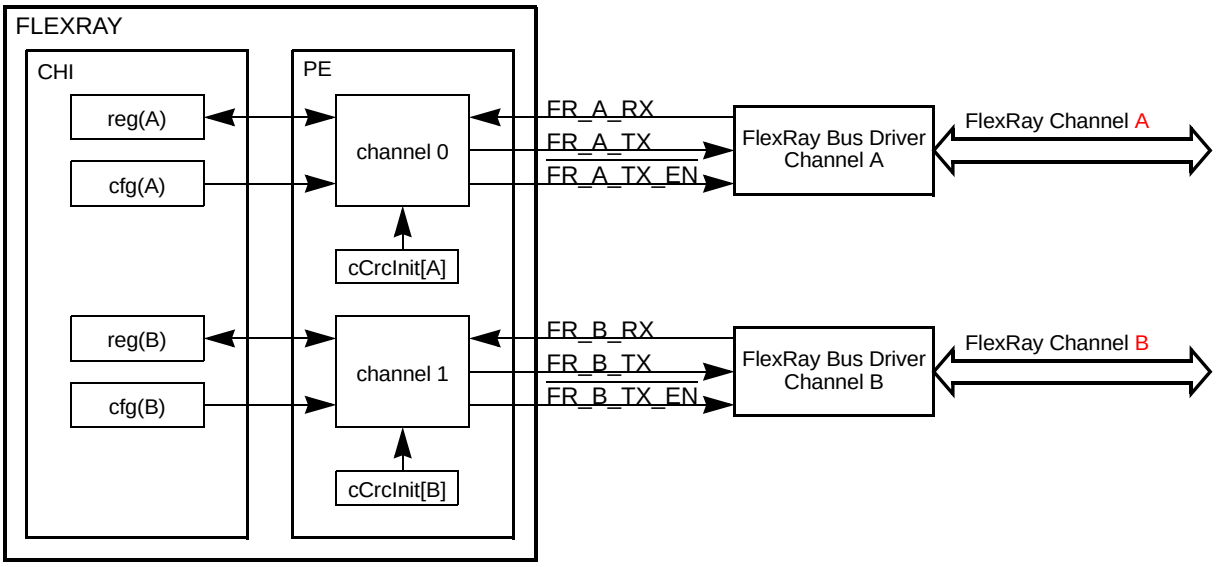


Figure 29-146. Dual channel device mode

29.6.10.2 Single Channel Device Mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ and can be connected to either the physical bus channel A (shown in Figure 29-147) or the physical bus channel B (shown in Figure 29-148).

If the device is configured as a single channel device by setting FR_MCR[SCM] to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of FR_MCR[CHA] and FR_MCR[CHB], the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit FR_MCR[CHA] must be set if the FlexRay Port A is connected to a FlexRay Channel A. The bit FR_MCR[CHB] must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

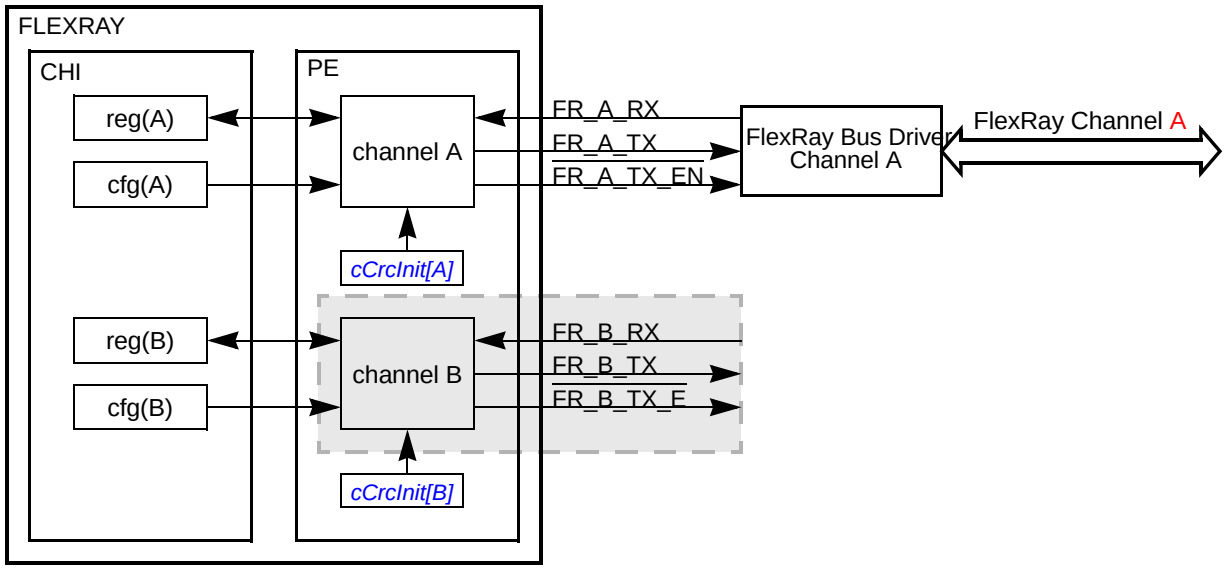


Figure 29-147. Single channel device mode (Channel A)

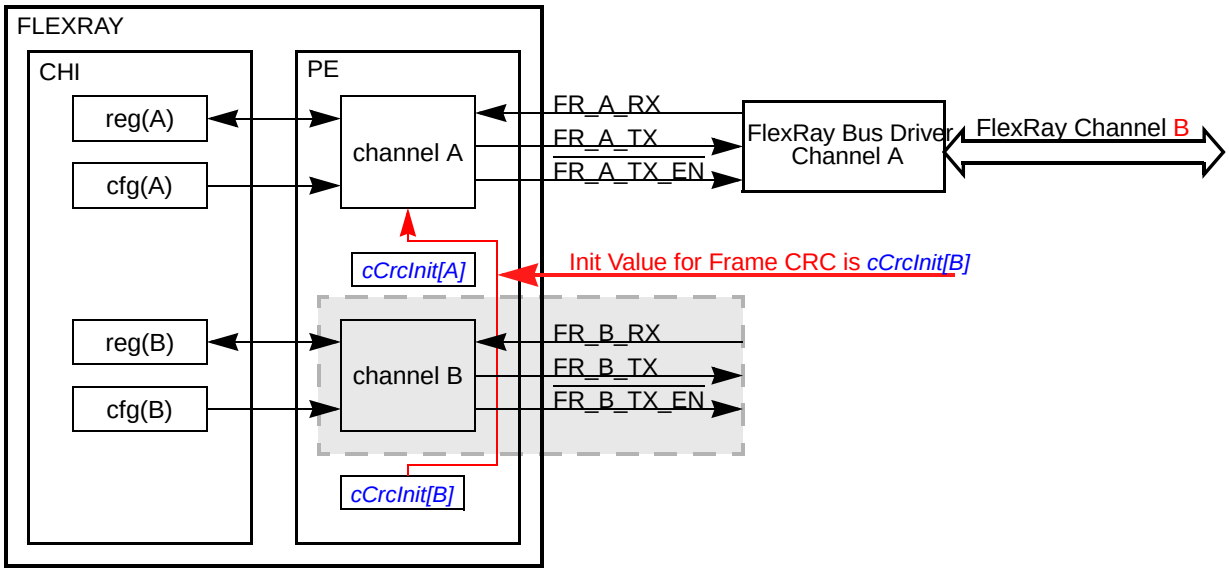


Figure 29-148. Single channel device mode (Channel B)

29.6.11 External Clock Synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the Protocol Operation Control Register (FR_POCR). The PE applies the external correction values in the next even-odd cycle pair as shown in Figure 29-149 and Figure 29-150.

NOTE

The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these values were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2*n* + 1 shall be affected by the external offset correction, the EOC_AP field must be written to after the start of cycle 2*n* and before the end of the static segment of cycle 2*n* + 1. If this field is written to after the end of the static segment of cycle 2*n* + 1, it is not guaranteed that the external correction value will be applied in cycle 2*n* + 1. If the value is not applied in cycle 2*n* + 1, then the value will be applied in the cycle 2*n* + 3. Refer to Figure 29-149 for timing details.

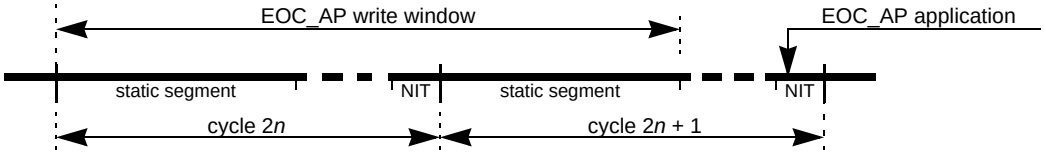


Figure 29-149. External offset correction write and application timing

If the rate correction for the cycle pair [2*n* + 2, 2*n* + 3] shall be affected by the external offset correction, the ERC_AP field must be written to after the start of cycle 2*n* and before the end of the static segment start of cycle 2*n* + 1. If this field is written to after the end of the static segment of cycle 2*n* + 1, it is not guaranteed that the external correction value will be applied in cycle pair [2*n* + 2, 2*n* + 3]. If the value is not applied for cycle pair [2*n* + 2, 2*n* + 3], then the value will be applied for cycle pair [2*n* + 4, 2*n* + 5]. Refer to Figure 29-150 for details.

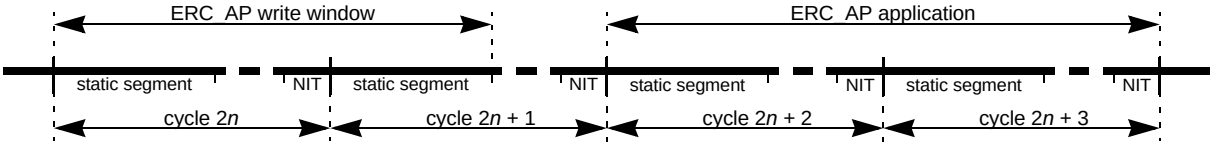


Figure 29-150. External rate correction write and application timing

29.6.12 Sync Frame ID and Sync Frame Deviation Tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The CC provides the means to write a copy of these internal tables into the FlexRay memory area and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the CC will not overwrite these tables and the application can read a consistent snapshot.

NOTE

Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

29.6.12.1 Sync Frame ID Table Content

The Sync Frame ID Table is a snapshot of the protocol-related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

29.6.12.2 Sync Frame Deviation Table Content

The Sync Frame Deviation Table is a snapshot of the protocol-related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

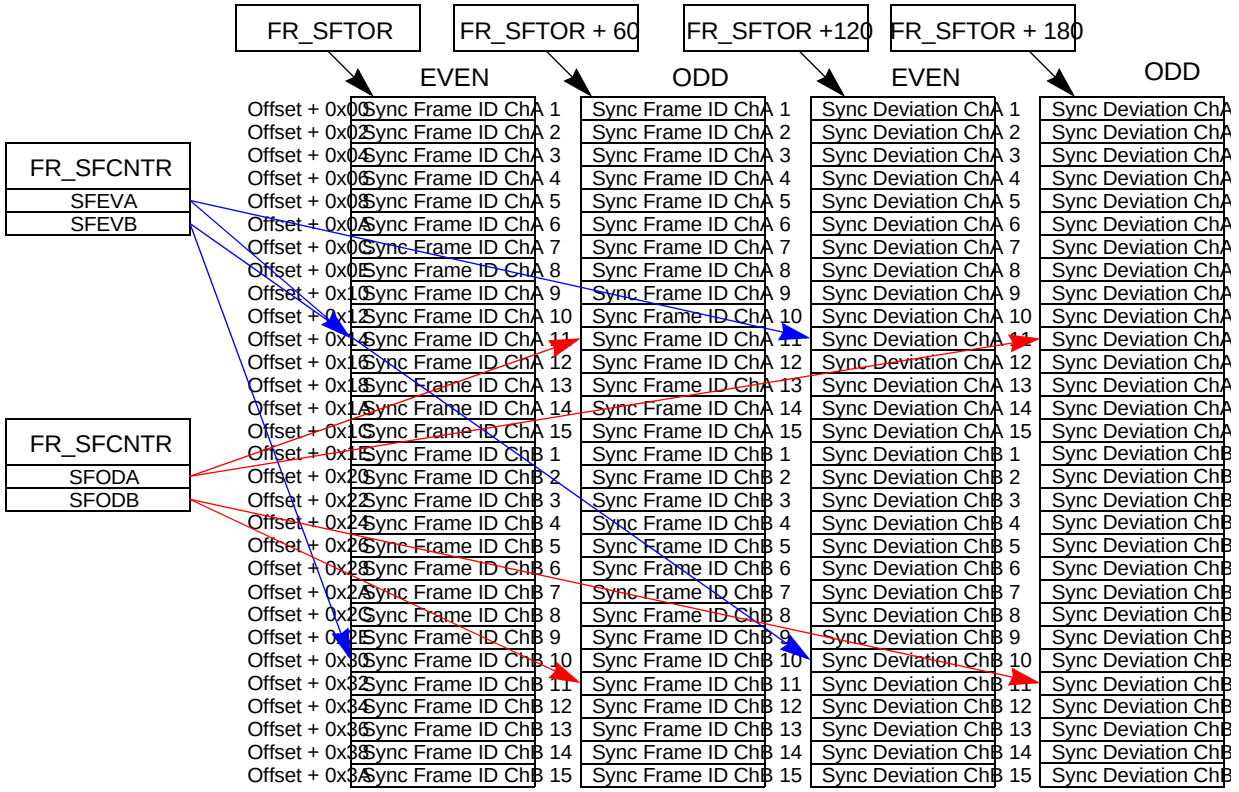


Figure 29-151. Sync table memory layout

29.6.12.3 Sync Frame ID and Sync Frame Deviation Table Setup

The CC writes a copy of the internal synchronization frame ID and deviation tables into the FlexRay memory area if requested by the application. The application must provide the appropriate amount of FlexRay memory area for the tables. The memory layout of the tables is given in Figure 29-151. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the Sync Frame Table Offset Register (FR_SFTOR).

29.6.12.4 Sync Frame ID and Sync Frame Deviation Table Generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FlexRay memory area using the Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR). A summary of the copy modes is given in Table 29-122.

Table 29-122. Sync frame table generation modes

FR_SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
1	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The sync frame table generation process is described here for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting FR_SFTCCSR[SIDEN] to 1, the CC starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The CC checks if the application has locked the tables by reading the FR_SFTCCSR[ELKS] lock status bit. If this bit is set, the CC will not update the table in this cycle. If this bit is cleared, the CC locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the CC clears the even table valid status bit FR_SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the FlexRay memory area, the CC sets the even table valid bit FR_SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the Protocol Interrupt Flag Register 1 (FR_PIFR1). If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the CC from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger FR_SFTCCSR[ELKT]. When the even table is not currently updated by the CC, the lock is granted and the even table lock status bit FR_SFTCCSR[ELKS] is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the Sync Frame Counter Register (FR_SFCNTR). The value in the FR_SFTCCSR[CYCNUM] field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the FR_SFCNTR[SFEVA] and FR_SFCNTR[SFEVB] fields. The application can now start to read the sync table data from the locations given in Figure 29-151.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger FR_SFTCCSR[ELKT] again. The even table lock status bit FR_SFTCCSR[ELKS] is reset immediately.

If the sync frame table generation is disabled, the table valid bits FR_SFTCCSR[EVAL] and FR_SFTCCSR[EVAL] are reset when the counter values in the Sync Frame Counter Register (FR_SFCNTR) are updated. This is done because the tables stored in the FlexRay memory area are no longer related to the values in the Sync Frame Counter Register (FR_SFCNTR).

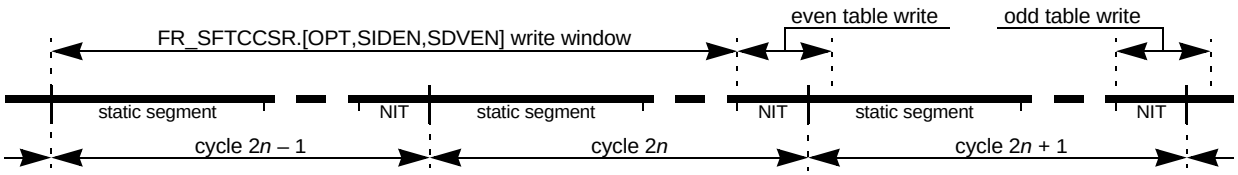


Figure 29-152. Sync frame table trigger and generation timing

29.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the FlexRay memory area during the table write windows shown in Figure 29-152. During the table write, the application cannot lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

29.6.12.5.1 Sync Frame Table Locking and Unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR). If the affected table is not currently written to the FlexRay memory area, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FlexRay memory area, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

29.6.13 MTS Generation

The CC provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the MTS A Configuration Register (FR_MTSACFR) and MTS B Configuration Register (MTSBCFR).

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the MTS A Configuration Register (FR_MTSACFR) or MTS B Configuration Register (MTSBCFR). If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if Equation 29-18, Equation 29-19, and Equation 29-20 are fulfilled.

FR_PSR0[PROTSTATE] = *POC:normal active* ***Eqn. 29-18***

FR_MTSACRF[MTE] = 1 ***Eqn. 29-19***

CYCCNT & FR_MTSACFR[CYCCNTMSK] = FR_MTSACFR[CYCCNTVAL] & FR_MTSACFR[CYCCNTMSK] ***Eqn. 29-20***

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if Equation 29-18, Equation 29-21, and Equation 29-22 are fulfilled.

FR_MTSBCRF[MTE] = 1 ***Eqn. 29-21***

CYCCNT & FR_MTSBCFR[CYCCNTMSK] = FR_MTSBCFR[CYCCNTVAL] & FR_MTSBCFR[CYCCNTMSK] ***Eqn. 29-22***

29.6.14 Key Slot Transmission

29.6.14.1 Key Slot Assignment

A key slot is assigned to the CC if the key_slot_id field in the Protocol Configuration Register 18 (FR_PCR18) is configured with a value greater than 0 and less than or equal to number_of_static_slots in Protocol Configuration Register 2 (FR_PCR2), otherwise no key slot is assigned.

29.6.14.2 Key Slot Transmission in *POC:startup*

If a key slot is assigned and the CC is in the *POC:startup* state, startup null frames will be transmitted as specified by FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

29.6.14.3 Key Slot Transmission in *POC:normal active*

If a key slot is assigned and the CC is in *POC:normal active*, a frame of the type as shown in Table 29-123 is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see Section 29.6.6.2.5, Message Transmission). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see Section 29.6.6.2.6, Null Frame Transmission).

Table 29-123. Key slot frame type

FR_PCR11[key_slot_used_for_sync]	FR_PCR11[key_slot_used_for_startup]	Key slot frame type
0	0	Normal frame
0	1	Normal frame[1]
1	0	Sync frame
1	1	Startup frame

[1] The frame transmitted has a semantically incorrect header and will be detected as an invalid frame at the receiver.

29.6.15 Sync Frame Filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e., the SFFE control bit in the Module Configuration Register (FR_MCR) is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

29.6.15.1 Sync Frame Acceptance Filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR) and the mask is configured in the Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR). A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if Equation 29-23 or Equation 29-24 evaluates to true.

$$\text{FR\_MCR[SFFE]} = 0 \qquad \textit{Eqn. 29-23}$$

$$\text{FID \& FR\_SFIDAFMR[FMSK]} = \text{FR\_SFIDAFVR[FVAL] \& FR\_SFIDAFMR[FMSK]} \qquad \textit{Eqn. 29-24}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID ≤ 1023.

29.6.15.2 Sync Frame Rejection Filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the Sync Frame ID Rejection Filter Register (FR_SFIDRFR). A received synchronization frame with the frame ID FID passes the sync frame rejection filter if Equation 29-25 or Equation 29-26 evaluates to true.

$$\text{FR\_MCR[SFFE]} = 0 \qquad \textit{Eqn. 29-25}$$

$$\text{FID} \neq \text{FR\_SFIDRFR[SYNFRID]} \qquad \textit{Eqn. 29-26}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID ≤ 1023.

29.6.16 Strobe Signal Support

The CC provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in Table 29-12.

29.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in Table 29-12 can be assigned to one of the four strobe ports using the Strobe Signal Control Register (FR_STBSCR). To assign multiple strobe signals, the application must write multiple times to the Strobe Signal Control Register (FR_STBSCR) with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to FR_STBSCR with WMD = 1 and SEL = N (updates SEL field only).
2. Read STBCSR.

The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

29.6.16.2 Strobe Signal Timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in Table 29-12 with a negative clock offset. An example waveform is given in Figure 29-153.

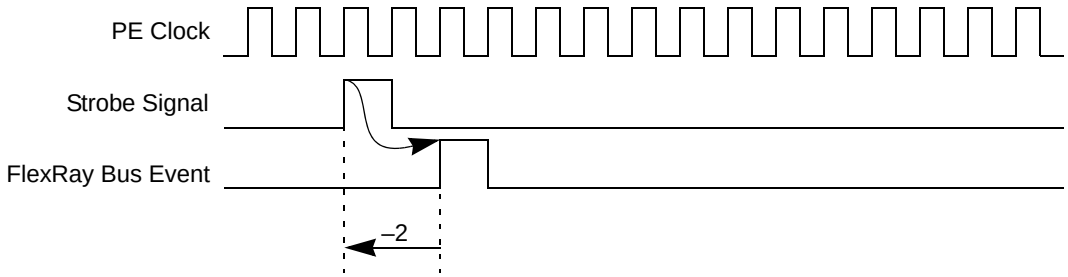


Figure 29-153. Strobe signal timing (type = pulse, clk_offset = –2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in Table 29-12 with a positive clock offset. An example waveform is given in Figure 29-154.

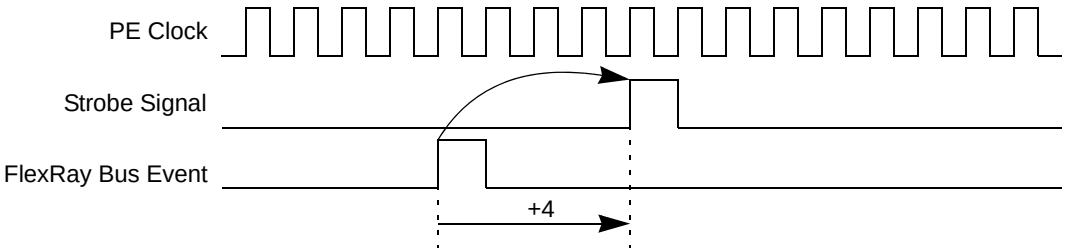


Figure 29-154. Strobe signal timing (type = pulse, clk_offset = +4)

29.6.17 Timer Support

The CC provides two timers that run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, the timer stops if it expires. In repetitive mode, the timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

29.6.17.1 Absolute Timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set at the macrotick start event, if Equation 29-27 and Equation 29-28 are fulfilled

$$\text{CYCTR[CTCCNT] \& FR\_TI1CYSR[T1\_CYC\_MSK] = FR\_TI1CYSR[T1\_CYC\_VAL] \& FR\_TI1CYSR[T1\_CYC\_MSK]} \quad \textit{Eqn. 29-27}$$

$$\text{FR\_MTCTR[MTCT] = FRMTOR[T1\_MTOFFSET]} \quad \textit{Eqn. 29-28}$$

If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (FR_PIER0) is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST bit is cleared when the timer is stopped.

29.6.17.2 Absolute / Relative Timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the Timer Configuration and Control Register (FR_TICCR). The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST bit is cleared when the timer is stopped.

29.6.17.2.1 Absolute Timer T2

If timer T2 is configured as an absolute timer, it has the same functionality as timer T1, but the configuration from Timer 2 Configuration Register 0 (FR_TI2CR0) and Timer 2 Configuration Register 1 (FR_TI2CR1) is used. On expiration of timer T2, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set. If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (FR_PIER0) is asserted, an interrupt request is generated.

29.6.17.2.2 Relative Timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set, when the programmed amount of macroticks MT[31:0], defined by Timer 2 Configuration Register 0 (FR_TI2CR0) and Timer 2 Configuration Register 1 (FR_TI2CR1), has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

29.6.18 Slot Status Monitoring

The CC provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is

described in Table 29-124. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in Figure 29-155.

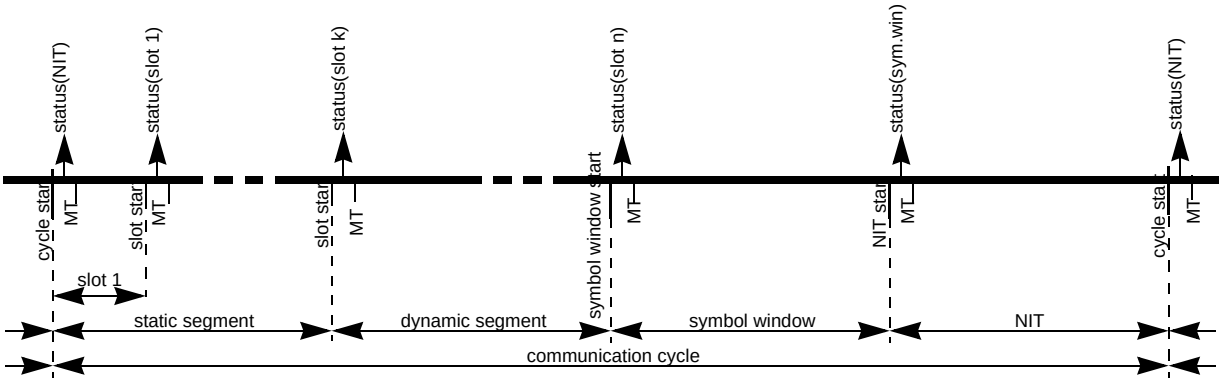


Figure 29-155. Slot status vector update

NOTE

The slot status for the NIT of cycle *n* is provided after the start of cycle *n* + 1.

Table 29-124. Slot status content

	Status content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if *vSS!ValidFrame* = 1
	b) last received header, if *vSS!ValidFrame* = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Table 29-124. Slot status content (continued)

	Status content
Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

29.6.18.1 Channel Status Error Counter Registers

The two channel status error counter registers, Channel A Status Error Counter Register (FR_CASERCR) and Channel B Status Error Counter Register (FR_CBSERCR), incremented by one, if at least one of four slot status error bits — *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* — is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap around after they have reached the maximum value.

29.6.18.2 Protocol Status Registers

The Protocol Status Register 2 (FR_PSR2) provides slot status information about the Network Idle Time NIT and the Symbol Window. The Protocol Status Register 3 (FR_PSR3) provides aggregated slot status information.

29.6.18.3 Slot Status Registers

The eight slot status registers, Slot Status Registers (FR_SSR0–FR_SSR7), can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in Table 29-124, except of the *first valid* indicator for non-transmission slots.

29.6.18.4 Slot Status Counter Registers

The CC provides four slot status error counter registers, Slot Status Counter Registers (FR_SSCR0–FR_SSCR3). Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the Slot Status Counter Condition Register

(FR_SSCCR), matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in Figure 29-156.

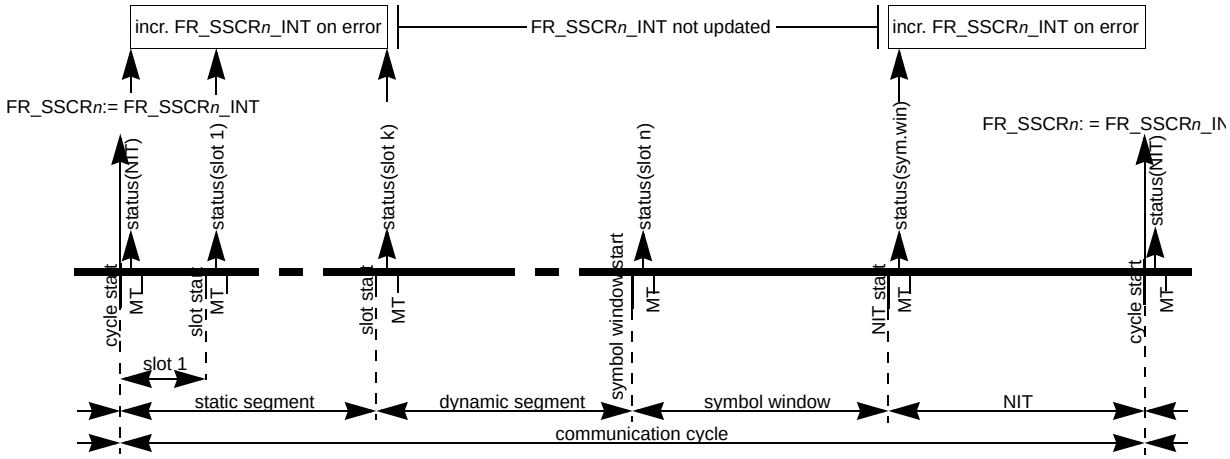


Figure 29-156. Slot status counting and FR_SSCR*n* update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the FR_SSCRn register reflects, in cycle *n*, the status of the NIT of cycle *n* – 2, and the status of all static slots and the symbol window of cycle *n* – 1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter FR_SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. Frame related condition:

   (FR_SSCCR*n*[VFR] | FR_SSCCR*n*[SYF] | FR_SSCCR*n*[NUF] | FR_SSCCR*n*[SUF]) // count on frame condition = 1;

and

   ((~FR_SSCCR*n*[VFR] | *vSS!ValidFrame*) & // valid frame restriction
   (~FR_SSCCR*n*[SYF] | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
   (~FR_SSCCR*n*[NUF] | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
   (~FR_SSCCR*n*[SUF] | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction = 1;

NOTE

The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. Slot related condition:

   ((FR_SSCCR*n*[STATUSMASK[3]] & *vSS!ContentError*) | // increment on content error
   (FR_SSCCR*n*[STATUSMASK[2]] & *vSS!SyntaxError*) | // increment on syntax error
   (FR_SSCCR*n*[STATUSMASK[1]] & *vSS!BViolation*) | // increment on boundary violation
   (FR_SSCCR*n*[STATUSMASK[0]] & *vSS!TxConflict*)) // increment on transmission conflict = 1;

If the slot status counter is in single cycle mode (FR_SSCCR*n*[MCY] = 0), the internal slot status counter FR_SSCR*n*_INT is reset at each cycle start. If the slot status counter is in the multicycle mode (FR_SSCCR*n*[MCY] = 1), the counter is not reset and incremented, until the maximum value is reached.

29.6.18.5 Message Buffer Slot Status Field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in Table 29-124 for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in Section 29.6.6, Individual message buffer functional description.

29.6.19 System Bus Access

This section provides a description of the system bus access failures and the related CC behavior. System bus access failures may occur when the CC transfers data to or from the FlexRay memory area.

The system bus access failure types are described in Section 29.6.19.1, System Bus Access Failure Types.

The behavior of the CC after the occurrence of a system bus access failure is described in Section 29.6.19.2, System Bus Access Failure Response.

29.6.19.1 System Bus Access Failure Types

This section describes the two types of system bus access failures.

29.6.19.1.1 System Bus Illegal Address Access

A system bus illegal address access is detected when the CC has used an illegal or invalid address to access the FlexRay system memory area. These conditions are treated as system bus illegal address accesses:

- The system bus subsystem detects a CC access to an illegal system memory address.
- The CC detects the usage of a data field offset with the value of 0.
- The CC detects a memory error while reading a data field offset from the CHI LRAM memory (see Section 29.6.24.3.1, CHI LRAM Error Response after CC Read).

If a system bus illegal address access is detected, the CC sets the ILSA_EF flag in the CHI Error Flag Register (FR_CHIERFR).

29.6.19.1.2 System Bus Access Timeout

A system bus access timout is detected if an access to the FlexRay memory area is not finished in time. The timeout value is derived from the SYMATOR[TIMEOUT] setting (see Section 29.7.1.1, Configure System Memory Access Time-Out Register (FR_SYMATOR)).

If a system bus access timout is detected, the CC sets the SBCF_EF flag in the CHI Error Flag Register (FR_CHIERFR).

29.6.19.2 System Bus Access Failure Response

This section describes the two types of behavior of the CC after the occurrence of a system bus access failure. The actual behavior is defined by the SBFF bit in the Module Configuration Register (FR_MCR).

29.6.19.2.1 Continue after System Bus Access Failure

If the SBFF bit in the Module Configuration Register (FR_MCR) is 0, the CC will continue its operation after the occurrence of the system bus access failure, but will not generate any system bus accesses until the start of the next communication cycle. Since no data is read from or written to the FlexRay memory area, no messages are received or transmitted. Consequently, none of the individual message buffers or receive FIFOs will be updated until the next communication cycle starts.

If the system bus failure occurs during frame transmission, a correct frame is generated with the remaining header and frame data replaced by all zeros. Depending on the point in time, this can affect the PPI bit, the Header CRC, the Payload Length (in case of a dynamic slot), and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of the next communication cycle.

29.6.19.2.2 Freeze after System Bus Access Failure

If the SBFF bit in the Module Configuration Register (FR_MCR) is set to 1, the CC will go into freeze mode immediately after the occurrence of one of the system bus access failures.

29.6.20 Interrupt Support

The CC provides 108 individual interrupt sources and five combined interrupt sources.

29.6.20.1 Individual Interrupt Sources

29.6.20.1.1 Message Buffer Interrupts

The CC provides 64 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag FR_MBCCSR*n*[MBIF] and an interrupt enable bit FR_MBCCSR*n*[MBIE]. The CC sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

29.6.20.1.2 FIFO Interrupts

The CC provides 2 FIFO interrupt sources.

Each of the 2 FIFOs provides a Receive FIFO Almost Full Interrupt Flag. The CC sets the Receive FIFO Almost Full Interrupt Flags (FR_GIFER[FAFBIF], FR_GIFER[FAFAIF]) in the Global Interrupt Flag and Enable Register (FR_GIFER) if the corresponding Receive FIFO fill level exceeds the defined watermark.

29.6.20.1.3 Wakeup Interrupt

The CC provides one interrupt source related to the wakeup.

The CC sets the Wakeup Interrupt Flag FR_GIFER[WUPIF] when it has received a wakeup symbol on the FlexRay bus. The CC generates an interrupt request if the interrupt enable bit FR_GIFER[WUPIE] is asserted.

29.6.20.1.4 Protocol Interrupts

The CC provides 25 interrupt sources for protocol-related events. For details, see Protocol Interrupt Flag Register 0 (FR_PIFR0) and Protocol Interrupt Flag Register 1 (FR_PIFR1). Each interrupt source has its own interrupt enable bit.

29.6.20.1.5 CHI Interrupts

The CC provides 16 interrupt sources for CHI-related error events. For details, see CHI Error Flag Register (FR_CHIERFR). There is one common interrupt enable bit FR_GIFER[CHIE] for all CHI error interrupt sources.

29.6.20.2 Combined Interrupt Sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that has been combined generates an interrupt request.

29.6.20.2.1 Receive Message Buffer Interrupt

The Receive Message Buffer Interrupt request is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[RBIE] is set.

29.6.20.2.2 Transmit Message Buffer Interrupt

The Transmit Message Buffer Interrupt request is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[TBIE] is asserted.

29.6.20.2.3 Protocol Interrupt

The Protocol Interrupt request is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[PRIE] is set.

29.6.20.2.4 CHI Interrupt

The CHI Interrupt request is generated when at least one of the individual CHI error interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[CHIE] is set.

29.6.20.2.5 Module Interrupt

The Module Interrupt request is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[MIE] is set.

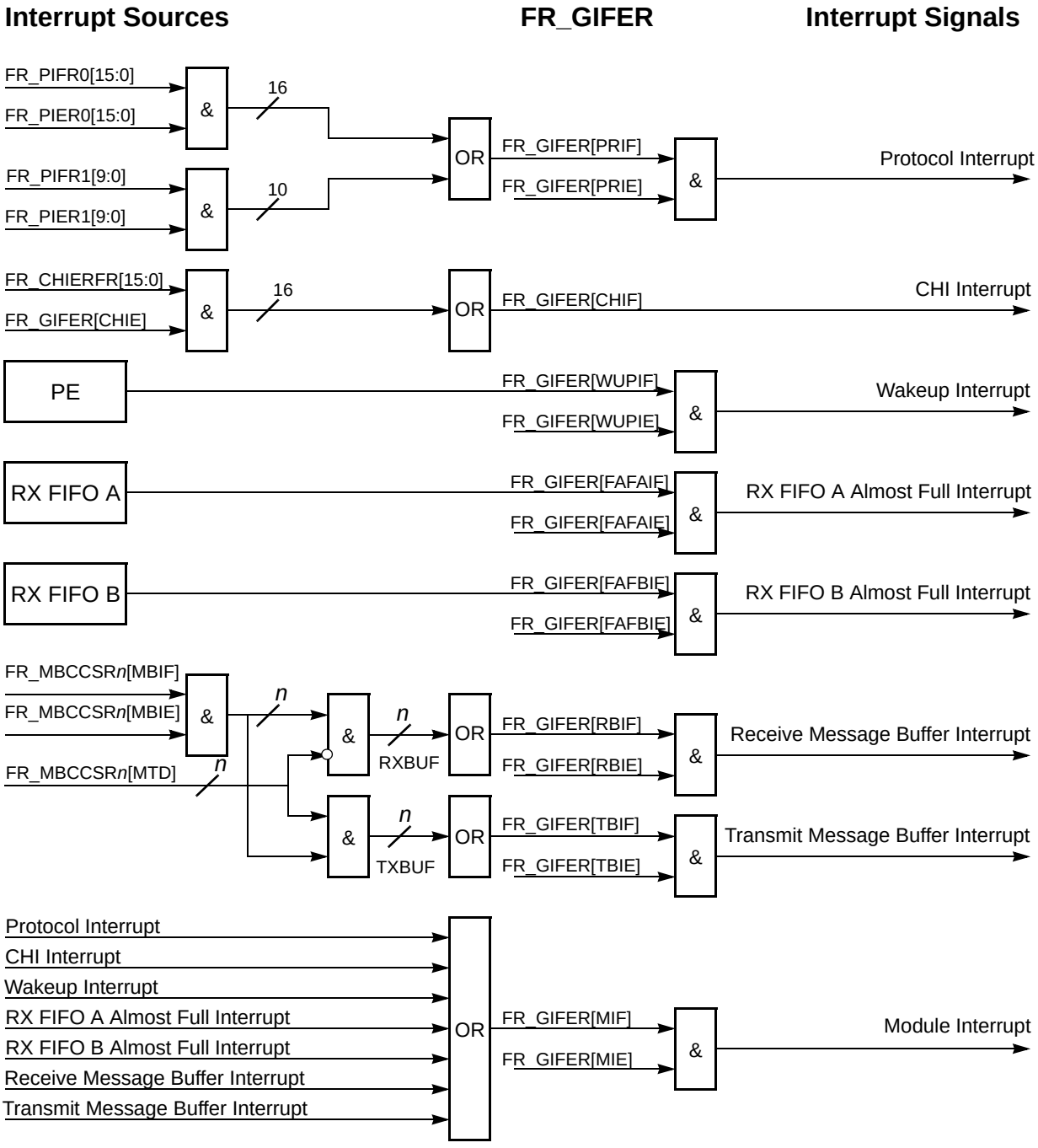


Figure 29-157. Scheme of FR_GIFER interrupt signal generation

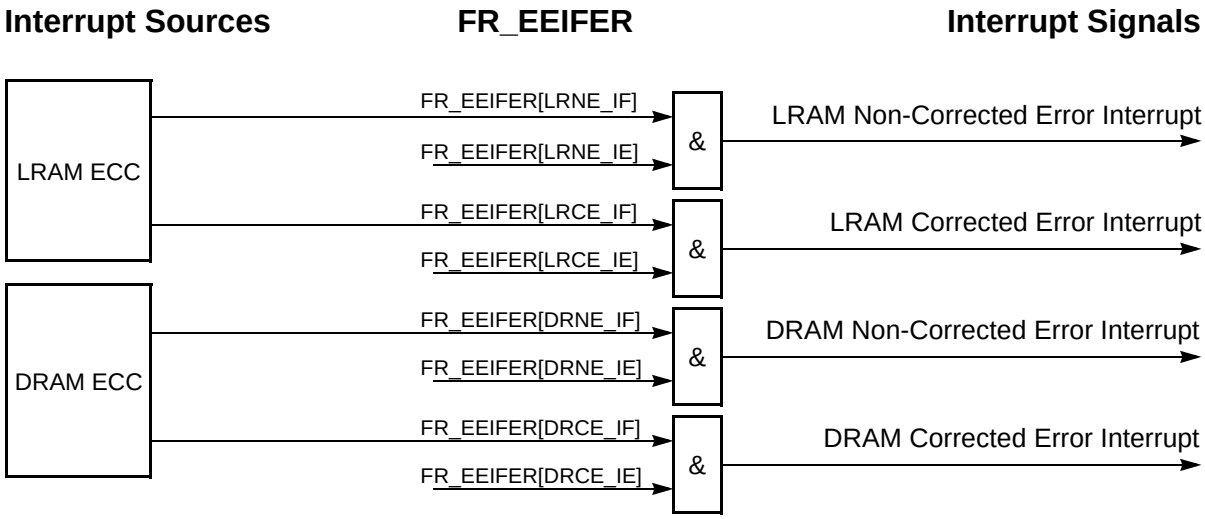


Figure 29-158. Scheme of FR_EEIFER interrupt signal generation

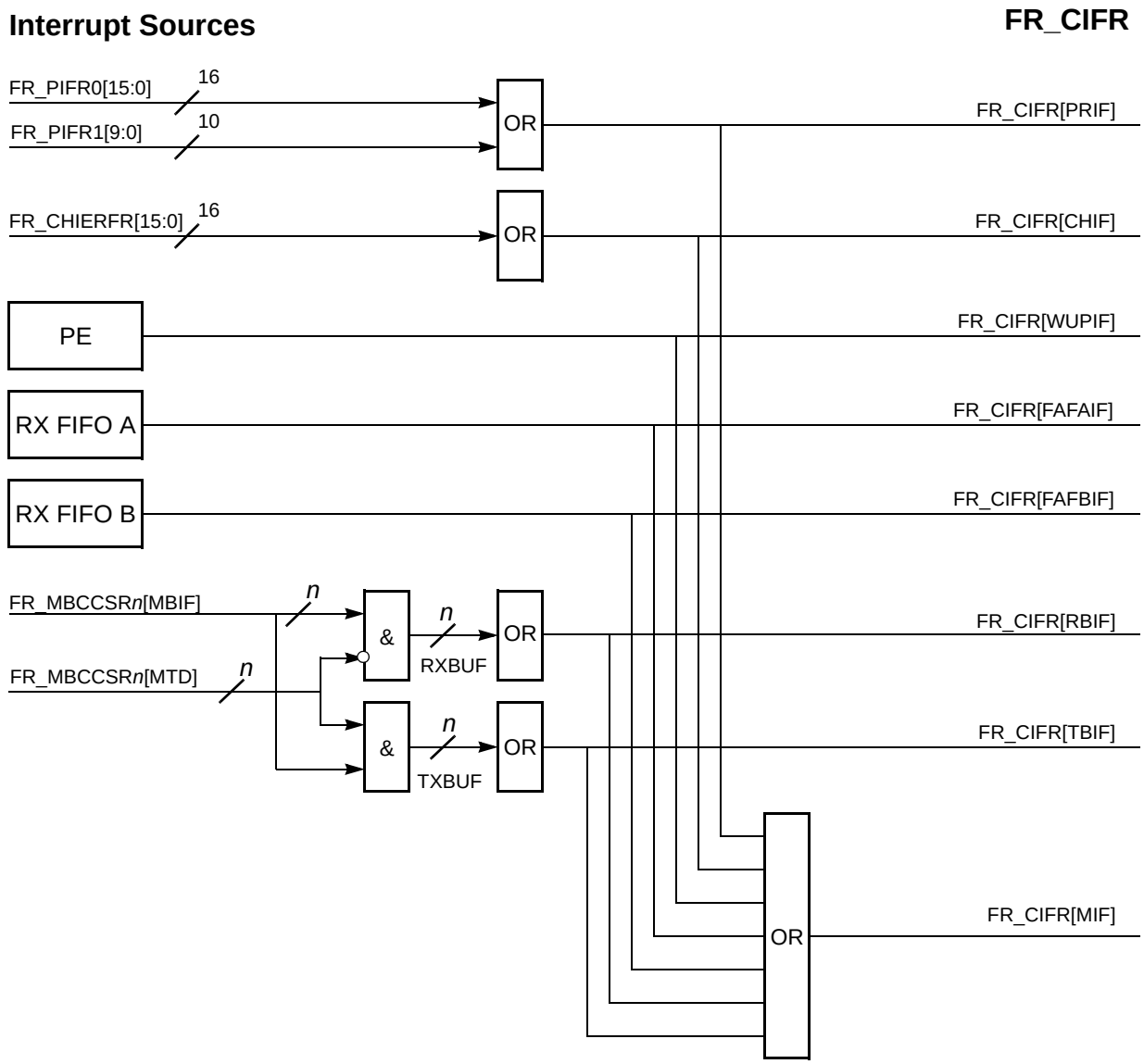


Figure 29-159. Scheme of FR_CIFR flags generation

29.6.21 Lower Bit Rate Support

The CC supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset*. The application configures the FlexRay channel bit rate by setting the BITRATE field in the Module Configuration Register (FR_MCR). The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings, and related protocol parameter values are shown in Table 29-125.

Table 29-125. FlexRay channel bit rate control

FlexRay channel bit rate [Mbit/s]	FR_MCR[BITRATE]	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	*pSamplesPerMicrotick*	*cSamplesPerBit*	*cStrobeOffset*
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

NOTE

The bit rate of 8 Mbit/s is not defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

29.6.22 PE Data Memory (PE DRAM)

The PE Data Memory (PE DRAM) is a 128-word, 16-bit wide memory with byte access, that contains the program data of the PE internal CPU. The PE DRAM is divided into two banks, 8-bit each. The memory data [7:0] are assigned to BANK0, the memory data [15:8] are assigned to BANK1.

Table 29-126. PE DRAM layout

ADDR	BANK1	BANK0
0x00	byte1	byte0
0x01	byte3	byte2
...		
0x7F	byte255	byte254

The FlexRay module provides means to access the PE DRAM from the application. The PE DRAM application access is initiated and controlled via PE DRAM Access Register (FR_PEDRAR) and PE DRAM Data Register (FR_PEDRDR). This functionality is used to check the memory error detection.

29.6.22.1 PE DRAM Read Access

A read access from the PE DRAM can be initiated in any protocol state. The following sequence describes a read access from the PE DRAM address 0x70.

1. FR_PEDRAR:= 0x50E0;
   // INST=0x5; ADDR=070
2. Wait until FR_PEDRAR[DAD] == 1;
   // Wait for end of PE DRAM access

3. val = FR_PEDRDR[DATA];
   // Read PE DRAM data

The read access is handled by the PE internal CPU with the lowest execution priority. This may cause a response delay with a maximum of 1000 PE clock cycle (25 µs).

29.6.22.2 PE DRAM Write Access

A write access into the PE DRAM can be initiated in any protocol state. The following sequence describes a write access to the PE DRAM address 0x70.

1. FR_PEDRDR:= DATA;
   // write value to be written into data register
2. FR_PEDRAR:= 0x30E0;
   // INST=0x3; ADDR=0x70
3. Wait until FR_PEDRAR[DAD] == 1;
   // Wait for end of PE DRAM access
4. val = FR_PEDRDR[DATA];
   // Read back PE DRAM data

The write access is handled by the PE internal CPU with the lowest execution priority. This may cause a response delay with a maximum of 1000 PE clock cycle (25 µs).

If the conditions given in Section 29.6.22.3, PE DRAM Write Access Limitations, are fulfilled, the data provided in PE DRAM Data Register (FR_PEDRDR) are written into the PE DRAM, read back in the next clock cycle, and stored in the PE DRAM Data Register (FR_PEDRDR). Otherwise, data is not written into the PE DRAM and 0x0000 is stored in the PE DRAM Data Register (FR_PEDRDR).

29.6.22.3 PE DRAM Write Access Limitations

The PE DRAM is used by the protocol engine if the module is not in *POC:default config* state. The only address not used by the protocol engine is 0x70. To prevent the corruption of protocol engine data, the following PE DRAM write access limitations apply for application writes:

- When the module is in *POC:default config* state, all PE DRAM addresses are writable.
- When the module is not in *POC:default config* state, only PE DRAM address 0x70 is writable.

29.6.23 CHI Lookup-Table Memory (CHI LRAM)

The CHI Lookup-Table Memory (CHI LRAM) is a CHI internal memory that contains the message buffer configuration data and the data field offsets for the physical message buffers. The configuration data for two message buffers or six data field offsets are contained in one memory row. The CHI LRAM is divided into six memory banks.

Table 29-127. CHI LRAM layout

ADR	BANK5	BANK4	BANK3	BANK2	BANK1	BANK0
0x00	FR_MBIDXR1	FR_MBFIDR1	FR_MBCCFR1	FR_MBIDXR0	FR_MBFIDR0	FR_MBCCFR0
0x01	FR_MBIDXR3	FR_MBFIDR3	FR_MBCCFR3	FR_MBIDXR2	FR_MBFIDR2	FR_MBCCFR2
...						
0x1F	FR_MBIDXR63	FR_MBFIDR63	MBCCF63	FR_MBIDXR62	MBFID62	FR_MBCCFR62
0x20	FR_MBDOR5	FR_MBDOR4	FR_MBDOR3	FR_MBDOR2	FR_MBDOR1	FR_MBDOR0
...						
0x2B	—	—	—	—	FR_MBDOR67	FR_MBDOR66
0x2C	FR_LEETR5	FR_LEETR4	FR_LEETR3	FR_LEETR2	FR_LEETR1	FR_LEETR0

29.6.23.1 CHI LRAM Read and Write Access

The CHI LRAM is accessed by the application via regular register read and write accesses.

29.6.24 Memory Content Error Detection

The FlexRay module provides integrated memory content error detection for both the CHI LRAM and PE DRAM, and memory content error correction for the PE DRAM. The memory error detection for the CHI LRAM uses a standard Hamming code with a Hamming distance of 3 and detects all single-bit and double-bit errors (SEDDED). The memory error detection and correction for the PE DRAM uses an enhanced Hamming code with a Hamming distance of 4 and detects and corrects all single-bit errors and detects all double-bit errors (SECDED).

This section describes the reporting of the occurrence of memory content errors, the reaction of the module on the occurrence, and how the application can inject memory errors in order to trigger the report and response behavior.

29.6.24.1 Memory Error Types

A memory error is the distortion of one or more bits read out of the memory. The reading of the values of all zeros and all ones is considered as a special case. The FlexRay module detects and indicates the memory errors as shown in Table 29-128. The entries on the top have higher priority.

Each memory read access reads out *all* banks of the addressed row, and runs error detection on *all* banks, even in the case that the application has triggered a read from only one bank. This may lead to the reporting of a memory error if at least one bank contains a memory error, even if an error-free bank has been read.

Table 29-128. Detected memory error types

Memory	Priority	Memory data	Indication
CHI LRAM	0 (highest)	All zeros	No error — valid data
PE DRAM			Non-corrected error
CHI LRAM		All ones	Non-corrected error
PE DRAM			
CHI LRAM	1 (lowest)	One bit flipped	Non-corrected error
PE DRAM			Corrected error
CHI LRAM		Two bits flipped	Non-corrected error
PE DRAM			
CHI LRAM		Three or more bits flipped	One out of {No error, Non-corrected error}, defined by coding given in Section 29.6.24.2.3, CHI LRAM Checkbits, and Section 29.6.24.2.3, CHI LRAM Checkbits.
PE DRAM			One out of {No error, Corrected error, Non-corrected error}, defined by coding given in Section 29.6.24.2.1, PE DRAM Checkbits, and Section 29.6.24.2.2, PE DRAM Syndrome.

29.6.24.2 Memory Error Reporting

The memory error reporting is enabled only if the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) is set.

Each of the two memories has two sets of internal registers to store the detection of one corrected and one non-corrected memory error.

If a memory error is detected, the module checks whether the related error interrupt flag in the ECC Error Interrupt Flag and Enable Register (FR_EEIFER) is set.

- If the error interrupt flag is set, the related internal error reporting register is not updated and the related error overflow flag is set to 1 to indicate a loss of error condition.
- If the error interrupt flag is not set, the internal reporting register is updated and the error interrupt flag is set to 1. If two or more memory errors of the same type are detected, the error for the bank with the lower bank number will be reported, and the error overflow flag will be set to 1.

If a memory error is detected for at least two banks of one memory, the related error overflow flag is set to 1 to indicate a loss of error condition.

29.6.24.2.1 PE DRAM Checkbits

The coding of the checkbits reported in ECC Error Report Code Register (FR_EERCR) for PE DRAM memory errors is shown in Table 29-130. This table shows the implemented enhanced Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 29-129. PE DRAM checkbits coding

CODE	CODE				DATA							
	3	2	1	0	7	6	5	4	3	2	1	0
4[1]	X	X	X	X	X	X	X	X	X	X	X	X
3[2]	—	—	—	—	X	X	X	X	—	—	—	—
2	—	—	—	—	X	—	—	—	X	X	X	—
1	—	—	—	—	—	X	X	—	X	X	—	X
0	—	—	—	—	—	X	—	X	X	—	X	X

1 The checkbit CODE[4] is set to 1 if and only if there is an even number of 1's in columns with X.

2 The checkbits CODE[3]... CODE[0] are set to 1 if and only if there is an odd number of 1's in all columns with X.

This coding of the checkbit ensures that neither 0x000 nor 0xFFF are valid code words and written into the memory.

29.6.24.2.2 PE DRAM Syndrome

The coding of the syndrome reported in ECC Error Report Code Register (FR_EERCR) for PE DRAM memory errors is shown in Table 29-130.

Table 29-130. FR_EERCR[CODE] PE DRAM syndrome coding

FR_EERCR[CODE]		Description
[4]	[3:0]	
0x1	0x0	No Error (Never occurs in error report registers)
0x0	0x0	If data == 0: Non Corrected Error (Dedicated Handling of All Zero Code Word) If data!= 0: Corrected Error (Parity Bit 4)
0x0	0x1	Corrected Error (Parity Bit 0)
0x0	0x2	Corrected Error (Parity Bit 1)
0x0	0x3	Corrected Error (Data Bit 0)
0x0	0x4	Corrected Error (Parity Bit 2)
0x0	0x5	Corrected Error (Data Bit 1)
0x0	0x6	Corrected Error (Data Bit 2)
0x0	0x7	Corrected Error (Data Bit 3)
0x0	0x8	Corrected Error (Parity Bit 3)
0x0	0x9	Corrected Error (Data Bit 4)
0x0	0xA	Corrected Error (Data Bit 5)
0x0	0xB	Corrected Error (Data Bit 6)
0x0	0xC	Corrected Error (Data Bit 7)
0x0	0xD–0xF	Non-Corrected Error
0x1	0x1–0xF	Non-Corrected Error

29.6.24.2.3 CHI LRAM Checkbits

The coding of the checkbits reported in ECC Error Report Code Register (FR_EERCR) for CHI LRAM memory errors is shown in Table 29-131. This table shows the implemented Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 29-131. CHI LRAM checkbits coding

CODE[1]	DATA															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	X	X	X	X	X	—	—	—	—	—	—	—	—	—	—	—
3	—	—	—	—	—	X	X	X	X	X	X	X	—	—	—	—
2	X	X	—	—	—	X	X	X	X	—	—	—	X	X	X	—
1	—	—	X	X	—	X	X	—	—	X	X	—	X	X	—	X
0	X	—	X	—	X	X	—	X	—	X	—	X	X	—	X	X

1 The checkbit CODE[n] is set to 1 if and only if there is an odd number of 1's in all columns with X.

29.6.24.2.4 CHI LRAM Syndrome

The coding of the syndrome reported in ECC Error Report Code Register (FR_EERCR) for CHI LRAM memory errors is shown in Table 29-132.

Table 29-132. FR_EERCR[CODE] CHI LRAM syndrome coding

FR_EERCR[CODE]	Description
0x00	No Error (Never occurs in error report registers)
0x01–0x1F	Non-Corrected Error

29.6.24.3 Memory Error Response

The memory error response is enabled only when the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) is set.

In case of the detection of a *corrected* memory error, the FlexRay module continues its normal operation using the corrected data word. This section describes the behavior of the FlexRay module after the detection of a *non-corrected* memory error.

29.6.24.3.1 CHI LRAM Error Response after CC Read

When the CC is out of the *POC:default config* state, it reads the configuration data and the data field offsets of all utilized message buffers in every slot and in the NIT. If a non-corrected memory error is detected during this module read access, the error response of the module depends on the LRAM location where the error occurred.

- If the LRAM address belongs to physical message buffer configuration data, the FlexRay module will consider the affected message buffer as disabled for the current search and will exclude this buffer from the search. The configuration of the affected message buffer is not changed.

If the affected message buffer is a transmit message buffer, no frame will be transmitted from this message buffer in the next slot. If the affected message buffer is a receive message buffer, no frame will be received to this message buffer in the next slot.

- If the LRAM address belongs to the data field offset area and the related physical message buffer is used for Rx or Tx, the first access to the system memory caused by payload read or write yields to the assertion of the FR_CHIERFR[ILSA_EF]. No memory access occurs w.r.t. payload access is performed for the complete frame.

29.6.24.3.2 CHI LRAM Error Response after Application Read

The application can read the content of the CHI LRAM via reading the FR_MBCCFR*n*, FR_MBFIDR*n*, FR_MBIDXR*n*, FR_MBDOR*n*, and FR_LEETR*n* registers. If a non-corrected memory error is detected during this kind of read access, the module indicates the detected memory error, delivers the non-corrected data read, and continues its normal operation.

29.6.24.3.3 PE DRAM Error Response after CC Read

If the CC detects a non-corrected memory error during internal read of program data contained in PE DRAM, this is considered a fatal protocol error and the module enters the protocol freeze state immediately.

29.6.24.3.4 PE DRAM Error Response after Application Read in *POC:default config* state

If the CC detects a non-corrected memory error during an application-triggered read from any PE DRAM address and the protocol is in the *POC:default config* state, this is considered a fatal protocol error and the module enters the protocol freeze state. This behavior allows for checking the freeze functionality in case of the detection of non-corrected errors.

29.6.24.3.5 PE DRAM Error Response after Application Read out of *POC:default config*

If the CC detects a non-corrected memory error during an application-triggered read from any PE DRAM address, and the protocol is not in the *POC:default config* state, this error is not considered a fatal error and the protocol state is not changed. This prevents any interference of the running protocol by PE DRAM error injection reads.

29.6.25 Memory Error Injection

The error injection functionality is used by the application to inject data errors into the memories to trigger and check the memory error detection functionality.

The error injection is enabled only if the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) and the error injection enable control bit EIE in the ECC Error Report and Injection Control Register (FR_EERICR) are set.

The error injection mode is configured by the EIM configuration bit in the ECC Error Report and Injection Control Register (FR_EERICR).When the error injection is enabled, each write access to the configured memory location will be distorted.

The injector has the same behavior for FlexRay module memory writes and application memory writes.

29.6.25.1 CHI LRAM Error Injection

The following sequence describes a memory error injection sequence for the CHI LRAM memory. This sequence consists of the setup of the error injector followed by an application-triggered write access to provoke a distortion of the memory content. The content of the CHI LRAM is described in Table 29-127.

When the CC is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see Section 29.7.3, Memory Error Injection out of POC:default config.

Injector Setup:

1. FR_MCR[ECCE]: = 1;
   // Enable ECC functionality
2. FR_EERICE[EIE]: = I_MODE;
   // Configure error injection mode
3. FR_EEIAR[MID]: = 1;
   // Select CHI LRAM for error injection
4. FR_EEIAR[BANK]: = I_BANK;
   // Select bank for error injection; I_BANK = {0,1,2,3,4,5}
5. FR_EEIAR[ADDR]: = I_ADDR;
   // Select address for error injection; I_ADDR ≤ 0x2C
6. FR_EEIDR[DATA]: = D_DIST;
   // Define data distortion pattern
7. FR_EEICR[CODE]: = C_DIST;
   // Define checkbit distortion pattern
8. FR_EERICE[EIE]: = 1;
   // Enable error injection

Application Write Access:

If (I_BANK==0) → FR_MBCCFR(2*n*) / FR_MBDOR(6*k*) / FR_LEETR0 := DATA;
If (I_BANK==1) → FR_MBFIDR(2*n*) / FR_MBDOR(6*k* + 1) / FR_LEETR1 := DATA;
If (I_BANK==2) → FR_MBIDXR(2*n*) / FR_MBDOR(6*k* + 2) / FR_LEETR2 := DATA;
If (I_BANK==3) → FR_MBCCFR(2*n* + 1) / FR_MBDOR(6*k* + 3) / FR_LEETR3 := DATA;
If (I_BANK==4) → FR_MBFIDR(2*n* + 1) / FR_MBDOR(6*k* + 4) / FR_LEETR4 := DATA;
If (I_BANK==5) → FR_MBIDXR(2*n* + 1) / FR_MBDOR(6*k* + 5) / FR_LEETR5 := DATA;
// Write DATA to the defined injection bank and injection address (see Table 29-127).

29.6.25.2 PE DRAM Error Injection

The following sequence describes a memory error injection sequence for the PE DRAM memory. This sequence consists of the setup of the error injector followed by an application-triggered write access to provoke a distortion of the memory content.

When the FlexRay module is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see Section 29.7.3.2, PE DRAM Error Injection out of POC:default config.

Injector Setup:

1. FR_MCR[ECCE]: = 1;
   // Enable ECC functionality
2. FR_EERICE[EIE]: = I_MODE;
   // Configure error injection mode
3. FR_EEIAR[MID]: = 0;
   // Select PE DRAM for error injection
4. FR_EEIAR[BANK]: = I_BANK;
   // Define bank for error injection; I_BANK = {0,1}
5. FR_EEIAR[ADDR]: = I_ADDR;
   // Define address for error injection; I_ADDR ≤ 0x7F
6. FR_EEIDR[DATA]: = D_DIST;
   // Define data distortion pattern
7. FR_EEICR[CODE]: = C_DIST;
   // Define checkbit distortion pattern
8. FR_EERICE[EIE]: = 1;
   // Enable error injection

Application Write Access (for example, I_ADDR = 0x70):

1. FR_PEDRAR: = 0x30E0;
   // INST = 0x3; ADDR = 0x70
2. wait until FR_PEDRAR[DAD] == 1;
   // Wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; |
   // Get read back PE DRAM data

Note: The write access to the PE DRAM triggers a subsequent read access from PE DRAM in the next cycle, which triggers the detection of the distorted data.

29.7 Application Information

29.7.1 Module Configuration

This section describes essential parts of the module configuration.

29.7.1.1 Configure System Memory Access Time-Out Register (FR_SYMATOR)

To ensure reliable operation of the CC, the application must ensure that the TIMEOUT value in System Memory Access Time-Out Register (FR_SYMATOR) and the CHI clock frequency f_{CHI} in MHz fulfill Equation 29-29[1].

$$0 \leq \text{SYMATOR[TIMEOUT]} \leq [0.45 \times f_{CHI} - 8] \qquad \textit{Eqn. 29-29}$$

For a given SYMATOR[TIMEOUT] value, f_{CHI} can be increased without causing unreliable operation of the CC. The same holds for reducing the SYMATOR[TIMEOUT] value for a given f_{CHI}.

Some examples for maximum values of the SYMATOR[TIMEOUT] for a minimum CHI frequency are given in Table 29-133.

Table 29-133. Maximum SYMATOR[TIMEOUT] examples

f_{CHI}	SYMATOR[TIMEOUT]	f_{CHI}	SYMATOR[TIMEOUT]
≥ 18 MHz	0	≥ 100 MHz	≤ 37
≥ 23 MHz	≤ 2	≥ 120 MHz	≤ 46
≥ 27 MHz	≤ 4	≥ 140 MHz	≤ 55
≥ 32 MHz	≤ 6	≥ 160 MHz	≤ 64
≥ 60 MHz	≤ 19	≥ 180 MHz	≤ 73
≥ 80 MHz	≤ 28	≥ 200 MHz	≤ 82

29.7.1.1.1 System Bus Wait State Constraints

The SYMATOR[TIMEOUT] value corresponds directly to a certain acceptable number of wait states on the system bus.

For single channel configurations and if the sync frame table generation functionality is *not* used (FR_SFTCCSR[SDVEN,SIDEN] = 0), no timeout will be detected if fewer than 2 × SYMATOR[TIMEOUT] + 1 wait states will be seen on the system bus for each system bus access.

For dual channel configurations, or if the sync frame table generation functionality is used, no timeout will be detected if fewer than SYMATOR[TIMEOUT] – 1 wait states will be seen on the system bus for each system bus access.

29.7.1.2 Configure Data Field Offsets

The data field offsets are located in the Message Buffer Data Field Offset Registers (FR_MBDORn) and Receive FIFO Start Data Offset Register (FR_RFSDOR). The application has to configure the data field offset values for all message buffers that are used.

The reset value of all data field offsets FR_MBDOR*n*[MBDO] and FR_RFSDOR[SDO] is 0. This value is considered to be illegal (see Section 29.6.19.1.1, System Bus Illegal Address Access).

29.7.2 Initialization Sequence

This section describes the required steps to initialize the CC. The first subsection describes the steps required after a system reset, and the second section describes the steps required after preceding shutdown of the CC.

1. See Section 29.3, Controller host interface clocking, for all constraints of minimum CHI clock frequency.

29.7.2.1 Module Initialization

This section describes the module-related initialization steps after a system reset.

1. Configure CC.
   a) Configure the control bits in the Module Configuration Register (FR_MCR).
   b) Configure the system memory base address in System Memory Base Address Register (FR_SYMBADR).
2. Enable the CC.
   a) Write 1 to the module enable bit MEN in the Module Configuration Register (FR_MCR).

The CC now enters the Normal Mode. The application can commence with the protocol initialization described in Section 29.7.2.2, Protocol Initialization.

29.7.2.2 Protocol Initialization

This section describes the protocol-related initialization steps.

1. Configure the protocol engine.
   a) Issue CONFIG command via Protocol Operation Control Register (FR_POCR).
   b) Wait for *POC:config* in Protocol Status Register 0 (FR_PSR0).
   c) Configure the FR_PCR0,..., FR_PCR30 registers to set all protocol parameters.
2. Configure the Message Buffers and FIFOs.
   a) Set the number of message buffers used and the message buffer segmentation in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).
   b) Define the message buffer data size in the Message Buffer Data Size Register (FR_MBDSR).
   c) Configure each message buffer by setting the configuration values in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn), Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn), Message Buffer Frame ID Registers (FR_MBFIDRn), Message Buffer Index Registers (FR_MBIDXRn).
   d) Configure the FIFOs.
   e) Issue CONFIG_COMPLETE command via Protocol Operation Control Register (FR_POCR).
   f) Wait for *POC:ready* in Protocol Status Register 0 (FR_PSR0).

After this sequence, the CC is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

29.7.2.3 CHI LRAM Initialization

The initialization of the CHI LRAM is performed by the CC when it leaves the Disabled Mode. The unitization runs for 45 CHI clock cycles. All fields in the FR_MBCCSR*n*, FR_MBCCFR*n*, FR_MBFIDR*n*, FR_MBDOR*n*, and LEETR*n* registers are initialized to 0. All application read or write accesses to these registers are delayed until the initialization is finished.

29.7.2.4 PE DRAM Initialization

The PE DRAM initialization is performed by the CC in the *POC:default config* state. This initialization runs for 4.8 µs, and will delay the state transition from *POC:default config* into *POC:config*.

29.7.3 Memory Error Injection out of *POC:default config*

This section provides information for application-driven memory error injection out of *POC:default config.* The CC provides means to inject memory errors from the application without any impact to the internal protocol operation of the CC.

29.7.3.1 CHI LRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the LRAM ECC Error Test Registers (FR_LEETRn). Any memory errors injected into these CHI LRAM locations will never be detected by internal access, independent from the protocol state.

The application should use these registers and related CHI LRAM locations to inject memory errors into the CHI LRAM. The injection sequence is described in Section 29.6.25.1, CHI LRAM Error Injection.

29.7.3.2 PE DRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the PE DRAM address 0x70. This is the only PE DRAM address writable by the application out of the *POC:default config* state.

The application should use these PE DRAM location to inject memory errors into the PE DRAM. The injection sequence is described in Section 29.6.25.2, PE DRAM Error Injection.

29.7.4 Shutdown Sequence

This section describes a secure shutdown sequence to stop the CC gracefully. The main targets of this sequence are:

- Finish all ongoing reception and transmission.
- Do not corrupt the FlexRay bus and do not disturb ongoing FlexRay bus communication.

For a graceful shutdown, the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) Repeatedly write 1 to FR_MBCCSR*n*[EDT] until FR_MBCCSR*n*[EDS] == 0.
2. Stop the protocol engine.
   a) Issue HALT command via Protocol Operation Control Register (FR_POCR).
   b) Wait for *POC:halt* in Protocol Status Register 0 (FR_PSR0).

29.7.5 Number of Usable Message Buffers

This section describes the required minimum CHI clock frequency for a specified number of utilized message buffers configured in the Message Buffer Segment Size and Utilization Register

(FR_MBSSUTR), a configured minislot length *gdMinislot*, and a configured nominal macrotick length *gdMacrotick*[1].

Additional constraints for the minimum CHI clock frequency are given in Section 29.3, Controller host interface clocking.

The CC uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search is started at the start of slot and must be finished before the start of the next slot.

The shortest FlexRay slot is a corrected empty dynamic slot. A corrected empty dynamic slot is a minislot and consists of *gdMinislot* corrected macroticks with a duration of *gdMacrotick*. The minimum duration of a corrected macrotick is $gdMacrotick_{min}$ = 39 µµT. This results in a minimum length of a correct slot.

$$\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot \qquad \textbf{Eqn. 29-30}$$

The message buffer search engine runs on the CHI clock and evaluates one individual message buffer per CHI clock cycle. For internal status update operations and to account for clock domain crossing jitter, an additional amount of 27 CHI clock cycles is required to ensure correct search engine operation.

For a given number of utilized message buffers FR_MBSSUTR[LAST_MB_UTIL] + 1 and for a given CHI clock frequency f_{chi}, this results in a search duration of

$$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\text{FR\_MBSSUTR[LAST\_MB\_UTIL]}+27) \qquad \textbf{Eqn. 29-31}$$

The message buffer search must be finished within one slot, which requires that Equation 29-32 must be fulfilled:

$$\Delta_{search} \leq \Delta_{slotmin} \qquad \textbf{Eqn. 29-32}$$

This results in the formula given in Equation 29-33, which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

$$f_{chi} \geq \frac{(\text{FR\_MBSSUTR[LAST\_MB\_UTIL]}+27)}{39 \cdot pdMicrotick \cdot gdMinislot} \qquad \textbf{Eqn. 29-33}$$

The required minimum CHI Clock frequency for a selected set of relevant protocol parameters and for the LAST_MB_UTIL field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR) set to 63 is given in Table 29-134.

Table 29-134. Minimum f_{chi} [MHz] examples (64 message buffers used)

pdMicrotick [ns]	*gdMinislot*					
	2	3	4	5	6	7
25.0	46.7	31.2	23.4	18.7	15.6	13.4
50.0	23.4	15.6	11.7	9.4	7.8	6.7

1. See Section 29.3, Controller host interface clocking, for all constraints of minimum CHI clock frequency.

NOTE

If the minimum CHI frequency is not met then the CHIERFR[MBS_EF] flag is set. Refer to Section 29.5.2.17, CHI Error Flag Register (FR_CHIERFR), for details.

29.7.6 Protocol Control Command Execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of Table 29-16 by writing the command to the POCCMD field of the Protocol Operation Control Register (FR_POCR). As a result the CC sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in Table 29-135. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the CC asserts the illegal protocol command interrupt flag IPC_IF in the Protocol Interrupt Flag Register 1 (FR_PIFR1). The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by Table 29-135. If the application issues the FREEZE or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 29-135. Protocol control command priorities

Protocol command	Priority	Interrupted by	Cleared and terminated by
FREEZE	(highest) 1	none	
READY	2		
CONFIG_COMPLETE	3		

Table 29-135. Protocol control command priorities (continued)

Protocol command	Priority	Interrupted by	Cleared and terminated by
ALL_SLOTS	4	FREEZE, READY, CONFIG_COMPLET, fatal protocol error	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		
RUN	6		FREEZE, fatal protocol error
WAKEUP	7		FREEZE, fatal protocol error
DEFAULT_CONFIG	8		FREEZE, fatal protocol error
CONFIG	9		
HALT	(lowest) 10		FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

29.7.7 Message Buffer Search on Simple Message Buffer Configuration

This section describes the message buffer search behavior for a simplified message buffer configuration. The FIFO behavior is not considered in this section.

29.7.7.1 Simple Message Buffer Configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers: one single-buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has the following configuration:

Table 29-136. Transmit buffer configuration

Register	Field	Value	Description
FR_MBCCSR*t*	MTD	1	Transmit buffer
FR_MBCCFR*t*	MTM	0	Event transition mode
	CHA	1	Assigned to channel A
	CHB	0	Not assigned to channel B
	CCFE	1	Cycle counter filter enabled
	CCFMSK	000011	Cycle set = {4*n*} = {0,4,8,12,...}
	CCFVAL	000000	
FR_MBFIDR*t*	FID	S	Assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit FR_MBCCSRt[CMT] and the lock bit FR_MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and has the following configuration:

Table 29-137. Receive buffer configuration

Register	Field	Value	Description
FR_MBCCSR*r*	MTD	0	Receive buffer
FR_MBCCFR*r*	MTM	—	n/a
	CHA	1	Assigned to channel A
	CHB	0	Not assigned to channel B
	CCFE	1	Cycle counter filter enabled
	CCFMSK	000001	Cycle set = {2*n*} = {0,2,4,6,...}
	CCFVAL	000000	
FR_MBFIDR*r*	FID	S	Subscribed slot

Furthermore the assumption is that both message buffers are enabled (FR_MBCCSR*t*[EDS] = 1 and FR_MBCCSR*r*[EDS] = 1).

NOTE

The cycle set {4*n* + 2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4*n*} = {0,4,8,12,...} is assigned to both buffers.

29.7.7.2 Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel then that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S*. In any case, the result of the message buffer search will be the transmit message buffer *t*. The receive message buffer *r* will not be found, and no reception is possible.

29.7.7.3 Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel then that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

29.7.7.3.1 Transmit Data Not Available

If transmit data is *not available*, that is, the transmit buffer is not committed FR_MBCCSR*t*[CMT]=0 and/or locked FR_MBCCSR*t*[LCKS]=1:

a) For the cycles in the set {4*n*}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) For the cycles in the set {4*n* + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in Figure 29-160.

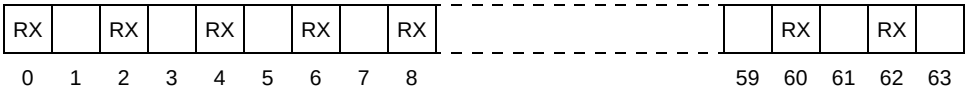


Figure 29-160. Transmit data not available

29.7.7.3.2 Transmit Data Available

If transmit data is *available*, that is, the transmit buffer is committed FR_MBCCSR*t*[CMT]=1 and not locked FR_MBCCSR*t*[LCKS]=0:

a) For the cycles in the set {4*n*}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) For the cycles in the set {4*n* + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in Figure 29-161.

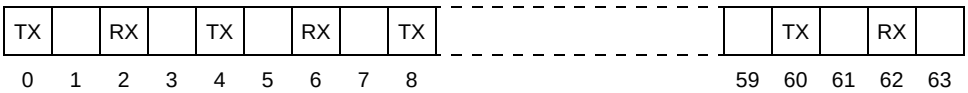


Figure 29-161. Transmit data available

Chapter 30
Frequency-Modulated Phase-Locked Loop (FMPLL)

30.1 Introduction

This section describes the features and functions of the two independent FMPLL modules implemented on the MPC5675K device.

30.2 Overview

MPC5675K has two PLLs: one for the system clock using frequency modulation (FM) and one that can be used for motor control peripherals.

The FMPLLs allow the user to generate high speed system clocks from a common 4–120 MHz input clock. Further, the FMPLLs support programmable frequency modulation of the system clock (although the one for motor control peripherals is intended to be operated without FM). The PLL multiplication factor and the output clock divider ratio are software-configurable.

The FMPLL's block diagram is shown in Figure 30-1.

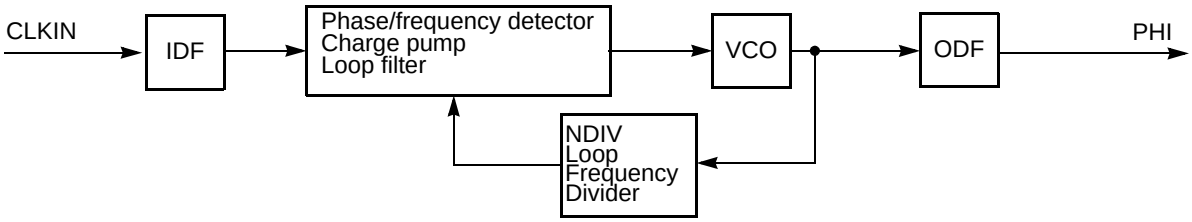


Figure 30-1. FMPLL block diagram

30.3 Features

The FMPLL has the following major features:

- Input clock frequency from 4–120 MHz
- Voltage controlled oscillator (VCO) range from 256–512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the PLL to relock
- Frequency modulated PLL (FMPLL)
  - — Modulation enabled/disabled through software
  - — Triangle wave modulation
- Programmable modulation depth
  - — ±0.25% to ±4% deviation from center spread frequency
  - — –0.5% to +8% deviation from down spread frequency
  - — Programmable modulation frequency dependent on reference frequency
- Progressive clock switching only on FMPLL_0 for the system clock

- Two operating modes
  - — PLL mode with crystal reference (default)
  - — PLL mode with external reference

30.4 Memory map

The FMPLLs are mapped through the MC_CGM register slot. Since the MC_CGM base address is unaffected by Lock Step Mode (LSM) and Decoupled Parallel Mode (DPM), the base addresses for the FMPLL modules are similarly unaffected.

Table 30-1 shows the memory map locations. Addresses are given as offsets of the module base address.

Table 30-1. FMPLL memory map

Offset from FMPLL_BASE FMPLL_0: 0xC3FE_00A0 FMPLL_1: 0xC3FE_00C0	Register	Access[1]	Reset Value[2]	Location
0x0000	Control Register (CR)	R/W	0x0540_0001	on page 1127
0x0004	Modulation Register (MR)	R/W	0x0000_0000	on page 1128
0x0008–0x000F	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

30.5 Register descriptions

The PLL operation is controlled by two registers. These registers can be written only in supervisor mode.

30.5.1 Control Register (CR)

Address: Base + 0x0000

Access: Supervisor read/write; User read-only

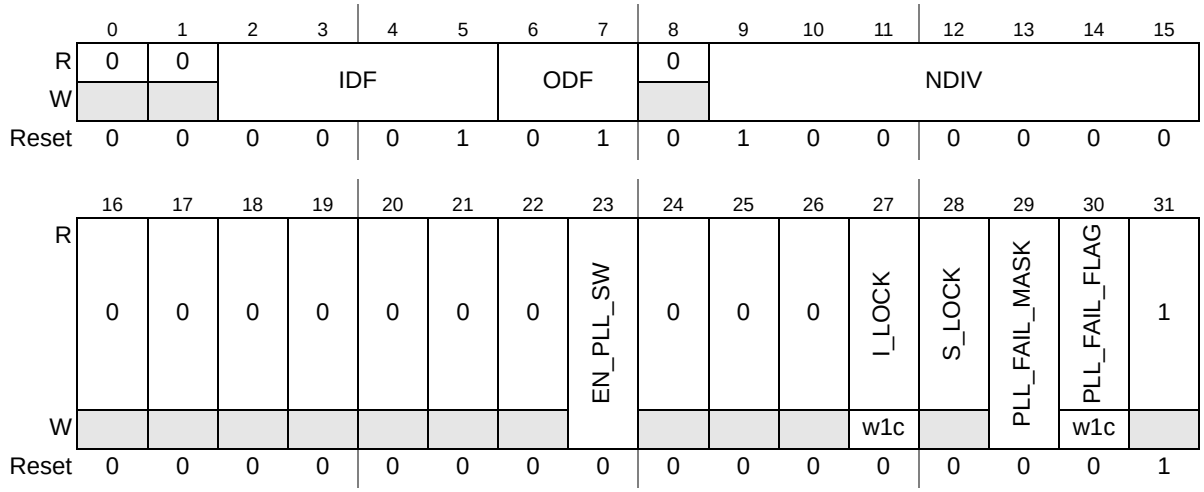


Figure 30-2. Control Register (CR)

Table 30-2. CR field descriptions

Field	Description
IDF	The value of this field sets *idf*, the PLL Input division factor as described in Table 30-3.
ODF	The value of this field sets *odf*, the PLL Output division factor. 00 Divide by 2. 01 Divide by 4. 10 Divide by 8. 11 Divide by 16.
NDIV	The value of this field sets *ldf*, the PLL Loop division factor as described in Table 30-4.
EN_PLL_SW	This bit enables progressive clock switching. 0 Progressive clock switching disabled. 1 Progressive clock switching enabled.
I_LOCK	This bit is set by hardware whenever a lock/unlock event occurs. It is cleared by writing a 1 to this bit.
S_LOCK	This bit is an indication of whether the PLL has acquired lock. 0 PLL unlocked. 1 PLL locked.
PLL_FAIL_MASK	This bit masks the pll_fail output. 0 pll_fail not masked. 1 pll_fail masked.
PLL_FAIL_FLAG	This bit is asynchronously set by hardware whenever a loss of lock event occurs while PLL is switched on. It is cleared by writing a 1 to this bit.

Table 30-3. Input divide ratios

IDF[3:0]	Input division factor	IDF[3:0]	Input division factor
0000	Divide by 1	1000	Divide by 9
0001	Divide by 2	1001	Divide by 10
0010	Divide by 3	1010	Divide by 11
0011	Divide by 4	1011	Divide by 12
0100	Divide by 5	1100	Divide by 13
0101	Divide by 6	1101	Divide by 14
0110	Divide by 7	1110	Divide by 15
0111	Divide by 8	1111	Clock Inhibit

Table 30-4. Loop divide ratios

NDIV[6:0]	Loop divide factor
0000000–0011111	NA
0100000	Divide by 32
0100001	Divide by 33
0100010	Divide by 34
...	...
1011111	Divide by 95
1100000	Divide by 96
1100001–1111111	NA

30.5.2 Modulation Register (MR)

Address: Base + 0x0004

Access: Supervisor read/write; User read-only

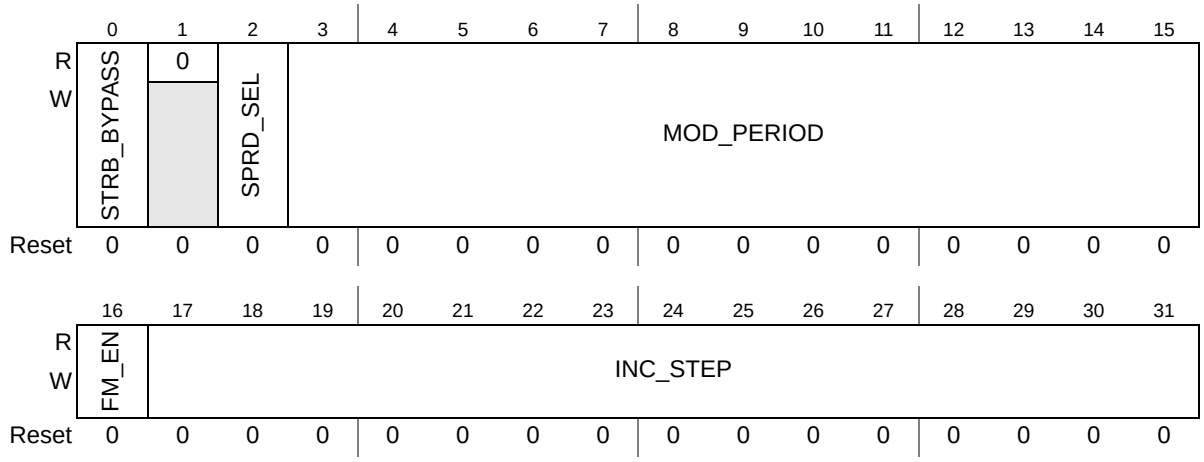


Figure 30-3. Modulation Register (MR)

Table 30-5. MR field descriptions

Field	Description
STRB_BYPASS	Strobe bypass. The STRB_BYPASS signal bypasses the STRB signal used inside PLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0 STRB is used to latch PLL modulation control bits. 1 STRB is bypassed. In this case control bits need to be static. The control bits must be changed only when PLL is in power down mode.
SPRD_SEL	Spread type selection. SPRD_SEL controls the spread type in Frequency Modulation mode. 0 Center SPREAD. 1 Down SPREAD.
MOD_PERIOD	Modulation period. The MOD_PERIOD field is the binary equivalent of the value *modperiod* derived from following formula: $$\text{modperiod} = \frac{f_{ref}}{4 \times f_{mod}}$$ where: *fref:* represents the frequency of the feedback divider *fmod*: represents the modulation frequency
FM_EN	Frequency Modulation Enable. The FM_EN enables the frequency modulation. 0 Frequency Modulation disabled. 1 Frequency Modulation enabled.
INC_STEP	Increment step. The INC_STEP field is the binary equivalent of the value *incstep* derived from following formula: $$\text{incstep} = \text{round}\left(\frac{(2^{15}-1) \times \text{md} \times \text{MDF}}{100 \times 5 \times \text{MODPERIOD}}\right)$$ where: *md*: represents the peak modulation depth in percentage (Center spread = pk-pk = ±md, Downspread = pk-pk = –2 × md) *MDF*: represents the nominal value of loop divider (NDIV in PLL Control Register)

30.6 Functional description

30.6.1 Normal mode

In Normal mode, the PLL inputs are driven by the CR. This means that when the PLL is in lock state, the PLL output clock (PHI) is derived by the reference clock (XOSC) through this relation:

Eqn. 30-1

$$\text{phi} = \frac{\text{xosc} \cdot \text{ldf}}{\text{idf} \cdot \text{odf}}$$

where the value of *idf*, *ldf* and *odf* are set in the Control Register (CR), corresponding to the IDF, NDIV, and ODF bits, respectively; and can be derived from Table 30-3 and Table 30-4, and the value of CR[ODF]. For example, if the reference clock (XOSC) is 8 MHz, CR[NDIV] = 0x1000000, CR[IDF] = 0x0001, CR[ODF] = 0x01, then *phi* = (8 x 64) / (2 x 4) = 64 MHz.

30.6.2 Progressive clock switching

Progressive clock switching allows you to switch the system clock to the PLL output clock, stepping through different division factors. This means that the current consumption gradually increases and so the voltage regulator has a better response.

This feature can be enabled by programming the EN_PLL_SW bit in CR. Then, when the input pin pll_select goes high, the output clock ck_pll_div progressively increases its frequency as shown in Table 30-6.

Table 30-6. Progressive clock switching on pll_select rising edge

Number of cycles	ck_pll_div frequency
8	(ck_pll_out frequency) ÷ 8
16	(ck_pll_out frequency) ÷ 4
32	(ck_pll_out frequency) ÷ 2
33 and higher	ck_pll_out frequency

30.6.3 Normal mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. Note that frequency modulation needs to be disabled on the PLL driving the motor control clocks.

When frequency modulation is enabled, two parameters must be set to generate the desired level of modulation: the PERIOD and the STEP. The modulation waveform is always a triangle wave and its shape is not programmable.

FM modulation is activated using these steps:

1. Configure the FM modulation characteristics: MOD_PERIOD, INC_STEP.
2. Enable the FM modulation by programming the SSCG_EN bit of MR register to 1. FM modulated mode can be enabled only when PLL is in lock state.

To latch these values inside the PLL, two ways are used, depending on the value of STRB_BYPASS register bit in MR.

If STRB_BYPASS is low, the modulation parameters are latched in the PLL only when the STRB signal goes high. The STRB signal is automatically generated in the FMPLL when the modulation is enabled (SSCG_EN goes high) if the PLL is locked (S_LOCK = 1) or when the modulation has been enabled (SSCG_EN = 1) and PLL enters in lock state (S_LOCK goes high).

If STRB_BYPASS is high, the STRB signal is bypassed. In this case, control bits (MOD_PERIOD[12:0], INC_STEP[14:0], SPREAD_CONTROL) must be changed only when the PLL is in power down mode.

The modulation depth in % is

Eqn. 30-2

$$\text{ModulationDepth} = \left(\frac{100 \times 5 \times \text{INCSTEP} \times \text{MODPERIOD}}{(2^{15} - 1) \times \text{MDF}}\right)$$

NOTE

You must ensure that the product of INCTEP and MODPERIOD is less than $(2^{15} - 1)$.

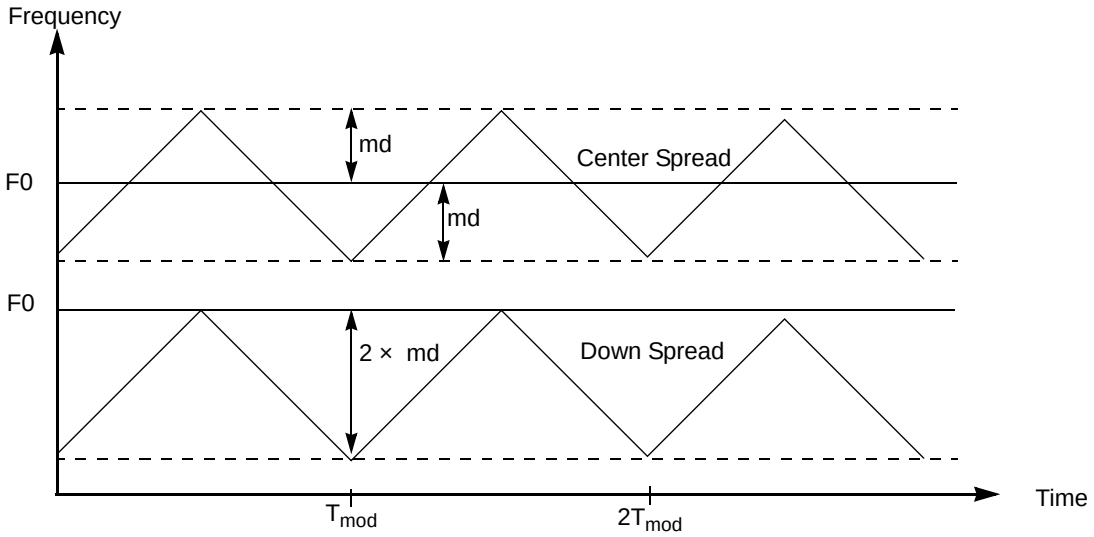


Figure 30-4. Frequency modulation

30.6.4 Power down mode

The PLL can be switched off when not required to achieve lower consumption by programming the ME_*x*_MC registers in the MC_ME module.

30.7 Recommendations

To avoid any unpredictable behavior of the PLL clock, it is recommended to respect the following guidelines:

- The FMPLL VCO frequency should reside in the range 256–512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- The user must change the multiplication and division factors only when the PLL output clock is not selected as the source for an active clock on the chip. The MOD_PERIOD, INC_STEP, and SPREAD_SEL bits should be modified before activating the FM modulated mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set to 1, then MOD_PERIOD, INC_STEP, and SPREAD_SEL can be modified only when PLL is in power-down mode.
- Use progressive clock switching for the system clock.
- XOSC needs to be connected and oscillating before the FMPLL can be turned on. OSC_CTL[EOCV] ensures that the external oscillator clock signal is stable before it can be selected by the system.

  The FMPLL is dependent on XOSC being switched on. This ensures that when the FMPLL is automatically switched off during various mode entries or exits, the FMPLL source is never

switched off before switching off the FMPLL. When the FMPLL is running from the XOSC, the current setting always ensures that FMPLL is switched off before the XOSC is switched off.

Otherwise, software must take care of this sequence. This puts a limitation on the system that XOSC should be toggling even when the FMPLL is running on IRC clock.

NOTE

Progressive clock switch is very fast and the current transition here can be too fast for certain regulators to handle. The following programming and switching will help both internal and external regulators during the start-up of the FMPLL.

1. Program PLL for 180 Mhz with ODF at 3 so PLL output is 22.5 Mhz.

2. Enable PLL. Wait for 200 μs from this point. PLL lock is coarse lock and not fine lock.

3. Shift System clk to PLL output.

4. Change ODF setting from $3 \rightarrow 2 \rightarrow 1 \rightarrow 0$ in 10 μs intervals.

Chapter 31
Inter-Integrated Circuit Bus Controller Module (I^2C)

31.1 Introduction

The I^2C bus is a two-wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communication over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate as fast as 100 kbps with maximum bus loading and timing. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

The MPC5675K provides three functionally identical I^2C modules, identified as I^2C0 through I^2C2.

31.1.1 Features

The I^2C has these major features:

- Compatible with I^2C bus standard
- Multi-master operation
- Software programmable for one of 256 serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Basic DMA interface

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

31.1.2 Block diagram

A simplified block diagram of the I^2C illustrates the functionality and interdependence of major blocks (see Figure 31-1).

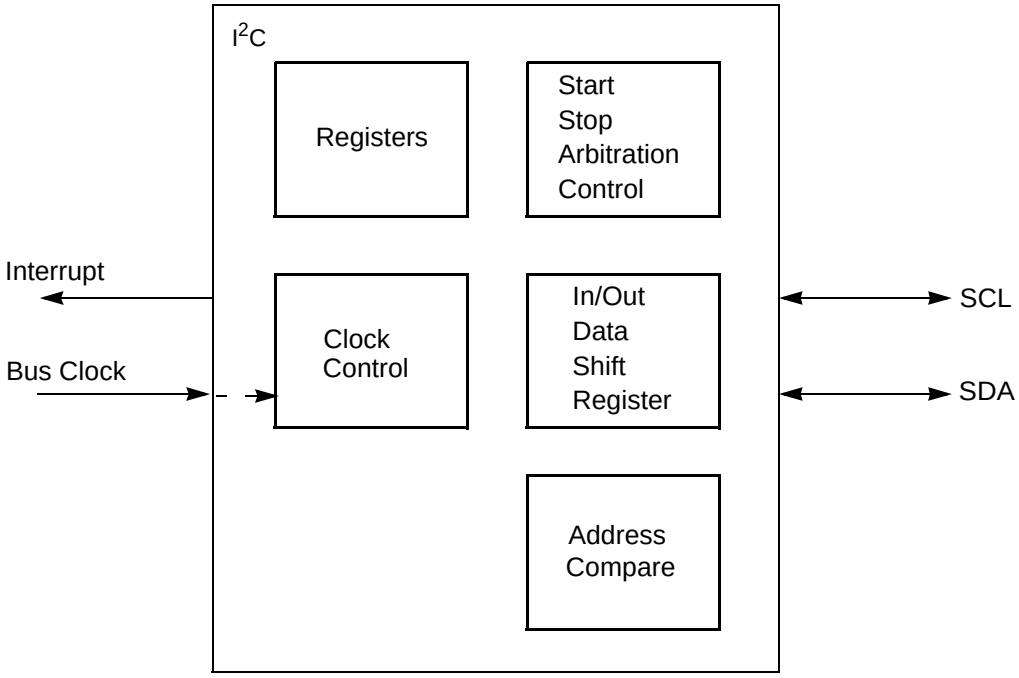


Figure 31-1. I2C block diagram

31.1.3 DMA interface

A simple DMA interface is implemented so that the I2C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting the DMAEN bit in the I2C Bus Control Register (IBCR). DMA requests can be performed on all four I2C channels.

The DMA interface is only valid when the I2C module is configured for master mode and the DMA channel mux has selected the I2C DMA request signals to be inputs to a DMA channel.

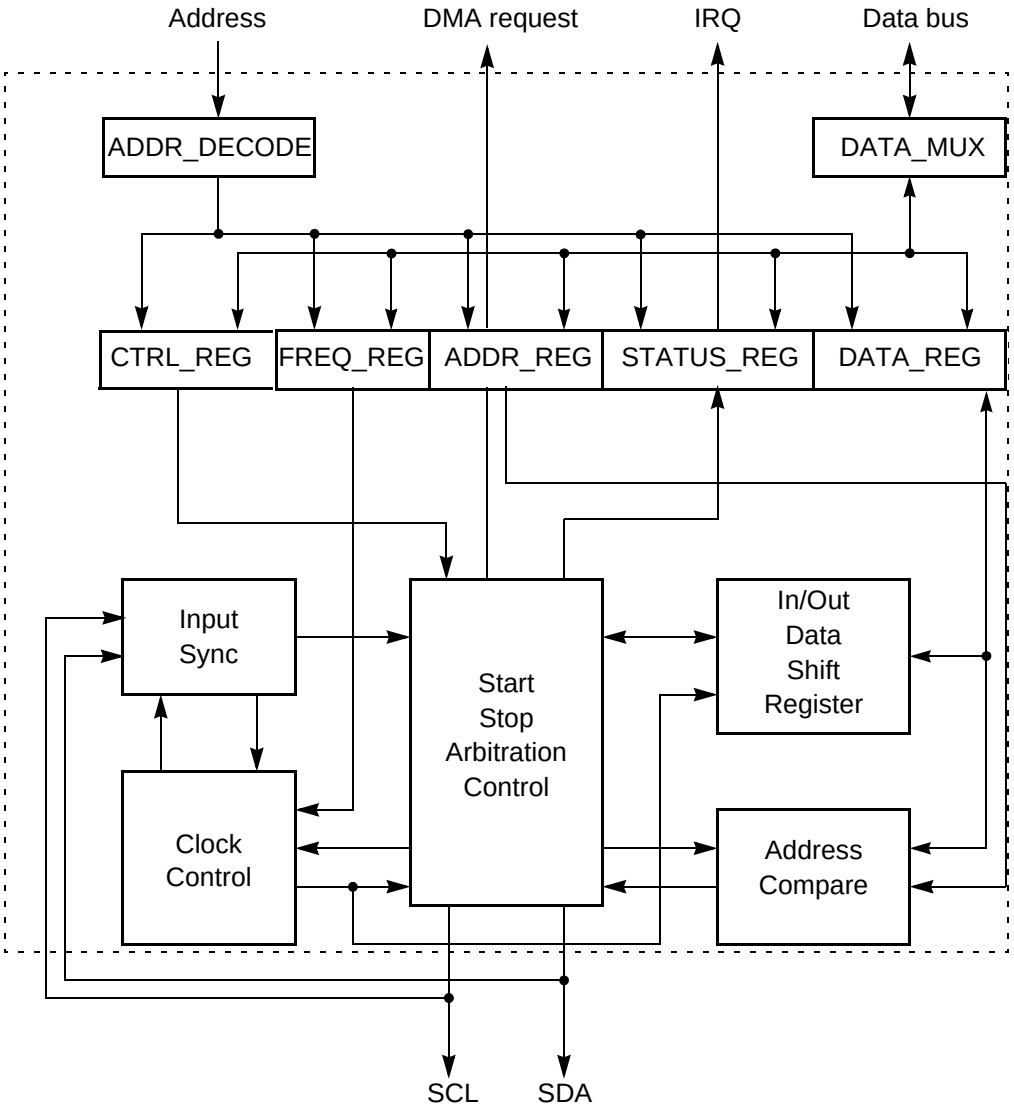


Figure 31-2. I2C module DMA interface block diagram

31.1.4 Modes of operation

The I2C module has two operating modes: run mode and halt mode. In run mode, I2C_*x* = 0 in the SIU_HLT0 register and all functional parts of the I2C module are running. In halt mode, I2C_*x* = 1 in the SIU_HLT0 register and all clocks to the I2C module are disabled.

31.1.4.1 Debug mode

When the MCU is in debug mode, the I2C behavior is unaffected and remains dictated by the mode of the I2C.

31.2 External signal description

Refer to Chapter 3, Signal Description, for detailed signal descriptions.

31.3 Memory map and registers

This section provides a detailed description of all I2C registers.

31.3.1 Module memory map

Table 31-1 shows the base addresses for the I2C modules. These addresses are not affected by Lock Step Mode (LSM) or Decoupled Parallel Mode (DPM). Table 31-2 shows the I2C memory map.

Table 31-1. I2C module base addresses

Mode	Module	Module base address
LSM and DPM	I2C_0	0xFFE3_0000
	I2C_1	0xFFE4_4000
	I2C_2	0xFFE4_8000

Table 31-2. I2C memory map

Offset from I2C_BASE	Register	Access [1]	Reset Value [2]	Section/Page
0x0000	I2C Bus Address Register (IBAD)	R/W	0x00	on page 1136
0x0001	I2C Bus Frequency Divider Register (IBFD)	R/W	0x00	on page 1137
0x0002	I2C Bus Control Register (IBCR)	R/W	0x80	on page 1140
0x0003	I2C Bus Status Register (IBSR)	R/W	0x80	on page 1141
0x0004	I2C Bus Data I/O Register (IBDR)	R/W	0x00	on page 1142
0x0005	I2C Bus Interrupt Configuration Register (IBIC)	R/W	0x00	on page 1143
0x0006–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

31.3.2 Register descriptions

This section lists the I2C registers in address order and describes the registers and their bit fields.

31.3.2.1 I2C Bus Address Register (IBAD)

This register contains the address the I2C bus responds to when addressed as a slave; it is not the address sent on the bus during the address transfer.

Address: Base + 0x0000 Access: User read/write

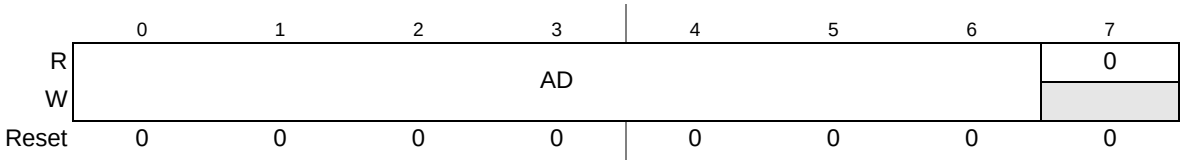


Figure 31-3. I^2C Bus Address Register (IBAD)

Table 31-3. IBAD field descriptions

Field	Description
AD	Slave Address. Specific slave address to be used by the I^2C bus module. **Note:** The default mode of I^2C bus is slave mode for an address match on the bus.

31.3.2.2 I^2C Bus Frequency Divider Register (IBFD)

Address: Base + 0x0001 Access: User read/write

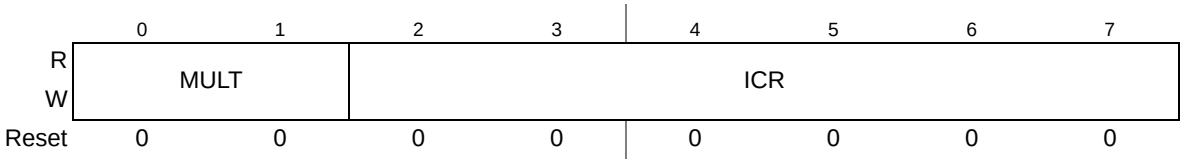


Figure 31-4. I^2C Bus Frequency Divider Register (IBFD)

Table 31-4. IBFD field descriptions

Field	Description
MULT	I^2C Multiplier Factor. The MULT bits define the multiplier factor *mul*. This factor is used along with the SCL divider to generate the I^2C baud rate. The multiplier factor *mul* as defined by the MULT bits is provided below. 00 *mul* = 1 01 *mul* = 2 10 *mul* = 4 11 Reserved

Table 31-4. IBFD field descriptions (continued)

Field	Description
ICR	I^2C Bus Clock Rate. The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the I^2C baud rate, the SDA hold time, the SCL Start hold time, and the SCL Stop hold time. Table 31-5 provides the SCL divider and hold values for corresponding values of the ICR. The SCL divider multiplied by multiplier factor *mul* is used to generate I^2C baud rate. **I^2C baud rate = bus speed (Hz)/(*mul* × SCL divider)** ***Eqn. 31-1*** SDA hold time is the delay from the falling edge of SDA (I^2C data) to the changing of SDA (I^2C data). **SDA hold time = bus period (s) × *mul* × SDA hold value** ***Eqn. 31-2*** SCL Start hold time is the delay from the falling edge of SDA (I^2C data) while SCL is high (Start condition) to the falling edge of SCL (I^2C clock). **SCL Start hold time = bus period (s) × *mul* × SCL Start hold value** ***Eqn. 31-3*** SCL Stop hold time is the delay from the rising edge of SCL (I^2C clock) to the rising edge of SDA SDA (I^2C data) while SCL is high (Stop condition). **SCL Stop hold time = bus period (s) × *mul* × SCL Stop hold value** ***Eqn. 31-4***

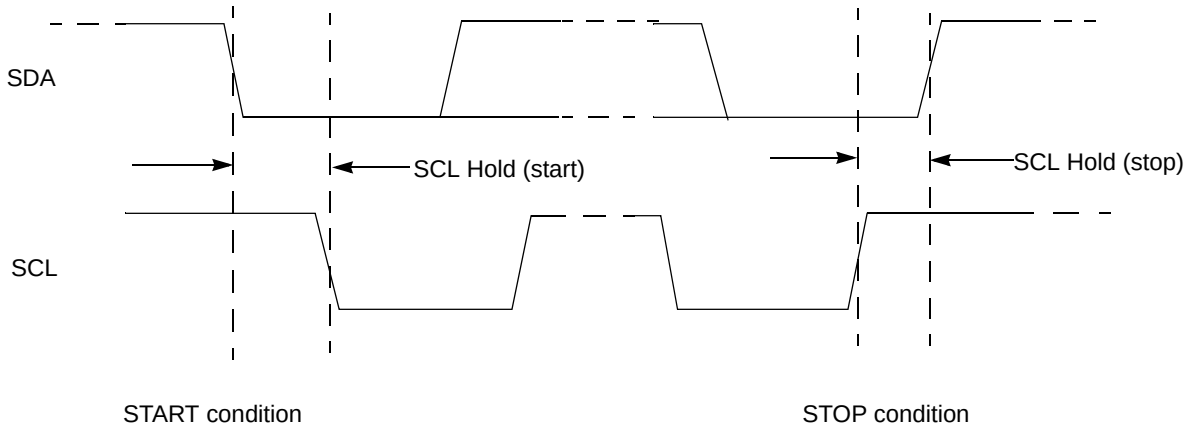


Figure 31-5. SCL divider and SDA hold

Table 31-5. I^2C divider and hold values

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SDA hold (stop) value	ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL hold (stop) value
00	20	7	6	11	20	160	17	78	81
01	22	7	7	12	21	192	17	94	97
02	24	8	8	13	22	224	33	110	113
03	26	8	9	14	23	256	33	126	129
04	28	9	10	15	24	288	49	142	145
05	30	9	11	16	25	320	49	158	161
06	34	10	13	18	26	384	65	190	193
07	40	10	16	21	27	480	65	238	241
08	28	7	10	15	28	320	33	158	161
09	32	7	12	17	29	384	33	190	193
0A	36	9	14	19	2A	448	65	222	225
0B	40	9	16	21	2B	512	65	254	257
0C	44	11	18	23	2C	576	97	286	289
0D	48	11	20	25	2D	640	97	318	321
0E	56	13	24	29	2E	768	129	382	385
0F	68	13	30	35	2F	960	129	478	481
10	48	9	18	25	30	640	65	318	321
11	56	9	22	29	31	768	65	382	385
12	64	13	26	33	32	896	129	446	449
13	72	13	30	37	33	1024	129	510	513
14	80	17	34	41	34	1152	193	574	577
15	88	17	38	45	35	1280	193	638	641
16	104	21	46	53	36	1536	257	766	769
17	128	21	58	65	37	1920	257	958	961
18	80	9	38	41	38	1280	129	638	641
19	96	9	46	49	39	1536	129	766	769
1A	112	17	54	57	3A	1792	257	894	897
1B	128	17	62	65	3B	2048	257	1022	1025
1C	144	25	70	73	3C	2304	385	1150	1153
1D	160	25	78	81	3D	2560	385	1278	1281
1E	192	33	94	97	3E	3072	513	1534	1537
1F	240	33	118	121	3F	3840	513	1918	1921

31.3.2.3 I^2C Bus Control Register (IBCR)

Address: Base + 0x0002 Access: User read/write

	0	1	2	3	4	5	6	7
R	MDIS	IBIE	MS	TX	NOACK	0	DMAEN	0
W						RSTA		
Reset	1	0	0	0	0	0	0	0

Figure 31-6. I^2C Bus Control Register (IBCR)

Table 31-6. IBCR field descriptions

Field	Description
MDIS	Module Disable. This bit controls the software reset of the entire I^2C bus module. 0 The I^2C bus module is enabled. This bit must be cleared before any other IBCR bits have any effect. 1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can be accessed. **Note:** If the I^2C bus module is enabled in the middle of a byte transfer, the interface behaves as follows: Slave mode ignores the current transfer on the bus and starts operating when a subsequent start condition is detected. Master mode is not aware that the bus is busy. Therefore, if a start cycle is initiated, the current bus cycle may become corrupt. This ultimately results in the current bus master or the I^2C bus module losing arbitration, after which bus operation returns to normal.
IBIE	I-Bus Interrupt Enable. 0 Interrupts from the I^2C bus module are disabled. This does not clear any currently pending interrupt condition. 1 Interrupts from the I^2C bus module are enabled. An I^2C bus interrupt occurs provided the IBIF bit in the status register is also set.
MS	Master/Slave Mode Select. This bit is cleared on reset. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated if only the IBIF flag is set. MS is cleared without generating a STOP signal when the master loses arbitration. 0 Slave mode. 1 Master mode.
TX	Transmit/Receive Mode Select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. 0 Receive. 1 Transmit.
NOACK	Data Acknowledge Disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I^2C module always acknowledges address matches, provided it is enabled, regardless of the value of NOACK. Values written to this bit are used only when the I^2C Bus is a receiver, not a transmitter. 0 An acknowledge signal is sent out to the bus at the 9th clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1).

Table 31-6. IBCR field descriptions (continued)

Field	Description
RSTA	Repeat Start. Writing a 1 to this bit generates a repeated START condition on the bus, provided it is the current bus master. This bit is always read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, results in loss of arbitration. 0 No effect. 1 Generate repeat start cycle.
DMAEN	DMA enable. When this bit is set, the DMA TX and RX lines are asserted when the I^2C module requires data to be read or written to the data register. No transfer done interrupts are generated when this bit is set; however, an interrupt is generated if loss of arbitration or addressed as slave conditions occur. The DMA mode is valid only when the I^2C module is configured as a master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA application information section for more details. 0 Disable the DMA TX/RX request signals. 1 Enable the DMA TX/RX request signals.

31.3.2.4 I^2C Bus Status Register (IBSR)

Address: Base + 0x0003 Access: User read/write

	0	1	2	3	4	5	6	7
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W							w1c	
Reset	1	0	0	0	0	0	0	0

Figure 31-7. I^2C Bus Status Register (IBSR)

Table 31-7. IBSR field descriptions

Field	Description
TCF	Transfer Complete. While one byte of data is transferred, this bit is cleared. It is set by the falling edge of the ninth clock of a byte transfer. This bit is valid only during or immediately following a transfer to the I^2C module or from the I^2C module. 0 Transfer in progress. 1 Transfer complete.
IAAS	Addressed as a Slave. When its own specific address (I-bus address register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU must check the SRW bit and set its Tx/$\overline{Rx}$ mode accordingly. Writing to the I-bus control register clears this bit. 0 Not addressed. 1 Addressed as a slave.
IBB	Bus Busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 0 Bus is idle. 1 Bus is busy.

Table 31-7. IBSR field descriptions (continued)

Field	Description
IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: • SDA is sampled low when the master drives a high during an address or data transmit cycle. • SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it. This bit must be cleared by software, by writing a one to it. A write of zero has no effect.
SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is valid only when the I-bus is in slave mode, a complete address transfer has occurred with an address match, and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: • Arbitration lost (IBAL bit set) • Byte transfer complete (TCF bit set and DMAEN bit not set) • Addressed as slave (IAAS bit set) • NoAck from slave (MS and TX bits set) • I2C bus going idle (IBB high-low transition and enabled by BIIE) A processor interrupt request is generated if the IBIE bit is set. This bit must be cleared by software, by writing a 1 to it. A write of 0 has no effect on this bit. In DMA mode (DMAEN set), a byte transfer complete condition does not trigger the setting of IBIF. All other conditions apply.
RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. 0 Acknowledge received. 1 No acknowledge received.

31.3.2.5 I2C Bus Data I/O Register (IBDR)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7
R W	Data							
Reset	0	0	0	0	0	0	0	0

Figure 31-8. I2C Bus Data I/O Register (IBDR)

In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. The TX bit in the IBCR must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin.

For instance, if the I^2C is configured for master transmit but a master receive is desired, then reading the IBDR does not initiate the receive.

Reading the IBDR returns the last byte received while the I^2C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I^2C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of MS is used for the address transfer and should comprise the calling address (in position D0–D6) concatenated with the required R/$\overline{W}$ bit (in position D7).

31.3.2.6 I^2C Bus Interrupt Configuration Register (IBIC)

Address: Base + 0x0005 Access: User read/write

	0	1	2	3	4	5	6	7
R	BIIE	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-9. I^2C Bus Interrupt Configuration Register (IBIC)

Table 31-8. IBIC field descriptions

Field	Description
BIIE	Bus Idle Interrupt Enable Bit. This configuration bit can be used to enable the generation of an interrupt after the I^2C bus becomes idle. After this bit is set, an IBB high-low transition sets the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I^2C bus. 0 Bus idle interrupts disabled. 1 Bus idle interrupts enabled.

31.4 Functional description

31.4.1 I-Bus protocol

The I^2C bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open-drain or open-collector outputs. A logical AND function is exercised on both lines with external pullup resistors. The value of these resistors is system-dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. They are described briefly in the following sections and illustrated in Figure 31-10.

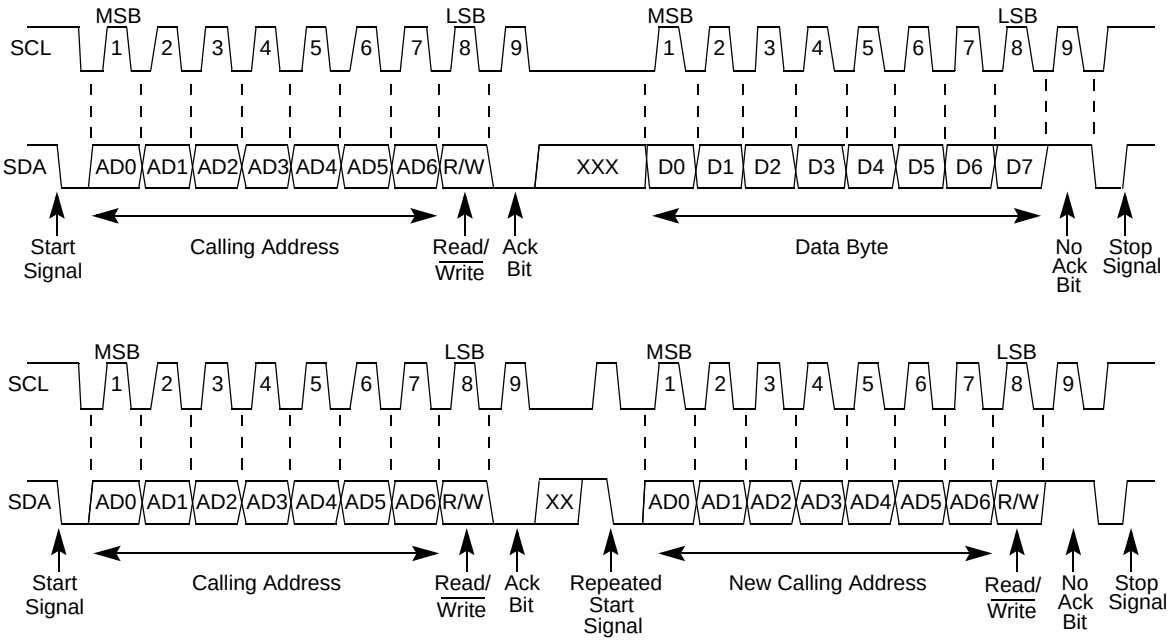


Figure 31-10. I2C bus transmission signals

31.4.1.1 START signal

When the bus is free, i.e.,no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 31-10, a START signal is a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

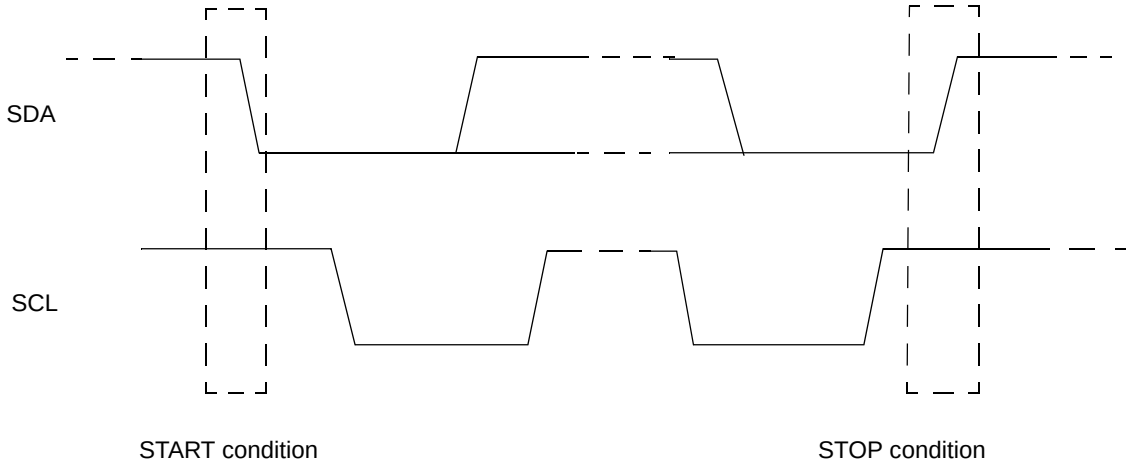


Figure 31-11. Start and Stop conditions

31.4.1.2 Slave address transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer—the slave transmits data to the master

0 = Write transfer—the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the ninth clock (see Figure 31-10).

No two slaves in the system may have the same address. If the I^2C bus is master, it must not transmit an address that is equal to its own slave address. The I^2C bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the I^2C bus reverts to slave mode and operates correctly, even if it is being addressed by another master.

31.4.1.3 Data transfer

After successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high (see Figure 31-10). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signaled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end of data to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

31.4.1.4 STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical 1 (see Figure 31-10).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

31.4.1.5 Repeated START signal

As shown in Figure 31-10, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

31.4.1.6 Arbitration procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus simultaneously, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic “1” while another master transmits logic 0. The losing masters immediately switch to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

31.4.1.7 Clock synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock remains within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see Figure 31-12). When all engaged devices have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

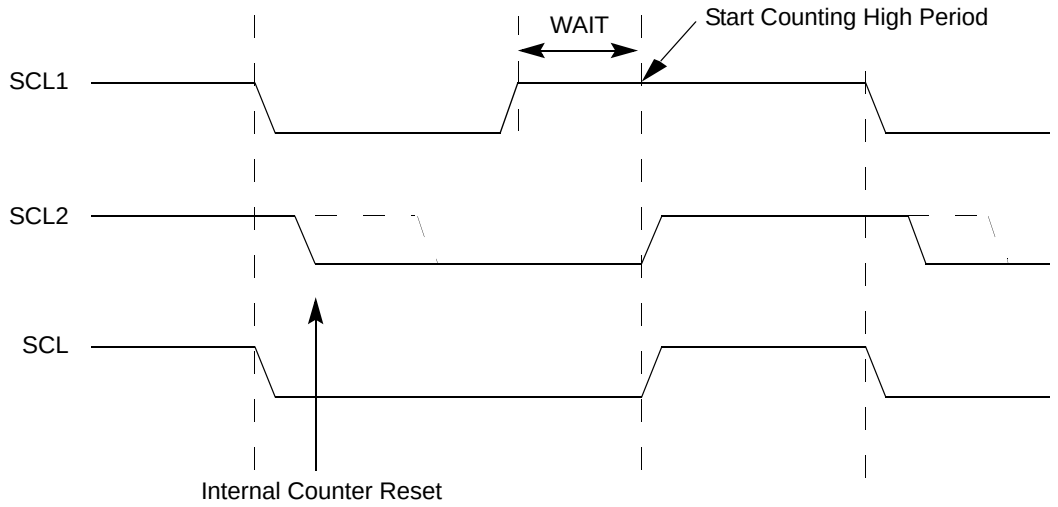


Figure 31-12. I^2C bus clock synchronization

31.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

31.4.1.9 Clock stretching

The clock synchronization mechanism can be used by slaves to slow the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period, then the resulting SCL bus signal low period is stretched.

31.4.2 Interrupts

31.4.2.1 General

The I^2C uses one interrupt vector only.

Table 31-9. Interrupt summary

Interrupt	Offset	Vector	Priority	Source	Description
I^2C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any IBAL, TCF, or IAAS bits are set then an interrupt may be caused based on arbitration lost, transfer complete, or address detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

31.4.2.2 Interrupt description

The I^2C has five types of internal interrupts. The interrupt service routine can determine the interrupt type by reading the status register.

I^2C Interrupt can be generated on:

- Arbitration lost condition (IBAL bit set)
- Byte transfer condition (TCF bit set and DMAEN bit not set)
- Address detect condition (IAAS bit set)
- No acknowledge from slave received when expected
- Bus going idle (IBB bit not set)

The I^2C interrupt is enabled by the IBIE bit in the I^2C control register. It must be cleared by writing 1 to the IBIF bit in the interrupt service routine. The bus going idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

31.5 Initialization/application information

31.5.1 I2C programming examples

31.5.1.1 Initialization sequence

Reset puts the I2C bus control register in its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the frequency divider register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I2C bus address register (IBAD) to define its slave address.
3. Clear the MDIS bit of the I2C bus control register (IBCR) to enable the I2C interface system.
4. Modify the bits of the IBCR to select master/slave mode, transmit/receive mode, and interrupt enable or not. Optionally modify the bits of the I2C bus interrupt configuration register (IBIC) to further refine the interrupt behavior.
5. Configure the SDA and SCL pads. (The SIU Pad Configuration registers must be configured to select the appropriate I2C function. Also, the open drain feature of the pad must be enabled by setting the ODE bit in the appropriate SIU pad configuration register.)

31.5.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. If the device is connected to a multi-master bus system, the state of the I2C bus busy bit (IBB) must be tested to check if the serial bus is free.

If the bus is free (IBB = 0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I2C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events that generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 2, IBSR ==1)                    // wait in loop for IBB flag to clear
bit3 and bit 2, IBCR = 1// set transmit and master mode, i.e. generate start condition
IBDR = calling_address          // send the calling address to the data register
while (bit 2, IBSR ==0)                   // wait in loop for IBB flag to be set
```

31.5.1.3 Post-transfer software response

Transmission or reception of a byte sets the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I2C Bus interrupt bit (IBIF) is set also; an interrupt is generated if the

interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit is cleared to indicate data transfer in progress by reading the IBDR data register in receive mode or writing the IBDR in transmit mode. The TCF bit must not be used as a data transfer complete flag because the flag timing depends on a number of factors including the I^2C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. Transfer complete situations must be detected using the IBIF flag

Software may service the I^2C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Polling should monitor the IBIF bit rather than the TCF bit because their operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode, i.e., the address is transmitted. If master receive mode is required, indicated by R/W bit in IBDR, then the TX bit should be toggled at this stage.

During slave mode address cycles (IAAS = 1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the TX bit is programmed accordingly. For slave mode data cycles (IAAS = 0) the SRW bit is not valid. The TX bit in the control register should be read to determine the direction of the current transfer.

The following is an example software sequence for master transmitter in the interrupt routine.

```
clear bit 6, IBSR                                    // Clear the IBIF flag
if (bit 2, IBCR ==0)
        slave_mode()                                 // run slave mode routine
if (bit 3, IBCR ==0))
        receive_mode()                              // run receive_mode routine
if (bit 7, IBSR == 1)                                          // if NO ACK
     end();                                                 // end transmission
else
IBDR = data_to_transmit                            // transmit next byte of data
```

31.5.1.4 Generation of STOP

A data transfer ends with a STOP signal generated by the master device. A master transmitter can generate a STOP signal after all the data has been transmitted. The following example shows how a stop condition is generated by a master transmitter.

```
if (tx_count == 0) or     // check to see if all data bytes have been transmitted
    (bit 7, IBSR == 1) {                                 // or if no ACK generated
         clear bit 2, IBCR                              // generate stop condition
        }
else {
IBDR = data_to_transmit                    // write byte of data to DATA register
         tx_count --                                              // decrement counter
      }                                                    // return from interrupt
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data. This can be done by setting the transmit acknowledge bit (TXAK) before reading the next-to-last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following example shows how a STOP signal is generated by a master receiver.

```
rx_count --                                          // decrease the rx counter
if (rx_count ==1)                                  // 2nd last byte to be read ?
   bit 4, IBCR = 1                                                // disable ACK
if (rx_count == 0)                                  // last byte to be read ?
  bit 6, IBCR = 0                                     // generate stop signal
else
data_received = IBDR                                 // read RX data and store
```

31.5.1.5 Generation of repeated START

At the end of data transfer, if the master wants to remain communicating on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 5, IBCR = 1                              // generate another start ( restart)
IBDR == calling_address                           // transmit the calling address
```

31.5.1.6 Slave mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) must be tested to check if a calling of its own address has been received. If IAAS is set, software sets the transmit/receive mode select bit (TX bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. IAAS is read as set when it is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers have IAAS cleared. A data transfer may be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave drives SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so the master can generate a STOP signal.

31.5.1.7 Arbitration lost

If several masters try to engage the bus simultaneously, one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL remains generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL = 1 and MS = 0. If one master attempts to start transmission while the bus is being engaged by another master, the hardware inhibits the transmission, switches the MS bit from 1 to 0 without generating a STOP condition, generates an interrupt to CPU, and sets the IBAL to indicate that the attempt to engage the bus has failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

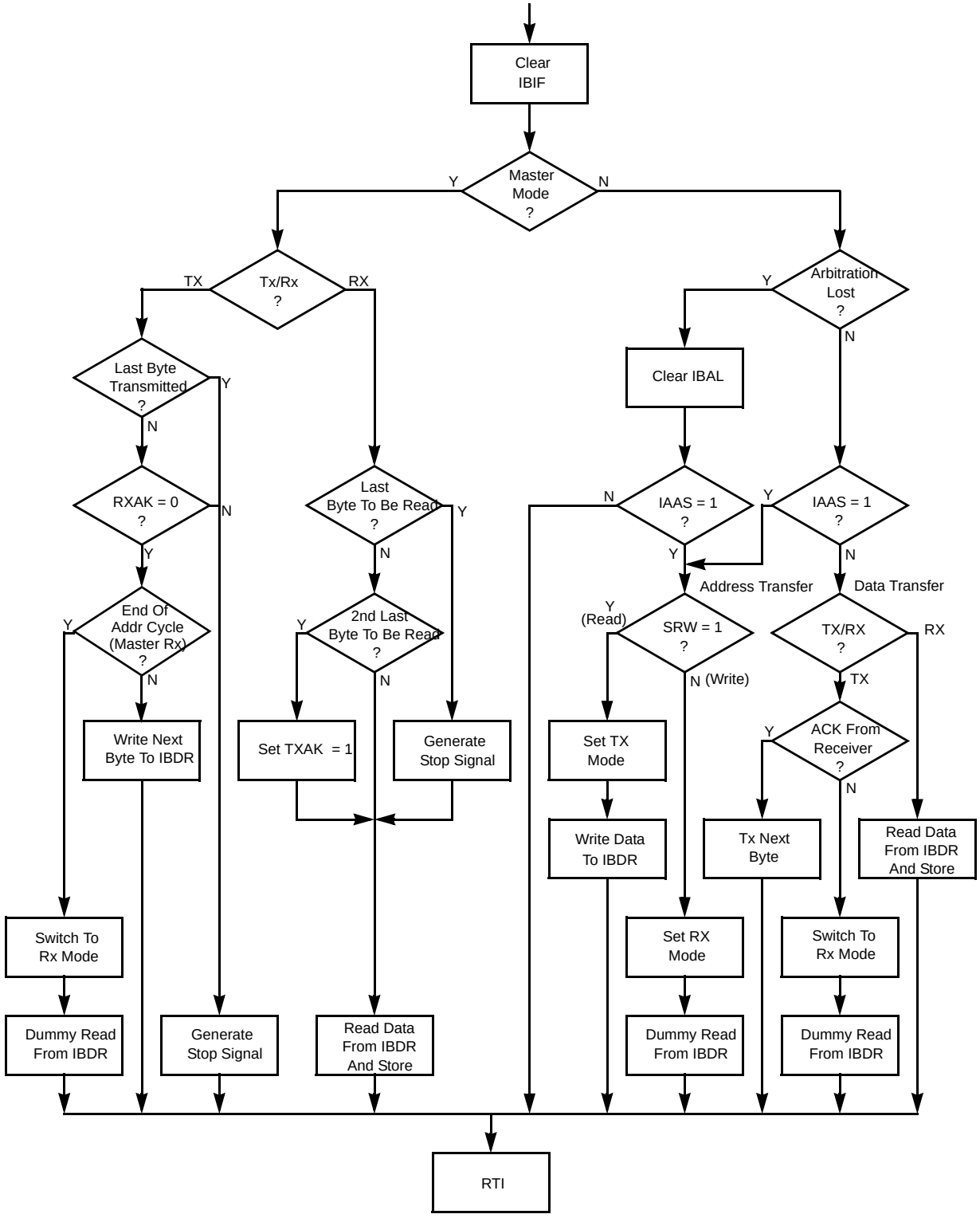


Figure 31-13. Flowchart of typical I^2C interrupt routine

31.5.2 DMA application information

The DMA interface on the I^2C is not completely autonomous and requires intervention from the CPU to start and to terminate the frame transfer. DMA mode is valid for master transmit and master receive modes only. Software must ensure that the DMA enable bit in the control register is not set when the I^2C module is configured in master mode.

The DMA controller must transfer only one byte of data per Tx/Rx request. This is because there is no FIFO on the I^2C block.

The CPU should also keep the I^2C interrupt enabled during a DMA transfer to detect the arbitration lost condition and take action to recover from this situation. The DMAEN bit in the IBCR register works as a disable for the transfer complete interrupt. This means that during normal transfers (no errors) there always is either an interrupt or a request to the DMA controller, depending on the setting of the DMAEN bit. All error conditions trigger an interrupt and require CPU intervention. The address match condition does not occur in DMA mode as the I^2C should never be configured for slave operation.

The following sections detail how to set up a DMA transfer and what intervention is required from the CPU. It is assumed that the system DMA controller is capable of generating an interrupt after a certain number of DMA transfers have taken place.

31.5.2.1 DMA mode, master transmit

Figure 31-14 details exactly the operation for using a DMA controller to transmit *n* data bytes to a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the last data byte) can be transferred by the DMA controller. The last data byte must be transferred by the CPU.

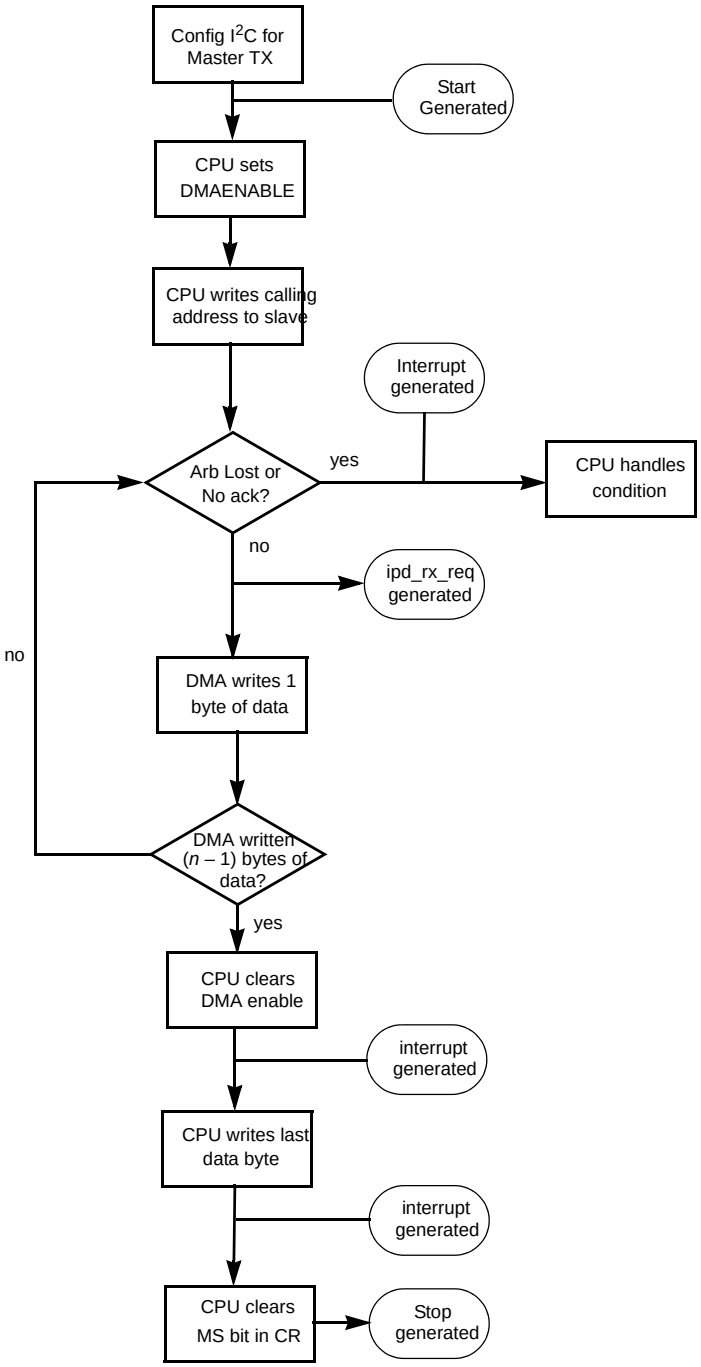


Figure 31-14. DMA mode, master transmit

31.5.2.2 DMA mode, master receive

Figure 31-15 details the exact operation for using a DMA controller to receive *n* data bytes from a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the two last data bytes) can be read by the DMA controller. The last two data bytes must be transferred by the CPU.

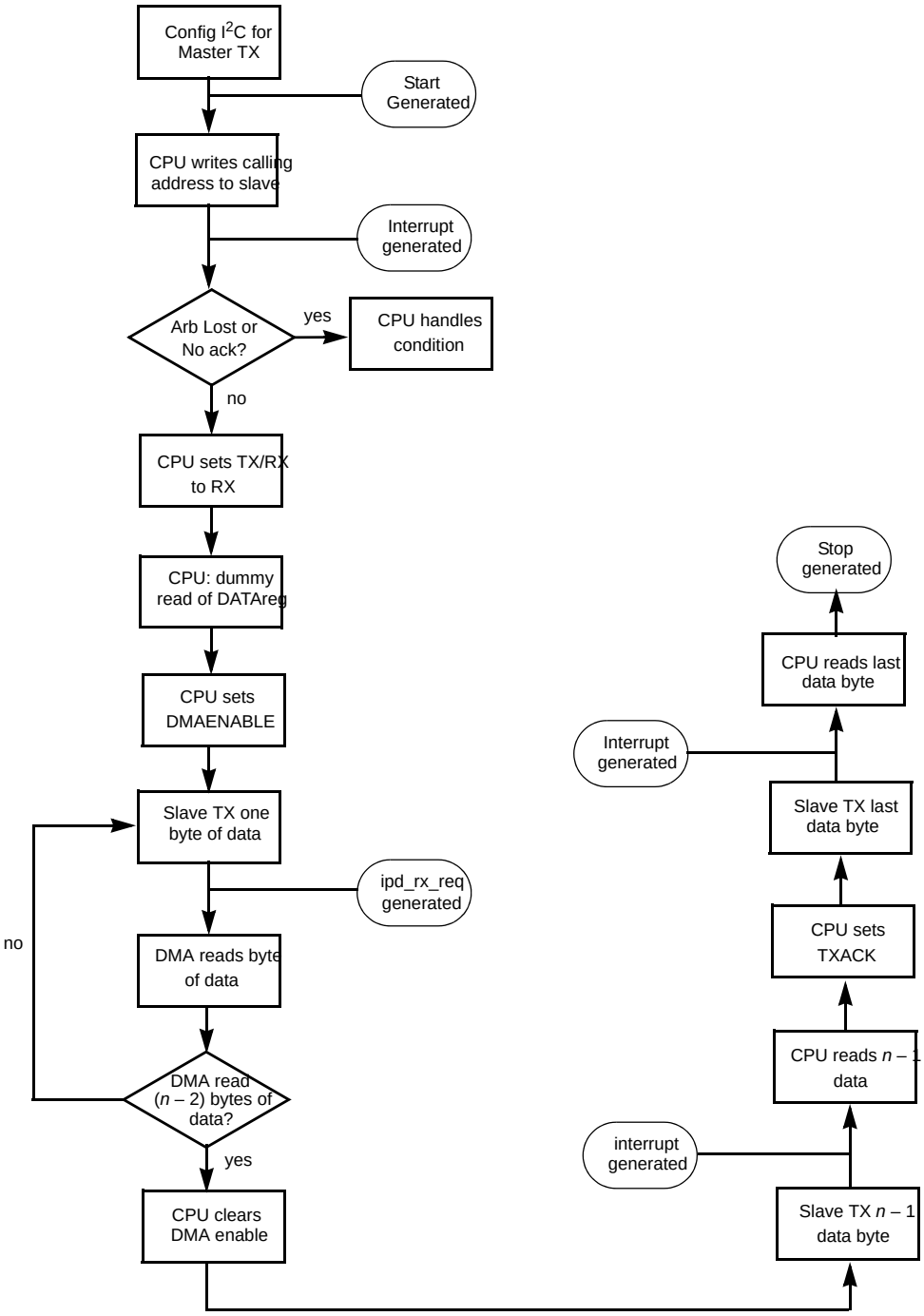


Figure 31-15. DMA mode, master receive

31.5.2.3 Exiting DMA mode, system requirement considerations

As described above, the final transfers of both Tx and Rx transfers need to be managed via interrupt by the CPU. To change from DMA to interrupt driven transfers in the I2C module, disable the DMAEN bit in the IBCR register. The trigger to exit the DMA mode is that the programmed DMA transfer control descriptor (TCD) has completed all its transfers to/from the I2C module.

After the last DMA write (TX mode) to the I^2C the module immediately starts the next I^2C-bus transfer. The same is true for RX mode. After the DMA read from the IBDR register the module initiates the next I^2C-bus transfer. This results in two possible scenarios in the DMA mode exiting scheme.

1. Fast reaction

   The DMAEN bit is cleared before the next I^2C-bus transfer completes. In this case, the module raises an interrupt request to the CPU that can be serviced normally.

2. Slow reaction

   The DMAEN bit is cleared after the next I^2C-bus transfer has already completed. In this case, the module does not raise an interrupt request to the CPU. Instead, the TCF bit can be read to determine that the transfer has completed and the module is ready for further transfer.

31.5.2.3.1 Fast vs. slow reaction

The reaction time T_R for the system to disable DMAEN after the last DMA controller access to the I^2C is the time required for one byte transfer over the I^2C. In a fast reaction the disabling has to occur before the ninth bit of the data transfer, which is the ACK bit. So the time available is eight times the SCL period.

$$T_R = 8 \times T_{SCL} \qquad \textit{Eqn. 31-5}$$

In fast mode, with 400 kbit/s, T_{SCL} is 2.5 µs, so T_R is 20 µs.

Depending on the system and DMA controller there are different possibilities for the deassertion of DMAEN. Three options are:

1. CPU intervention via interrupt

   The DMA controller is programmed to signal an interrupt to the CPU, which is then responsible for the deassertion of DMAEN. This scheme is supported by most systems but can result in a slow reaction time if higher priority interrupts interfere. Therefore, the interrupt handling routine can become complicated as it has to check which of the two scenarios happened (check TCF bit) and act accordingly. In case of slow reaction, you can force an interrupt for the I^2C in the interrupt controller to have the further transfer handled by the normal I^2C interrupt routine. The use of nested interrupts can cause problems in this scenario, if the DMA interrupt stalls between the deassertion and the DMAEN bit and the checking of the TCF bit.

2. DMA channel linking (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer is linked to another channel that does a write to the I^2C IBCR register to disable the DMAEN bit. This is probably the fastest system solution, but it uses two DMA channels. On the system level, no higher priority DMA requests must occur between the two linked TCDs because those can result in slow reaction.

3. DMA scatter/gather process (if supported)

   The transfer control descriptor in the DMA controller that performs the data transfer has the scatter-gather feature activated. This feature initiates a reload of another TCD from system RAM after the completion of the first TCD. The new TCD has its start bit already set and immediately starts the required write to the I^2C IBCR register to disable the DMAEN bit. This TCD also has scatter-gather activated and is programmed to reload the initial TCD upon completion, bringing the

system back into a ready-for-I^2C-transfer state. The advantage over the two other solutions is that this does not require CPU intervention or a second DMA channel. This comes at the cost of 64 bytes RAM (two TCDs), some system bus transfer overhead, and a small increase in application code complexity. On the system level, no higher-priority DMA requests must occur during the scatter-gather process, because those can result in a slow reaction.

Example latencies for a 32 MHz system with a full speed 32-bit AHB bus and an I^2C connected via half speed IPI bus:

- Accessing the I^2C from the DMA controller via IPI bus typically requires four cycles (consecutive accesses to the I^2C could be faster):

$$4 \times T_{IPI} = 4 / 16\ MHz = 250\ ns \qquad \textit{Eqn. 31-6}$$

- Reloading a new TCD (8 × 32-bit) via AHB to the DMA controller (scatter/gather process):

$$8 \times T_{AHB} = 8 / 32\ MHz = 250\ ns \qquad \textit{Eqn. 31-7}$$

With the DMA scatter-gather process, the required IBCR access can be done in 0.5 µs, leaving a large margin of 19.5 µs for additional system delays. The slow reaction case can be prevented in this way. The system user must decide which usage model suits the overall requirements best.

Chapter 32
Interrupt Controller (INTC)

32.1 Introduction

32.1.1 Module overview

The interrupt controller (INTC) provides priority-based preemptive scheduling of interrupt requests. This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 337 interrupt requests. It is targeted to work with a Power Architecture Book E processor and automotive powertrain applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

For high priority interrupt requests in these target applications, the time from the assertion of the interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the Priority Ceiling Protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks that share the resource cannot preempt each other.

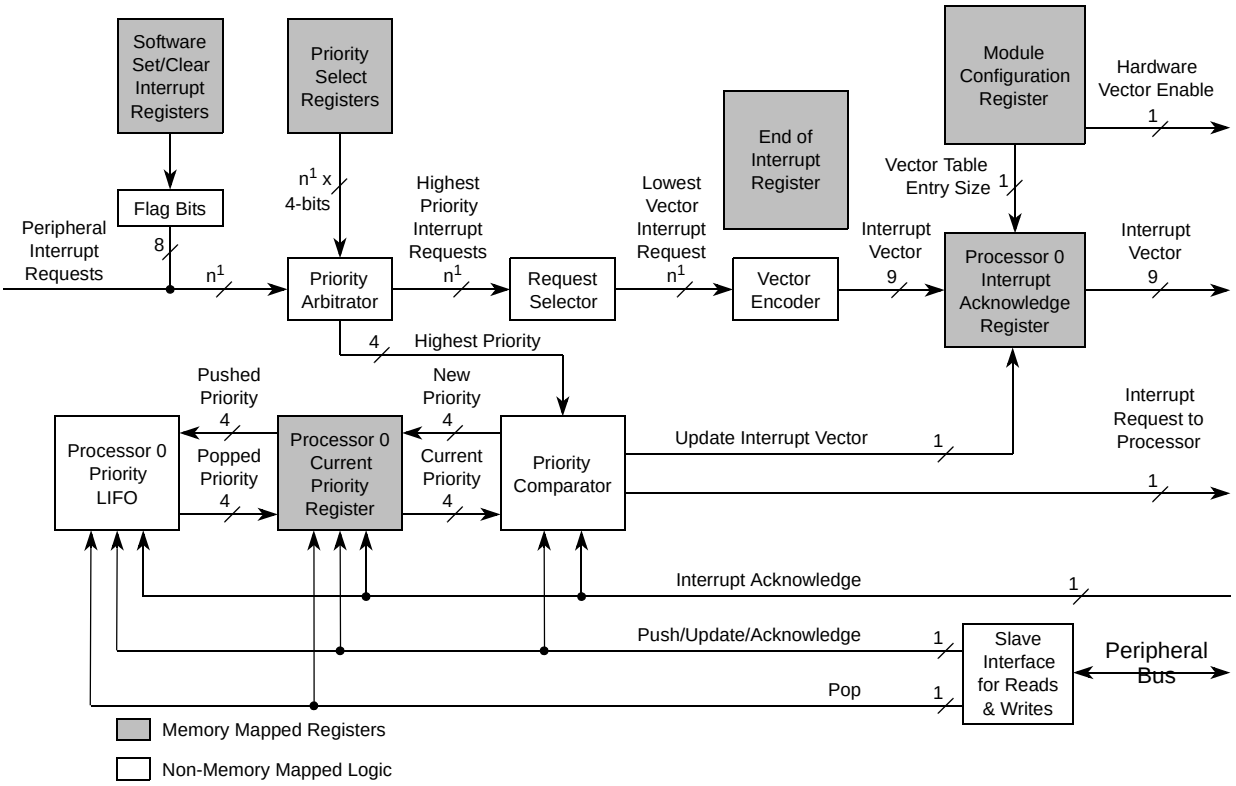
Multiple processors can assert interrupt requests to each other through software-settable interrupt requests. These same software-settable interrupt requests also can be used to split the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software-settable interrupt request to finish the servicing in a lower priority ISR. Therefore these software-settable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

NOTE

In Decoupled Parallel Mode (DPM), two instances of the INTC are available, INTC_0 and INTC_1. In DPM, each instance of the INTC needs to be programmed separately. Thus, all the following references to Core_0 that apply to INTC_0 will also apply to Core_1 when INTC_1 is being programmed.

32.1.2 Block diagram

Figure 32-1 shows a block diagram of the interrupt controller (INTC).



1 The total number of interrupt sources is 337.

Figure 32-1. INTC block diagram

32.1.3 Features

- Supports 329 peripheral interrupt and 8 software-configurable interrupt request sources
- 9-bit vector
  - — Unique vector for each interrupt request source
  - — Hardware connection to processor or read from register
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
  - — Preemptive prioritized interrupt requests to processor
  - — ISR at a higher priority preempts ISRs or tasks at lower priorities
  - — Automatic pushing or popping of preempted priority to or from a LIFO
  - — Ability to modify the ISR or task priority (modifying the priority can be used to implement the Priority Ceiling Protocol for accessing shared resources)
- Low latency—three clocks from receipt of interrupt request from peripheral to interrupt request to processor

32.2 Modes of operation

32.2.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

32.2.1.1 Software vector mode

In software vector mode, software (in this case the interrupt exception handler) must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC uses software vector mode for a given processor when its associated HVEN_PRC0 bit in the INTC_BCR register is negated. The hardware vector enable signal to the processor is driven as negated when its associated HVEN_PRC0 bit is negated. The vector is read from the INTC_ACKR_PRC0 register. Reading the INTC_IACKR_PRC0 negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in the INTC_CPR_PRC0 register onto the associated LIFO and updates PRI in the associated INTC_CPR_PRC0 with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

32.2.1.2 Hardware vector mode

In hardware vector mode, the hardware is the interrupt vector signal from the INTC in conjunction with a processor with the capability to use that vector. In hardware vector mode, this hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore the interrupt exception handler is specific to a peripheral or software-settable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when its associated HVEN_PRC0 bit in the INTC_BCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software-settable interrupt request. The vector value matches the value of the INTVEC_PRC0 field in the INTC_IACKR_PRC0.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC_CPR_PRC0 register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR_PRC0 register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR_PRC0 does not occur when the associated interrupt acknowledge signal asserts and the INTC_SSCIR0_3 or INTC_SSCIR4_7 register is written at a time such that the PRI value in the associated INTC_CPR_PRC0 register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the

associated INTC_CPR_PRC0 is updated with the new priority, and the associated LIFO is neither pushed nor popped.

32.2.2 Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

32.3 Memory map and register description

32.3.1 Memory map

Table 32-1 lists the INTC module base addresses according to mode. Table 32-2 is the INTC memory map.

Table 32-1. INTC module base addresses

Mode	Module	INTC base address
Lock Step Mode (LSM)	INTC_0	0xFFF4_8000
	INTC_1	
Decoupled Parallel Mode (DPM)	INTC_0	0xFFF4_8000 (same as LSM)
	INTC_1	0x8FF4_8000

Table 32-2. INTC memory map

Offset from INTC_BASE	Register	Access [1]	Reset value [2]	Location
0x0000	INTC Block Configuration Register (INTC_BCR)	R/W	0x0000_0000	on page 1161
0x0004	Reserved			
0x0008	INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)	R/W	0x0000_000F	on page 1162
0x000C	Reserved			
0x0010	INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)	R/W	0x0000_0000	on page 1163
0x0014	Reserved			
0x0018	INTC End Of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)	R	0x0000_0000	on page 1164
0x001C	Reserved			
0x0020	INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR0_3)	R/W	0x0000_0000	on page 1165
0x0024	INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR4_7)	R/W	0x0000_0000	on page 1165
0x0028–0x003C	Reserved			

Table 32-2. INTC memory map (continued)

Offset from INTC_BASE	Register	Access [1]	Reset value [2]	Location
0x0040–0x0190 [3]	INTC_PSR0_3—INTC priority select register 0 – 3 to INTC_PSR336 — INTC priority select register 336 [4]	R/W	0x0000_0000	on page 1166
0x0194–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

3 A complete list of address offsets for INTC_PSR is provided in Table 32-8.

4 The PRI fields are 'reserved' for peripheral interrupt requests whose vectors are labeled as Not Used in Table 32-10.

32.3.2 Register information

With exception of the INTC_SSCIR*n* and INTC_PSR*n*, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, misaligned 16 bits to the middle 2 bytes, and aligned 32 bits.

Although INTC_SSCIR*n* and INTC_PSR*n* are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of the INTC_IACKR_PRC0 register are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to the INTC_SSCIR*x_x* register or the INTC_EOIR_PRC0 register does not affect the operation of the write.

32.3.2.1 INTC Block Configuration Register (INTC_BCR)

The INTC Block Configuration Register (INTC_BCR) configures options of the INTC.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES_PRC0	0	0	0	0	HVEN_PRC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-2. INTC Block Configuration Register (INTC_BCR)

Table 32-3. INTC_BCR field descriptions

Field	Description
VTES_PRC0	Vector Table Entry Size. The VTES_PRC0 bit controls the number of '0's to the right of INTVEC_PRC0 in Section 32.3.2.3, INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0). If the contents of INTC_IACKR_PRC0 are used as an address of an entry in a vector table (for example, in software vector mode), then the number of rightmost '0's will determine the size of each vector table entry. This bit does not determine the size of each vector table entry in hardware vector mode but needs to be accounted for if software reads the INTC_IACKR_PRC0 for vector number information. 0 4 bytes. 1 8 bytes.
HVEN_PRC0	Hardware Vector Enable. The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 32.2.1, Normal mode, for the details of the handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

32.3.2.2 INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)

The Current Priority Register masks any peripheral or software-settable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR_PRC0 from generating an interrupt request to Processor 0. When the INTC_IACKR_PRC0 register is read in software vector mode, or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC_SSCIR*x*_*x* registers are written, the LIFO is popped into the INTC_CPR_PRC0's PRI field. An exception case in hardware vector mode to this behavior is described in Section 32.2.1.2, Hardware vector mode.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to Section 32.5.5, Priority ceiling protocol.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 32-3. INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0)

Table 32-4. INTC_CPR_PRC0 field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>PRI[0:3]</td><td>Priority. PRI is the priority of the currently executing ISR according to these values:
<table>
<tr><th>PRI</th><th>Meaning</th><th>PRI</th><th>Meaning</th></tr>
<tr><td>1111</td><td>Priority 15—highest priority</td><td>0111</td><td>Priority 7</td></tr>
<tr><td>1110</td><td>Priority 14</td><td>0110</td><td>Priority 6</td></tr>
<tr><td>1101</td><td>Priority 13</td><td>0101</td><td>Priority 5</td></tr>
<tr><td>1100</td><td>Priority 12</td><td>0100</td><td>Priority 4</td></tr>
<tr><td>1011</td><td>Priority 11</td><td>0011</td><td>Priority 3</td></tr>
<tr><td>1010</td><td>Priority 10</td><td>0010</td><td>Priority 2</td></tr>
<tr><td>1001</td><td>Priority 9</td><td>0001</td><td>Priority 1</td></tr>
<tr><td>1000</td><td>Priority 8</td><td>0000</td><td>Priority 0—lowest priority</td></tr>
</table>
</td></tr>
</table>

NOTE

A store to modify the PRI field that closely precedes or follows an access to a shared resource can result in a non-coherent access to that resource. Refer to Section 32.5.5.2, Ensuring coherency, for example code to ensure coherency.

32.3.2.3 INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)

The INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0) provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

Also, in software vector mode, the INTC_IACKR_PRC0 has side effects from reads. Therefore, it must not be read speculatively while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR_PRC0 does not have side effects in hardware vector mode.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA_PRC0 (most significant 16 bits)															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29[1]	30	31
R	VTBA_PRC0 (least significant 5 bits)				INTVEC_PRC0										0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-4. INTC Interrupt Acknowledge Register for Processor 0 (INTC_IACKR_PRC0)

1 When the INTC_BCR[VTES_PRC0] bit is asserted, INTVEC_PRC0 is shifted to the left one bit. Bit 29 is read as 0. VTBA_PRC0 is narrowed to 20 bits in width.

Table 32-5. INTC_IACKR_PRC0 field descriptions

Field	Description
VTBA_PRC0	Vector Table Base Address for Processor 0. VTBA_PRC0 can be the base address of a vector table of addresses of ISRs for Processor 0. The VTBA_PRC0 only uses the leftmost 20 bits when the VTES_PRC0 bit in INTC_BCR is asserted.
INTVEC_PRC0	Interrupt Vector for Processor 0. INTVEC_PRC0 is the vector of the peripheral or software-settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC0 is updated, whether the INTC is in software or hardware vector mode.

32.3.2.4 INTC End of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	See text															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	See text															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-5. INTC End Of Interrupt Register for Processor 0 (INTC_EOIR_PRC0)

Writing to the INTC End of Interrupt Register for Processor 0 (INTC_EOIR_PRC0) signals the end of the servicing of the interrupt request. When the INTC_EOIR_PRC0 is written, the priority last pushed on the LIFO is popped into the INTC_CPR_PRC0 register. An exception case in hardware vector mode to this behavior is described in Section 32.2.1.2, Hardware vector mode.

Although this is a read/write register, writes are not displayed, and reads are always 0. The values and size of data written to the INTC_EOIR_PRC0 are ignored. Values and sizes written to this register neither update the INTC_EOIR_PRC0 contents nor affect whether the LIFO pops.

To ensure future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC0.

32.3.2.5 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 and INTC_SSCIR4_7)

The INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3 and INTC_SSCIR4_7) support the setting or clearing of software-settable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request. Writing a 1 to SET*x* leaves SET*x* unchanged at 0, but sets the corresponding CLR*x*. Writing a 0 to SET*x* has no effect. CLR*x* is the flag bit. Writing a 1 to CLR*x* clears it. Writing a 0 to CLR*x* has no effect. If a 1 is written to a pair of SET*x* and CLR*x* bits at the same time, CLR*x* is asserted, regardless of whether CLR*x* was asserted before the write.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR 0	0	0	0	0	0	0	0	CLR 1
W							SET 0	w1c							SET 1	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR 2	0	0	0	0	0	0	0	CLR 3
W							SET 2	w1c							SET 3	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-6. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR0_3)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR 4	0	0	0	0	0	0	0	CLR 5
W							SET 4	w1c							SET 5	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR 6	0	0	0	0	0	0	0	CLR 7
W							SET 6	w1c							SET 7	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-7. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR4_7)

Table 32-6. INTC_SSCIR4_7 field descriptions

Field	Description
SET*n*	Set Flag bits. Writing a 1 sets the corresponding CLR***x*** bit. Writing a 0 has no effect. Each SET*x* bit is always read as 0.
CLR*n*	Clear Flag bits. CLRx is the flag bit. Writing a 1 to CLR*x* clears it, provided that a 1 is not written simultaneously to its corresponding SET*x* bit. Writing a 0 to CLR*x* has no effect. 0 An interrupt request is not pending within the INTC. 1 An interrupt request is pending within the INTC.

32.3.2.6 INTC Priority Select Registers (INTC_PSR0_3—INTC_PSR336)

The Priority Select Registers support the selection of an individual priority for each source of interrupt request. The unique vector of each peripheral or software-settable interrupt request determines which INTC_PSR*n* is assigned to that interrupt request. The software-settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0_3 and INTC_PSR4_7, respectively. The peripheral interrupt requests are assigned vectors 8–336, and their priorities are configured in INTC_PSR8 through INTC_PSR336, respectively.

The INTC_PSR4_7 through INTC_PSR328_331 registers follow the same format as shown in Figure 32-8. For address offsets, see Table 32-8.

NOTE

The PRI*x* field of an INTC_PSR*n* must not be modified while its corresponding peripheral or software-settable interrupt request is asserted.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-8. INTC Priority Select Register 0–3 (INTC_PSR0_3)

For address offsets for registers INTC_PSR4_7 through INTC_PSR328_331, see Table 32-8.

Address: Base + 0x018C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI332				0	0	0	0	PRI333			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI334				0	0	0	0	PRI335			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-9. INTC Priority Select Register 332–335 (INTC_PSR332_335)

Address: Base + 0x0190 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI336				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-10. INTC Priority Select Register 336 (INTC_PSR336)

Table 32-7. INTC_PSR0_3—INTC_PSR336 field descriptions

Field	Description
PRI*x*	Priority Select. PRI*x* selects the priority for the interrupt requests. Refer to the field values in Table 32-8.

Table 32-8. INTC Priority Select Register address offsets

INTC_PSR*n*_*n*	Offset Address	INTC_PSR*n*_*n*	Offset Address	INTC_PSR*n*_*n*	Offset Address
INTC_PSR0_3	0x0040	INTC_PSR116_119	0x00B4	INTC_PSR228_231	0x0124
INTC_PSR4_7	0x0044	INTC_PSR120_123	0x00B8	INTC_PSR232_235	0x0128
INTC_PSR8_11	0x0048	INTC_PSR124_127	0x00BC	INTC_PSR236_239	0x012C
INTC_PSR12_15	0x004C	INTC_PSR156_159	0x00DC	INTC_PSR240_243	0x0130
INTC_PSR16_19	0x0050	INTC_PSR160_163	0x00E0	INTC_PSR244_247	0x0134
INTC_PSR20_23	0x0054	INTC_PSR164_167	0x00E4	INTC_PSR248_251	0x0138
INTC_PSR24_27	0x0058	INTC_PSR168_171	0x00E8	INTC_PSR252_255	0x013C
INTC_PSR28_31	0x005C	INTC_PSR140_143	0x00CC	INTC_PSR256_259	0x0140
INTC_PSR32_35	0x0060	INTC_PSR144_147	0x00D0	INTC_PSR260_263	0x0144
INTC_PSR36_39	0x0064	INTC_PSR148_151	0x00D4	INTC_PSR264_267	0x0148
INTC_PSR40_43	0x0068	INTC_PSR152_155	0x00D8	INTC_PSR268_271	0x014C
INTC_PSR44_47	0x006C	INTC_PSR156_159	0x00DC	INTC_PSR272_275	0x0150
INTC_PSR48_51	0x0070	INTC_PSR160_163	0x00E0	INTC_PSR276_279	0x0154
INTC_PSR52_55	0x0074	INTC_PSR164_167	0x00E4	INTC_PSR280_283	0x0158
INTC_PSR56_59	0x0078	INTC_PSR168_171	0x00E8	INTC_PSR284_287	0x015C
INTC_PSR60_63	0x007C	INTC_PSR172_175	0x00EC	INTC_PSR288_291	0x0160
INTC_PSR64_67	0x0080	INTC_PSR176_179	0x00F0	INTC_PSR292_295	0x0164
INTC_PSR68_71	0x0084	INTC_PSR180_183	0x00F4	INTC_PSR296_299	0x0168
INTC_PSR72_75	0x0088	INTC_PSR184_187	0x00F8	INTC_PSR300_303	0x016C
INTC_PSR76_79	0x008C	INTC_PSR188_191	0x00FC	INTC_PSR304_307	0x0170

Table 32-8. INTC Priority Select Register address offsets (continued)

INTC_PSR*n*_*n*	Offset Address	INTC_PSR*n*_*n*	Offset Address	INTC_PSR*n*_*n*	Offset Address
INTC_PSR80-83	0x0090	INTC_PSR192_195	0x0100	INTC_PSR308_311	0x0174
INTC_PSR84_87	0x0094	INTC_PSR196_199	0x0104	INTC_PSR312_315	0x0178
INTC_PSR88_91	0x0098	INTC_PSR200_203	0x0108	INTC_PSR316_319	0x017C
INTC_PSR92_95	0x009C	INTC_PSR204_207	0x010C	INTC_PSR320_323	0x0180
INTC_PSR96_99	0x00A0	INTC_PSR208_211	0x0100	INTC_PSR324_327	0x0184
INTC_PSR100_103	0x00A4	INTC_PSR212_215	0x0114	INTC_PSR328_331	0x0188
INTC_PSR104_107	0x00A8	INTC_PSR216_219	0x0118	INTC_PSR332_335	0x018C
INTC_PSR108_111	0x00AC	INTC_PSR220_223	0x011C	INTC_PSR336	0x0190
INTC_PSR112_115	0x00B0	INTC_PSR224_227	0x0120		

32.4 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor. In addition, spaces in the memory map have been reserved for other possible implementations of the INTC.

32.4.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software-settable. These interrupt requests can assert on any clock cycle.

The INTC has no spurious vector support. Therefore, if an asserted peripheral or software-settable interrupt request whose PRI*x* value in the INTC_PSR*x*_*x* register is higher than the PRI value in the INTC_CPR_PRC0 register negates before the interrupt request to the processor for that peripheral or software-settable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software-settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software-settable interrupt request. Also, the PRI value in the INTC_CPR_PRC0 will be updated with the corresponding PRI*x* value in INTC_PSR*x*_*x*.

Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

32.4.1.1 Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in that peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

32.4.1.2 Software-settable interrupt requests

An interrupt request is triggered by software by writing a 1 to a SET*x* bit in the INTC_SSCIR*x*_*x* register. This write sets the corresponding CLR*x* flag bit, resulting in the interrupt request. The interrupt request is cleared by writing a 1 to the CLR*x* bit.

The time from the write to the SET*x* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

32.4.1.3 Unique vector for each interrupt request source

Each peripheral and software-settable interrupt request is assigned a hardwired unique 9-bit vector. Software-settable interrupts 0–7 are assigned vectors 0–7, respectively. The peripheral interrupt requests are assigned in a range from vector 8 to as high as needed to cover all of the peripheral interrupt requests.

32.4.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in the INTC_PSR*x*_*x* register. The result of that comparison also is compared to PRI*x* in the associated INTC_CPR_PRC0 register. The results of those comparisons are used to manage the priority of the ISR being executed by the associated processor. The associated LIFO also assists in managing that priority.

32.4.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in Figure 32-1 are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software-settable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software-settable interrupt request is generated for the associated INTC_IACKR_PRC0 register, and if in hardware vector mode, for the interrupt vector provided to the processor.

32.4.2.1.1 Priority arbitrator subblock

The priority arbitrator subblock for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software-settable. The output of the priority arbitrator subblock is the highest of those priorities assigned to a given processor. Also, any interrupt

requests that have this highest priority are output as asserted interrupt requests to the associated request selector subblock.

32.4.2.1.2 Request selector subblock

If only one interrupt request from the associated priority arbitrator subblock is asserted, then it is passed as asserted to the associated vector encoder subblock. If multiple interrupt requests from the associated priority arbitrator subblock are asserted, then only the one with the lowest vector is passed as asserted to the associated vector encoder subblock. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software-settable interrupt requests.

32.4.2.1.3 Vector encoder subblock

The vector encoder subblock generates the unique 9-bit vector for the asserted interrupt request from the request selector subblock for the associated processor.

32.4.2.1.4 Priority comparator subblock

The priority comparator subblock compares the highest priority output from the associated priority arbitrator subblock with PRI in the associated INTC_CPR_PRC0. If the priority comparator subblock detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in the associated INTC_CPR_PRC0, or the PRI value in the associated INTC_CPR_PRC0 is lowered below this highest priority. This highest priority then becomes the new priority, which is written to PRI in the associated INTC_CPR_PRC0 when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*x* in INTC_PSR*x*_*x* are zero will not cause a preemption because their PRI*x* will not be higher than PRI in the associated INTC_CPR_PRC0.

32.4.2.2 LIFO

The LIFO stores the preempted PRI values from the associated INTC_CPR_PRC0. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the associated INTC_CPR_PRC0 does not need to be loaded from the associated INTC_CPR_PRC0 and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the associated INTC_CPR_PRC0.

The PRI value in the associated INTC_CPR_PRC0 is pushed onto the LIFO when the associated INTC_IACKR_PRC0 is read in software vector mode or the interrupt acknowledge signal from the associated processor is asserted in hardware vector mode. The priority is popped into PRI in the associated INTC_CPR_PRC0 whenever the associated INTC_EOIR_PRC0 is written. An exception case in hardware vector mode to this behavior is described in Section 32.2.1.2, Hardware vector mode.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of the way that the actions of pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten

priority. However, the LIFO will pop 0s if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory-mapped.

32.4.3 Handshaking with processor

32.4.3.1 Software vector mode handshaking

32.4.3.1.1 Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in Figure 32-11. The INTC examines the peripheral and software-settable interrupt requests. When it finds an asserted peripheral or software-settable interrupt request with a higher priority than PRI in the associated INTC_CPR_PRC0 register, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 register is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in Section 32.2.1.1, Software vector mode.

32.4.3.1.2 End of interrupt exception handler

Before the interrupt exception handling completes, the INTC_EOIR_PRC0 register must be written. When it is written, the associated LIFO is popped so that the preempted priority is restored into PRI of the associated INTC_CPR_PRC0. Before it is written, the peripheral or software-settable flag bit must be cleared so that the peripheral or software-settable interrupt request is negated.

NOTE

A store to clear the peripheral or software-settable interrupt flag bit that closely precedes the store to the INTC_EOIR_PRC0 or INTC_EOIR_PRC1 can result in that peripheral or software-settable interrupt request being serviced again. To prevent this, execute a Power Architecture ISYNC, MSYNC, or MBAR instruction before the store to the INTC_EOIR_PRC0, as shown in Example 32-1.

When returning from the preemption, the INTC does not search for the peripheral or software-settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer even be asserted. When PRI in the associated INTC_CPR_PRC0 is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software-settable interrupt request at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor will return to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

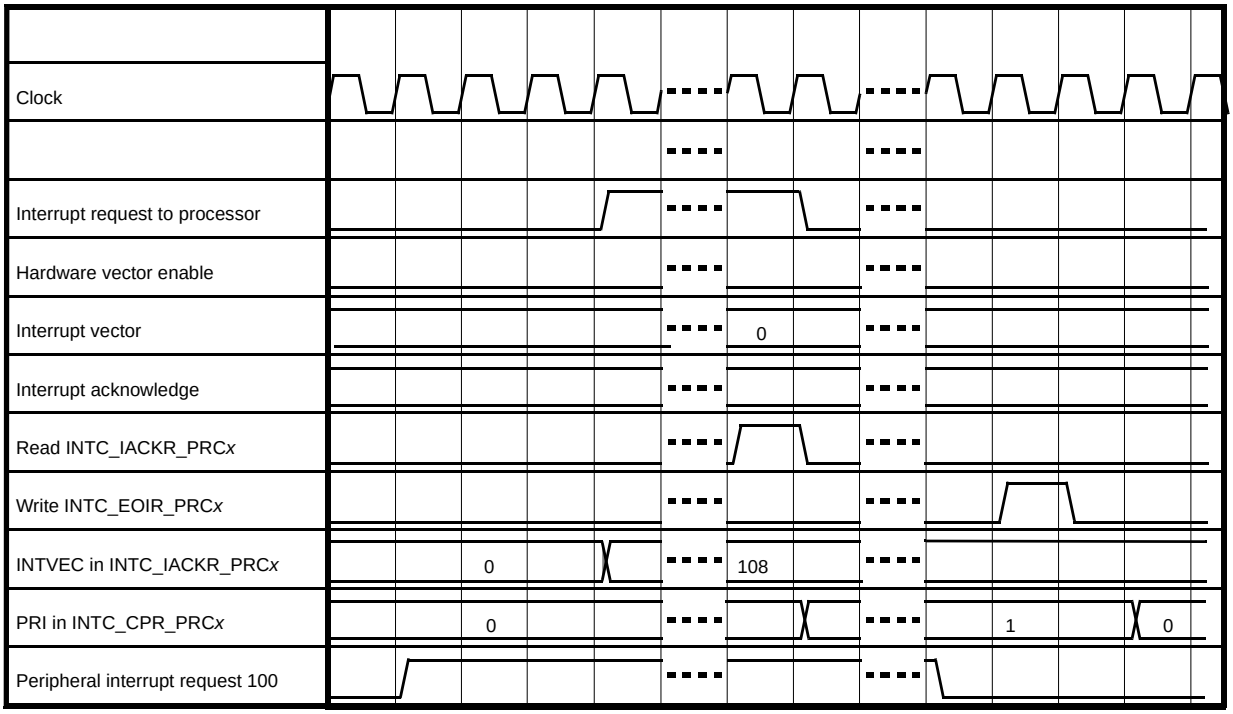


Figure 32-11. Software vector mode handshaking timing diagram

32.4.3.2 Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in Figure 32-12. As in software vector mode, the INTC examines the peripheral and software-settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in the associated INTC_CPR_PRC0, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 is updated with the preempting peripheral or software-settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the associated processor is asserted. In addition, the value of the interrupt vector to the associated processor matches the value of the INTVEC field in the associated INTC_IACKR_PRC0. The rest of the handshaking is described in Section 32.2.1.2, Hardware vector mode.

The handshaking near the end of the interrupt exception handler, that is the writing to the associated INTC_EOIR_PRC0, is the same as in software vector mode. Refer to Section 32.4.3.1.2, End of interrupt exception handler.

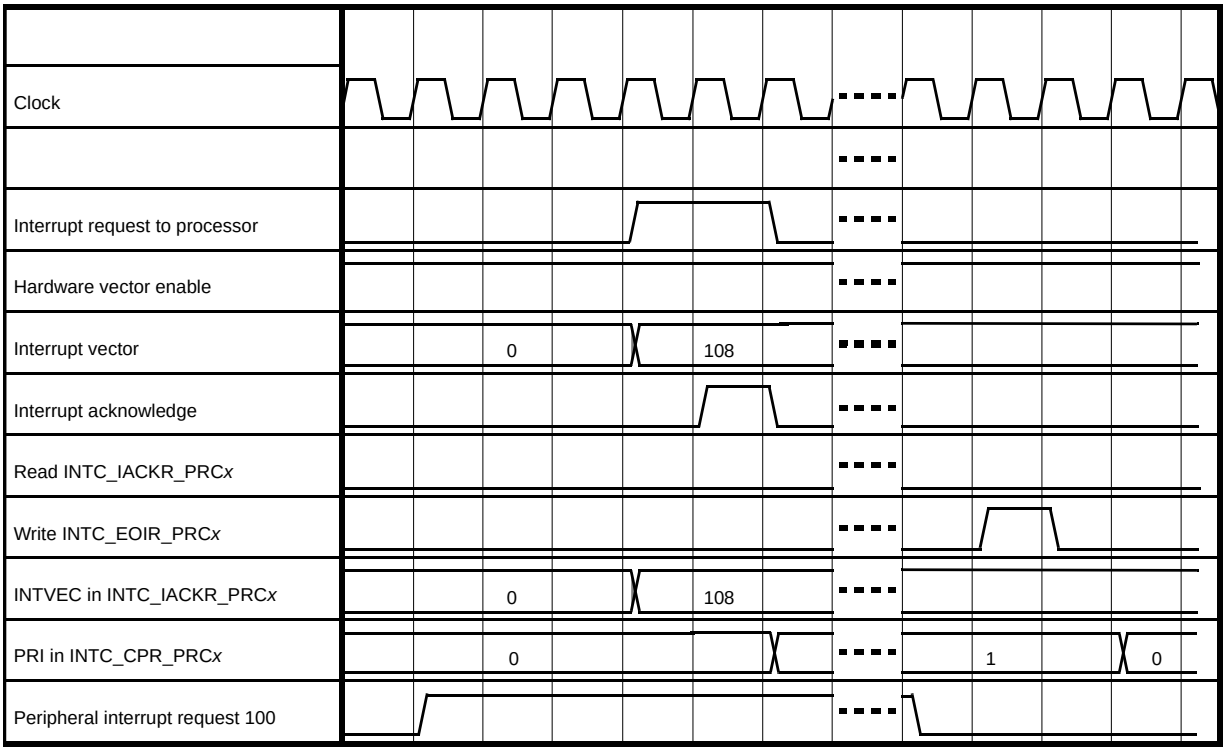


Figure 32-12. Hardware vector mode handshaking timing diagram

32.5 Initialization/application information

32.5.1 Initialization flow

After exiting reset, all of the PRI*x* and PRC_SEL*x* fields in the INTC_PSR0_3—INTC_PSR336 registers are 0, and PRI in the INTC_CPR_PRC0 register is 0xF. These reset values prevent the INTC from asserting the interrupt request to the processors. In addition, interrupts can be enabled or masked via the interrupts control registers located inside the peripherals. An initialization sequence for allowing the peripheral and software-settable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES_PRC0 and HVEN_PRC0 in INTC_BCR
configure VTBA_PRC0 in INTC_IACKR_PRC0
raise the PRIx fields to the desired processor in INTC_PSRx_x
set the enable bits or clear the mask bits for the peripheral interrupt requests
lower PRI in INTC_CPR_PRC0 to zero
enable processor recognition of interrupts
```

32.5.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

Example 32-1. Software vector mode

```
interrupt_exception_handler:
code to save SRR0 and SRR1
```

```
lis      r3,hi(INTC_IACKR_PRC0)     # form INTC_IACKR_PRC0 address
ori      r3,r3,lo(INTC_IACKR_PRC0)
lwz      r3,0x0(r3)                 # load INTC_IACKR_PRC0, which clears request to processor
lwz      r3,0x0(r3)                 # load address of ISR from vector table

code to enable processor recognition of interrupts and save context required by EABI

mtlr     r3                         # move INTC_IACKR_PRC0 contents into link register
blrl                                # branch to ISR; link register updated with epilog
                                    # address

epilog:
lis      r3,hi(INTC_EOIR_PRC0)      # form INTC_EOIR_PRC0 address
ori      r3,r3,lo(INTC_EOIR_PRC0)
li       r4,0x0                     # form 0 to write to INTC_EOIR_PRC0
stw      r4,0x0(r3)                 # store to INTC_EOIR_PRC0, informing INTC to lower priority

code to restore context required by EABI and disable processor recognition of interrupts
code to restore SRR0 and SRR1

rfi


vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
         .
         .
         .
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # return to epilog
```

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

Example 32-2. Hardware vector mode

```
interrupt_exception_handlerx:
b        interrupt_exception_handler_continuedx# 4 instructions available, branch to continue


interrupt_exception_handler_continuedx:
code to save SRR0 and SRR1
code to enable processor recognition of interrupts and save context required by EABI
```

```
bl      ISRx                    # branch to ISR for interrupt with vector x


epilog:
lis     r3,hi(INTC_EOIR_PRC0)   # form INTC_EOIR_PRC0 address
ori     r3,r3,lo(INTC_EOIR_PRC0)
li      r4,0x0                  # form 0 to write to INTC_EOIR_PRC0
stw     r4,0x0(r3)              # store to INTC_EOIR_PRC0, informing INTC to lower priority

code to restore context required by EABI and disable processor recognition of interrupts
code to restore SRR0 and SRR1

rfi


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                             # branch to epilog
```

32.5.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in the INTC_CPR_PRC0 register having a value of 0. The RTOS executes the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR_PRC0 priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRC0 priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRC0 priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR_PRC0 while the shared resource is being accessed.

An ISR whose PRI*x* in the INTC_PSR*x*_*x* register has a value of 0 will not cause an interrupt request to the selected processor, even if its peripheral or software-settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

32.5.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software-settable interrupt requests. However, if multiple peripheral or software-settable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, then the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software-settable interrupt requests asserted.

The example in Table 32-9 shows the order of execution of both situations, ISRs with different priorities and ISRs with the same priority.

Table 32-9. Order of ISR execution example

Step #	Step Description	Code executing at end of step						PRI in INTC_CPR at end of step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt exception handler	
	RTOS at priority 0 is executing.	X						0
	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
	Peripheral interrupt request 400 at priority 4 asserts. Interrupt taken.					X		4
	Peripheral interrupt request 300 at priority 3 asserts.					X		4
	Peripheral interrupt request 200 at priority 3 asserts.					X		4
	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	1
	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
	ISR208 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	1
	Interrupt taken. ISR308 starts to execute.				X			3
	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	1
	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRC0.						X	0
	RTOS continues execution.	X						0

1 ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software-settable interrupt requests.

32.5.5 Priority ceiling protocol

32.5.5.1 Elevating priority

The PRI field in the INTC_CPR_PRC0 register is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol therefore allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR_PRC*x* to 3, the ceiling of all of the ISR priorities. After they release the resource, they must lower the PRI value in INTC_CPR_PRC*x* to prevent further priority inversion. If they do not raise their priority, then ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts reduces the priority inversion time when accessing a shared resource. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

32.5.5.2 Ensuring coherency

Non-coherent accesses to a shared resource can occur. As an example, ISR1 and ISR2 both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing, and it writes to the INTC_CPR_PRC0. The instruction following this store is a store to a value in a shared coherent data block. Either just before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both of these stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent this corruption of a coherent data block, modifications to PRI in INTC_CPR_PRC0 can be made by those system services with the code sequence:

```
GetResource:
raise PRI
mbar
isync
ReleaseResource:
mbar
lower PRI
```

Non-OSEK applications can support modifications to PRI in INTC_CPR_PRC0 using the following steps:

1. Disable processor recognition of interrupts before elevating the current priority.
2. Elevate the current priority using a guarded and cache-inhibited write.
3. After the current priority is elevated, re-enable processor recognition of interrupts.
4. Delay 5 core clocks.

32.5.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using Rate Monotonic Scheduling (RMS) or a superset of it, Deadline Monotonic Scheduling (DMS). In RMS, the ISRs that have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2 which has a higher priority than ISR3. However, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which could be much less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 µs would share a priority, ISRs with request rates around 250 µs would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be covered, regardless of the number of ISRs.

Reducing the number of priorities does cause some priority inversion, which reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They can be placed at the same priority without any further priority inversion, and they do not need to use the PCP to access the shared resource.

32.5.7 Software-settable interrupt requests

The software-settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

32.5.7.1 Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in the INTC_PSR0*x*_*x* register, which becomes the PRI value in either INTC_CPR_PRC0 register with the interrupt acknowledge. The ISR, however, can have a portion of it that does not need to be executed at this higher priority. Therefore, executing this later portion that does not need to be executed at this higher priority can block the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than is needed for the later portion of the ISR. This priority inversion reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*x* bit in the INTC_SSCIR0_3 or the INTC_SSCIR4_7 register. Writing a 1 to SET*x* causes a software-settable interrupt request. This software-settable interrupt request, which usually will have a lower PRI*x* value in the INTC_PSR*x*_*x*, therefore will not cause priority inversion.

32.5.7.2 Scheduling an ISR on another processor

Since the SET*x* bits in the INTC_SSCIR*x_x* are memory mapped, processors in multiple processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work, and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that processor executing the software-settable ISR has not completed the work before asking it to again execute that ISR, it can check if the corresponding CLR*x* bit in INTC_SSCIR*x_x* is asserted before again writing a 1 to the SET*x* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. The procedure is that the first processor writes a 1 to a SET*x* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*x* bit and then writes 1 to a SET*x* bit on the first processor, informing it that it now can access the block of data.

32.5.8 Lowering priority within an ISR

In implementations without the software-settable interrupt requests in the INTC_SSCIR0_3 or INTC_SSCIR4_7 register, the only other way besides scheduling a task through an RTOS to not have priority inversion with an ISR whose work spans multiple priorities as described in Section 32.5.7.1, Scheduling a lower priority portion of an ISR, is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

NOTE

Lowering the PRI value in either INTC_CPR_PRC0 register within an ISR to below the ISR's corresponding PRI value in the INTC_PSR*x_x* register allows more preemptions than the depth of the LIFO can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid priority inversion.

32.5.9 Negating an interrupt request outside of its ISR

32.5.9.1 Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits, and consequently their corresponding interrupt requests too. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

32.5.9.2 Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own flag bit. One reason that an ISR clears multiple flag bits is because it serviced those other flag bits, and therefore the ISRs for these other flag bits do not need to be executed.

32.5.9.3 Proper setting of interrupt request priority

Whether an interrupt request negates outside of its own ISR due to the side effect of an ISR execution or the intentional clearing of a flag bit, the priorities of the peripheral or software-settable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in the INTC_PSR*x*_*x* registers must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits still can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to the INTC_SSCIR0_3 or the INTC_SSCIR4_7 register as the clearing of the flag bit that caused the present ISR to be executed. Refer to Section 32.4.3.1.2, End of interrupt exception handler.

A flag bit whose enable bit or mask bit is negating its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI*x* value in INTC_PSR*x*_*x*.

32.5.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he should want to read the contents, such as in debug mode, they are not memory mapped. The contents still can be read by popping the LIFO and reading the PRI field in Section 32.3.2.2, INTC Current Priority Register for Processor 0 (INTC_CPR_PRC0). The code sequence is:

```
pop_lifo:
store to INTC_EOIR_PRC0
load INTC_CPR_PRC0, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR_PRCx
load INTC_IACKR_PRC0
if stacked PRI values are not depleted, branch to push_lifo
```

32.6 Interrupt sources

Table 32-10 defines the interrupt sources for interrupt controller INTC_0 and INTC_1. Interrupts generated by on-platform peripherals are routed to their own interrupt controller only. Interrupts generated by off-platform peripherals are routed to both interrupt controllers.

In Decoupled Parallel Mode (DPM), the interrupt load can be distributed to both cores by assigning different interrupt levels (interrupt level 0 to effectively disable an IRQ) to the same source at each interrupt controller.

Table 32-10. Interrupt sources for INTC_0 and INTC_1

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
Section B (On-Platform Peripherals)					
0	0x0000	16	Software-settable flag 0	INTC_0 (Software)	INTC_1 (Software)
1	0x0010	16	Software-settable flag 1	INTC_0 (Software)	INTC_1 (Software)
2	0x0020	16	Software-settable flag 2	INTC_0 (Software)	INTC_1 (Software)
3	0x0030	16	Software-settable flag 3	INTC_0 (Software)	INTC_1 (Software)
4	0x0040	16	Software-settable flag 4	INTC_0 (Software)	INTC_1 (Software)
5	0x0050	16	Software-settable flag 5	INTC_0 (Software)	INTC_1 (Software)
6	0x0060	16	Software-settable flag 6	INTC_0 (Software)	INTC_1 (Software)
7	0x0070	16	Software-settable flag 7	INTC_0 (Software)	INTC_1 (Software)
8	0x0080	16	Not used		
9	0x0090	16	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall | Platform Flash Bank 2 Abort | Platform Flash Bank 2 Stall | Platform Flash Bank 3 Abort | Platform Flash Bank 3 Stall	ECSM_0	ECSM_1[1]
10	0x00A0	16	Combined Error	DMA_0	DMA_1
11	0x00B0	16	Channel 0	DMA_0	DMA_1
12	0x00C0	16	Channel 1	DMA_0	DMA_1
13	0x00D0	16	Channel 2	DMA_0	DMA_1
14	0x00E0	16	Channel 3	DMA_0	DMA_1
15	0x00F0	16	Channel 4	DMA_0	DMA_1
16	0x0100	16	Channel 5	DMA_0	DMA_1
17	0x0110	16	Channel 6	DMA_0	DMA_1
18	0x0120	16	Channel 7	DMA_0	DMA_1
19	0x0130	16	Channel 8	DMA_0	DMA_1
20	0x0140	16	Channel 9	DMA_0	DMA_1
21	0x0150	16	Channel 10	DMA_0	DMA_1
22	0x0160	16	Channel 11	DMA_0	DMA_1
23	0x0170	16	Channel 12	DMA_0	DMA_1
24	0x0180	16	Channel 13	DMA_0	DMA_1
25	0x0190	16	Channel 14	DMA_0	DMA_1

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
26	0x01A0	16	Channel 15	DMA_0	DMA_1
27	0x01B0	16	Not used		
28	0x01C0	16	SWT Timeout	SSWT_0	
29	0x01D0	16	SWT Timeout	SSWT_1	
30	0x01E0	16	Match on channel 0	STM_0	STM_1
31	0x01F0	16	Match on channel 1	STM_0	STM_1
32	0x0200	16	Match on channel 2	STM_0	STM_1
33	0x0210	16	Match on channel 3	STM_0	STM_1
34	0x0220	16	Not used		
35	0x0230	16	ECC_PlatformFlash_ noncorrectable error | ECC_PlatformRAM_ noncorrectable error	ECSM_0	ECSM_1[1]
36	0x0240	16	ECC_PlatformFlash_1bit_correction | ECC_PlatformRAM_1bit_correction	ECSM_0	ECSM_1[1]
37	0x0250	16	Master0 Snoop ipi_int	PCU[2]	—
			Master1 Snoop ipi_int	—	PCU[2]
Section C (Off-Platform Peripherals)					
38	0x0260	16	Not used		
39	0x0270	16	Not used		
40	0x0280	16	Not used		
41	0x0290	16	SIU External IRQ_0	SIUL	
42	0x02A0	16	SIU External IRQ_1	SIUL	
43	0x02B0	16	SIU External IRQ_2	SIUL	
44	0x02C0	16	SIU External IRQ_3	SIUL	
45	0x02D0	16	Not used		
46	0x02E0	16	Not used		
47	0x02F0	16	Not used		
48	0x0300	16	Not used		
49	0x0310	16	Not used		
MC_ME and MC_RGM					
50	0x0320	16	Not used		
51	0x0330	16	Safe Mode Interrupt	MC_ME	MC_ME

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
52	0x0340	16	Mode Transition Interrupt	MC_ME	MC_ME
53	0x0350	16	Invalid Mode Interrupt	MC_ME	MC_ME
54	0x0360	16	Invalid Mode Configuration	MC_ME	MC_ME
55	0x0370	16	Not used		
56	0x0380	16	Functional and destructive reset alternate event interrupt (ipi_int)	MC_RGM	MC_RGM
XOSC					
57	0x0390	16	XOSC counter expired (ipi_int_osc)	XOSC	XOSC
PIT					
58	0x03A0	16	Not used		
59	0x03B0	16	PITimer Channel 0	PIT	PIT
60	0x03C0	16	PITimer Channel 1	PIT	PIT
61	0x03D0	16	PITimer Channel 2	PIT	PIT
ADC_0					
62	0x03E0	16	ADC_EOC	ADC_0	
63	0x03F0	16	ADC_ER	ADC_0	
64	0x0400	16	ADC_WD	ADC_0	
FlexCAN_0					
65	0x0410	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_0	FlexCAN_0
66	0x0420	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_0	FlexCAN_0
67	0x0430	16	Reserved FLEXCAN_ESR_WAK	FlexCAN_0	FlexCAN_0
68	0x0440	16	FLEXCAN_BUF_00_03	FlexCAN_0	FlexCAN_0
69	0x0450	16	FLEXCAN_BUF_04_07	FlexCAN_0	FlexCAN_0
70	0x0460	16	FLEXCAN_BUF_08_11	FlexCAN_0	FlexCAN_0
71	0x0470	16	FLEXCAN_BUF_12_15	FlexCAN_0	FlexCAN_0
72	0x0480	16	FLEXCAN_BUF_16_31	FlexCAN_0	FlexCAN_0
73	0x0490	16	Not used		
DSPI_0					
74	0x04A0	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_0	DSPI_0
75	0x04B0	16	DSPI_SR[EOQF]	DSPI_0	DSPI_0
76	0x04C0	16	DSPI_SR[TFFF]	DSPI_0	DSPI_0

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
77	0x04D0	16	DSPI_SR[TCF]	DSPI_0	DSPI_0
78	0x04E0	16	DSPI_SR[RFDF]	DSPI_0	DSPI_0
LINFlex_0					
79	0x04F0	16	LINFlex_RXI	LINFlex_0	LINFlex_0
80	0x0500	16	LINFlex_TXI	LINFlex_0	LINFlex_0
81	0x0510	16	LINFlex_ERR	LINFlex_0	LINFlex_0
ADC_1					
82	0x0520	16	ADC_EOC	ADC_1	
83	0x0530	16	ADC_ER	ADC_1	
84	0x0540	16	ADC_WD	ADC_1	
FlexCAN_1					
85	0x0550	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_1	FlexCAN_1
86	0x0560	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_1	FlexCAN_1
87	0x0570	16	Reserved FLEXCAN_ESR_WAK		
88	0x0580	16	FLEXCAN_BUF_00_03	FlexCAN_1	FlexCAN_1
89	0x0590	16	FLEXCAN_BUF_04_07	FlexCAN_1	FlexCAN_1
90	0x05A0	16	FLEXCAN_BUF_08_11	FlexCAN_1	FlexCAN_1
91	0x05B0	16	FLEXCAN_BUF_12_15	FlexCAN_1	FlexCAN_1
92	0x05C0	16	FLEXCAN_BUF_16_31	FlexCAN_1	FlexCAN_1
93	0x05D0	16	Not used		
DSPI_1					
94	0x05E0	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_1	DSPI_1
95	0x05F0	16	DSPI_SR[EOQF]	DSPI_1	DSPI_1
96	0x0600	16	DSPI_SR[TFFF]	DSPI_1	DSPI_1
97	0x0610	16	DSPI_SR[TCF]	DSPI_1	DSPI_1
98	0x0620	16	DSPI_SR[RFDF]	DSPI_1	DSPI_1
LINFlex_1					
99	0x0630	16	LINFlex_RXI	LINFlex_1	LINFlex_1
100	0x0640	16	LINFlex_TXI	LINFlex_1	LINFlex_1
101	0x0650	16	LINFlex_ERR	LINFlex_1	LINFlex_1

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
ADC_2					
102	0x0660	16	ADC_EOC	ADC_2	
103	0x0670	16	ADC_ER	ADC_2	
104	0x0680	16	ADC_WD	ADC_2	
FlexCAN_2					
105	0x0690	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_2	
106	0x06A0	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_2	
107	0x06B0	16	Not used		
108	0x06C0	16	FLEXCAN_BUF_00_03	FlexCAN_2	
109	0x06D0	16	FLEXCAN_BUF_04_07	FlexCAN_2	
110	0x06E0	16	FLEXCAN_BUF_08_11	FlexCAN_2	
111	0x06F0	16	FLEXCAN_BUF_12_15	FlexCAN_2	
112	0x0700	16	FLEXCAN_BUF_16_31	FlexCAN_2	
113	0x0710	16	Not used		
DSPI_2					
114	0x0720	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	DSPI_2	
115	0x0730	16	DSPI_SR[EOQF]	DSPI_2	
116	0x0740	16	DSPI_SR[TFFF]	DSPI_2	
117	0x0750	16	DSPI_SR[TCF]	DSPI_2	
118	0x0760	16	DSPI_SR[RFDF]	DSPI_2	
LINFlex_2					
119	0x0770	16	LINFlex_RXI	LINFlex_2	
120	0x0780	16	LINFlex_TXI	LINFlex_2	
121	0x0790	16	LINFlex_ERR	LINFlex_2	
LINFlex_3					
122	0x07A0	16	LINFlex_RXI	LINFlex_3	
123	0x07B0	16	LINFlex_TXI	LINFlex_3	
124	0x07C0	16	LINFlex_ERR	LINFlex_3	
IIC_0					
125	0x07D0	16	I2C_0_IBSR[IBIF]	I2C_0	

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
IIC_1					
126	0x07E0	16	I2C_1_IBSR[IBIF]	I2C_1	
PIT					
127	0x07F0	16	PITimer Channel 3	PIT	PIT
128	0x0800	16	Not used		
129	0x0810	16	Not used		
130	0x0820	16	Not used		
FlexRay					
131	0x0830	16	LRNEIF | DRNEIF	FlexRay	
132	0x0840	16	LRCEIF | DRCEIF	FlexRay	
133	0x0850	16	FNEAIF	FlexRay	
134	0x0860	16	FNEBIF	FlexRay	
135	0x0870	16	WUPIF	FlexRay	
136	0x0880	16	PRIF	FlexRay	
137	0x0890	16	CHIF	FlexRay	
138	0x08A0	16	TBIF	FlexRay	
139	0x08B0	16	RBIF	FlexRay	
140	0x08C0	16	MIF	FlexRay	
EMIOS_0					
141	0x08D0	16	Reserve FLAG[0]	EMIOS_0	
142	0x08E0	16	Reserve FLAG[1]	EMIOS_0	
143	0x08F0	16	Reserve FLAG[2]	EMIOS_0	
144	0x0900	16	Reserve FLAG[3]	EMIOS_0	
145	0x0910	16	Reserve FLAG[4]	EMIOS_0	
146	0x0920	16	Reserve FLAG[5]	EMIOS_0	
147	0x0930	16	Reserve FLAG[6]	EMIOS_0	
148	0x0940	16	Reserve FLAG[7]	EMIOS_0	
149	0x0950	16	Reserve FLAG[8]	EMIOS_0	
150	0x0960	16	Reserve FLAG[9]	EMIOS_0	
151	0x0970	16	Reserve FLAG[10]	EMIOS_0	
152	0x0980	16	Reserve FLAG[11]	EMIOS_0	
153	0x0990	16	Reserve FLAG[12]	EMIOS_0	

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
154	0x09A0	16	Reserve FLAG[13]	EMIOS_0	
155	0x09B0	16	Reserve FLAG[14]	EMIOS_0	
156	0x09C0	16	Reserve FLAG[15]	EMIOS_0	
Section D (Off-Platform Peripherals)					
eTimer_0					
157	0x09D0	16	TC0IR	eTimer_0	
158	0x09E0	16	TC1IR	eTimer_0	
159	0x09F0	16	TC2IR	eTimer_0	
160	0x0A00	16	TC3IR	eTimer_0	
161	0x0A10	16	TC4IR	eTimer_0	
162	0x0A20	16	TC5IR	eTimer_0	
163	0x0A30	16	Not used		
164	0x0A40	16	Not used		
165	0x0A50	16	WTIF	eTimer_0	
166	0x0A60	16	Not used		
167	0x0A70	16	RCF	eTimer_0	
eTimer_1					
168	0x0A80	16	TC0IR	eTimer_1	
169	0x0A90	16	TC1IR	eTimer_1	
170	0x0AA0	16	TC2IR	eTimer_1	
171	0x0AB0	16	TC3IR	eTimer_1	
172	0x0AC0	16	TC4IR	eTimer_1	
173	0x0AD0	16	TC5IR	eTimer_1	
174	0x0AE0	16	Not used		
175	0x0AF0	16	Not used		
176	0x0B00	16	Not used		
177	0x0B10	16	Not used		
178	0x0B20	16	RCF	eTimer_1	
FlexPWM_0					
179	0x0B30	16	RF0	FlexPWM_0	
180	0x0B40	16	COF0	FlexPWM_0	
181	0x0B50	16	CAF0	FlexPWM_0	

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
182	0x0B60	16	RF1	FlexPWM_0	
183	0x0B70	16	COF1	FlexPWM_0	
184	0x0B80	16	CAF1	FlexPWM_0	
185	0x0B90	16	RF2	FlexPWM_0	
186	0x0BA0	16	COF2	FlexPWM_0	
187	0x0BB0	16	CAF2	FlexPWM_0	
188	0x0BC0	16	RF3	FlexPWM_0	
189	0x0BD0	16	COF3	FlexPWM_0	
190	0x0BE0	16	CAF3	FlexPWM_0	
191	0x0BF0	16	FFLAG	FlexPWM_0	
192	0x0C00	16	REF	FlexPWM_0	
CTU_0					
193	0x0C10	16	MRS_I	CTU_0	
194	0x0C20	16	T0_I	CTU_0	
195	0x0C30	16	T1_I	CTU_0	
196	0x0C40	16	T2_I	CTU_0	
197	0x0C50	16	T3_I	CTU_0	
198	0x0C60	16	T4_I	CTU_0	
199	0x0C70	16	T5_I	CTU_0	
200	0x0C80	16	T6_I	CTU_0	
201	0x0C90	16	T7_I	CTU_0	
202	0x0CA0	16	FIFO0_I	CTU_0	
203	0x0CB0	16	FIFO1_I	CTU_0	
204	0x0CC0	16	FIFO2_I	CTU_0	
205	0x0CD0	16	FIFO3_I	CTU_0	
206	0x0CE0	16	ADC_I	CTU_0	
207	0x0CF0	16	ERR_I	CTU_0	
Safety Port					
208	0x0D00	16	Not used		
209	0x0D10	16	Not used		
210	0x0D20	16	Not used		
211	0x0D30	16	Not used		

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
212	0x0D40	16	Not used		
213	0x0D50	16	Not used		
214	0x0D60	16	Not used		
215	0x0D70	16	Not used		
216	0x0D80	16	Not used		
DSPI_3					
217	0x0D90	16	DSPI_SR[TFUF] | DSPI_SR[RFOF]	Reserve DSPI_3	
218	0x0DA0	16	DSPI_SR[EOQF]	Reserve DSPI_3	
219	0x0DB0	16	DSPI_SR[TFFF]	Reserve DSPI_3	
220	0x0DC0	16	DSPI_SR[TCF]	Reserve DSPI_3	
221	0x0DD0	16	DSPI_SR[RFDF]	Reserve DSPI_3	
eTimer_2					
222	0x0DE0	16	TC0IR	eTimer_2	
223	0x0DF0	16	TC1IR	eTimer_2	
224	0x0E00	16	TC2IR	eTimer_2	
225	0x0E10	16	TC3IR	eTimer_2	
226	0x0E20	16	TC4IR	eTimer_2	
227	0x0E30	16	TC5IR	eTimer_2	
228	0x0E40	16	Not used		
229	0x0E50	16	Not used		
230	0x0E60	16	Not used		
231	0x0E70	16	Not used		
232	0x0E80	16	RCF	eTimer_2	
FlexPWM_1					
233	0x0E90	16	RF0	FlexPWM_1	
234	0x0EA0	16	COF0	FlexPWM_1	
235	0x0EB0	16	CAF0	FlexPWM_1	
236	0x0EC0	16	RF1	FlexPWM_1	
237	0x0ED0	16	COF1	FlexPWM_1	
238	0x0EE0	16	CAF1	FlexPWM_1	
239	0x0EF0	16	RF2	FlexPWM_1	

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
240	0x0F00	16	COF2	FlexPWM_1	
241	0x0F10	16	CAF2	FlexPWM_1	
242	0x0F20	16	RF3	FlexPWM_1	
243	0x0F30	16	COF3	FlexPWM_1	
244	0x0F40	16	CAF3	FlexPWM_1	
245	0x0F50	16	FFLAG	FlexPWM_1	
246	0x0F60	16	REF	FlexPWM_1	
SEMA4_0					
247	0x0F70	16	cp0_semaphore_int	SEMA4_0	—
			cp1_semaphore_int	—	SEMA4_0
SEMA4_1					
248	0x0F80	16	cp0_semaphore_int	SEMA4_1	—
			cp1_semaphore_int	—	SEMA4_1
FCCU					
249	0x0F90	16	Not used		
250	0x0FA0	16	irq_alarm_b	FCCU	
251	0x0FB0	16	external alarm interrupt line	FCCU	
252	0x0FC0	16	irq_rccs_b[0]	FCCU	
253	0x0FD0	16	irq_rccs_b[1]	FCCU	
PMU					
254	0x0FE0	16	Not used		
SWG					
255	0x0FF0	16	Not used		
256	0x1000	16	Not used		
257	0x1010	16	Not used		
258	0x1020	16	Not used		
259	0x1030	16	Not used		
DMA (continued)					
260	0x1040	16	Channel 16	DMA_0	DMA_1
261	0x1050	16	Channel 17	DMA_0	DMA_1
262	0x1060	16	Channel 18	DMA_0	DMA_1
263	0x1070	16	Channel 19	DMA_0	DMA_1

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
264	0x1080	16	Channel 20	DMA_0	DMA_1
265	0x1090	16	Channel 21	DMA_0	DMA_1
266	0x10A0	16	Channel 22	DMA_0	DMA_1
267	0x10B0	16	Channel 23	DMA_0	DMA_1
268	0x10C0	16	Channel 24	DMA_0	DMA_1
269	0x10D0	16	Channel 25	DMA_0	DMA_1
270	0x10E0	16	Channel 26	DMA_0	DMA_1
271	0x10F0	16	Channel 27	DMA_0	DMA_1
272	0x1100	16	Channel 28	DMA_0	DMA_1
273	0x1110	16	Channel 29	DMA_0	DMA_1
274	0x1120	16	Channel 30	DMA_0	DMA_1
275	0x1130	16	Channel 31	DMA_0	DMA_1
PDI					
276	0x1140	16	Line Valid	PDI	
277	0x1150	16	Frame Valid	PDI	
278	0x1160	16	Error Interrupt (FIFO underflow | FIFO overflow)	PDI	
279	0x1170	16	DMA Done	PDI	
280	0x1180	16	Not used		
281	0x1190	16	Not used		
FlexCAN_3					
282	0x11A0	16	FLEXCAN_ESR[ERR_INT]	FlexCAN_3	
283	0x11B0	16	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_3	
284	0x11C0	16	Reserve FLEXCAN_ESR_WAK	FlexCAN_3	
285	0x11D0	16	FLEXCAN_BUF_00_03	FlexCAN_3	
286	0x11E0	16	FLEXCAN_BUF_04_07	FlexCAN_3	
287	0x11F0	16	FLEXCAN_BUF_08_11	FlexCAN_3	
288	0x1200	16	FLEXCAN_BUF_12_15	FlexCAN_3	
289	0x1210	16	FLEXCAN_BUF_16_31	FlexCAN_3	
290	0x1220	16	Not used		
DRAMC					

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
291	0x1230	16	FIFO OV | FIFO UV	DRAMC	
IIC_2					
292	0x1240	16	I2C_2_IBSR[IBIF]	I2C_2	
293	0x1250	16	Not used		
294	0x1260	16	Not used		
295	0x1270	16	Not used		
296	0x1280	16	Not used		
297	0x1290	16	Not used		
298	0x12A0	16	Not used		
299	0x12B0	16	Not used		
300	0x12C0	16	Not used		
301	0x12D0	16	Not used		
ADC_3					
302	0x12E0	16	ADC_EOC	ADC_2	
303	0x12F0	16	ADC_ER	ADC_2	
304	0x1300	16	ADC_WD	ADC_2	
FlexPWM_2					
305	0x1310	16	RF0	FlexPWM_2	
306	0x1320	16	COF0	FlexPWM_2	
307	0x1330	16	CAF0	FlexPWM_2	
308	0x1340	16	RF1	FlexPWM_2	
309	0x1350	16	COF1	FlexPWM_2	
310	0x1360	16	CAF1	FlexPWM_2	
311	0x1370	16	RF2	FlexPWM_2	
312	0x1380	16	COF2	FlexPWM_2	
313	0x1390	16	CAF2	FlexPWM_2	
314	0x13A0	16	RF3	FlexPWM_2	
315	0x13B0	16	COF3	FlexPWM_2	
316	0x13C0	16	CAF3	FlexPWM_2	
317	0x13D0	16	FFLAG	FlexPWM_2	
318	0x13E0	16	REF	FlexPWM_2	

Table 32-10. Interrupt sources for INTC_0 and INTC_1 (continued)

IRQ #	Offset	Size (bytes)	Source signal	Source module for INTC_0	Source module for INTC_1
CTU_1					
319	0x13F0	16	MRS_I	CTU_1	
320	0x1400	16	T0_I	CTU_1	
321	0x1410	16	T1_I	CTU_1	
322	0x1420	16	T2_I	CTU_1	
323	0x1430	16	T3_I	CTU_1	
324	0x1440	16	T4_I	CTU_1	
325	0x1450	16	T5_I	CTU_1	
326	0x1460	16	T6_I	CTU_1	
327	0x1470	16	T7_I	CTU_1	
328	0x1480	16	FIFO0_I	CTU_1	
329	0x1490	16	FIFO1_I	CTU_1	
330	0x14A0	16	FIFO2_I	CTU_1	
331	0x14B0	16	FIFO3_I	CTU_1	
332	0x14C0	16	ADC_I	CTU_1	
333	0x14D0	16	ERR_I	CTU_1	
Ethernet (FEC)					
334	0x14E0	16	FEC Transmit Frame Flag	Ethernet (FEC)	
335	0x14F0	16	FEC Receive Frame Flag	Ethernet (FEC)	
336	0x1500	16	EIR[HBERR] | EIR[BABR] | EIR[BABT] | EIR[GRA] | EIR[TXB] | EIR[RXB] | EIR[MII] | EIR[EBERR] | EIR[LC] | EIR[RL] | EIR[UN]	Ethernet (FEC)	

1 The PFLASHC_1 is also operational in DPM. The ECSM_1 instantiation receives any flash memory ECC status or RWW information and therefore can signal any RWW, ECC SBC, or non-correctable ECC interrupt to the INTC_1. Therefore CORE_1 will also be able to get any status notification on flash memory RWW events and ECC errors in DPM.

2 Available in DPM only.

This page is intentionally left blank.

Chapter 33
JTAG Controller (JTAGC)

33.1 Introduction

Figure 33-1 shows a block diagram of the JTAGC module.

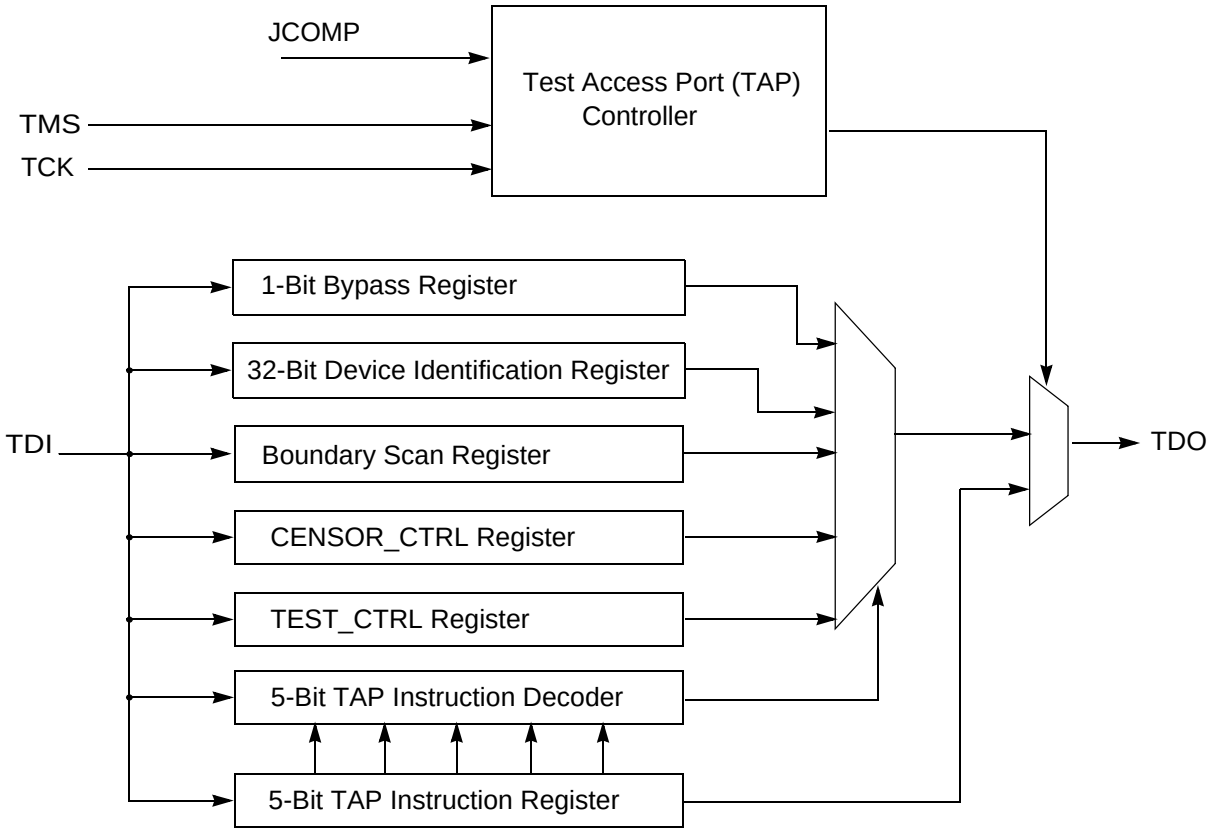


Figure 33-1. JTAG (IEEE 1149.1) block diagram

33.1.1 Overview

The JTAGC block provides the means to test device functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. All data input to and output from the JTAGC block is communicated in serial format.

33.1.2 Features

The JTAGC block is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
  - 4 pins (TDI, TMS, TCK, and TDO)

- A JCOMP input that provides reset control and the ability to share the TAP.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions as well as several public and private device-specific instructions. Refer to Table 33-4 for a list of supported instructions.
- Sharing of the TAP with other TAP controllers via ACCESS_AUX_TAP_*x* instructions.
- Test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry.

33.1.3 Modes of operation

The JTAGC block uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

33.1.3.1 Reset

The JTAGC block is placed in reset when either power-on reset is asserted, JCOMP is negated, or the TMS input is held high for enough consecutive rising edges of TCK to sequence the TAP controller state machine into the Test-Logic-Reset state. Holding TMS high for 5 consecutive rising edges of TCK evokes the Test-Logic-Reset state regardless of the current TAP controller state. Asserting power-on reset or setting JCOMP to a value other than the value required to enable the JTAGC block results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

33.1.3.2 IEEE 1149.1-2001 defined test modes

The JTAGC block supports several IEEE 1149.1-2001 defined test modes. A test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE, and SAMPLE/PRELOAD. Each instruction defines the set of data register(s) that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP, or reserved instructions are active. The functionality of each test mode is explained in more detail in Section 33.4.5, JTAGC block instructions.

33.1.3.3 Bypass mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC block into bypass mode. While in bypass mode, the single-bit bypass shift register provides a minimum-length serial path to shift data between TDI and TDO.

33.2 External signal description

33.2.1 Overview

The JTAGC provides five signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in Table 33-1.

Table 33-1. JTAG signal properties[1]

Name	I/O	Function	Reset State	Pull[2]
TCK	Input	Test Clock	—	Down
TDI	Input	Test Data In	—	Up
TDO	Output	Test Data Out	High-impedance[3]	—
TMS	Input	Test Mode Select	—	Up
JCOMP	Input	JTAG Compliancy	—	Down

1 If LBIST is enabled, an external pull between 1K and 100K ohms must be connected from TCK to either power or ground to avoid LBIST failures. LBIST MISRs can show mismatch when self test is started upon software induced destructive reset using the MC_ME module or a VDD_HV_FLA supply LVD induced reset. Long External reset can also trigger a self test failure but it should not be used as per caution in MPC567xK Reference manual.

2 Pullup/pulldown is not implemented in this block. Pullup/pulldown devices are implemented in the pads.

3 TDO output buffer enable is negated when the JTAGC is not in the Shift-IR or Shift-DR states. A weak pull may be required on TDO to prevent the signal from floating when tools are attached.

33.2.2 Detailed signal descriptions

This section describes each of the signals listed in Table 33-1 in more detail.

33.2.2.1 TCK—Test Clock Input

Test Clock Input (TCK) is an input pin used to synchronize the test logic and control register access through the TAP.

33.2.2.2 TDI—Test Data Input

Test Data Input (TDI) is an input pin that receives serial test instructions and data. TDI is sampled on the rising edge of TCK.

33.2.2.3 TDO—Test Data Output

Test Data Output (TDO) is an output pin that transmits serial output for test instructions and data. TDO is three-stateable and is actively driven only in the Shift-IR and Shift-DR states of the TAP controller state machine, which is described in Section 33.4.3, TAP controller state machine. The TDO output of this block is clocked on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

33.2.2.4 TMS—Test Mode Select

Test Mode Select (TMS) is an input pin used to sequence the IEEE 1149.1-2001 test control state machine. TMS is sampled on the rising edge of TCK.

33.2.2.5 JCOMP—JTAG Compliancy

The JCOMP signal provides IEEE 1149.1-2001 compatibility and provides the ability to share the TAP. The JTAGC TAP controller is enabled when JCOMP is set to the JTAGC enable encoding — otherwise the JTAGC TAP controller remains in reset.

33.3 Register description

This section provides a detailed description of the JTAGC block registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

33.3.1 Register descriptions

The JTAGC block registers are described in this section.

33.3.1.1 Instruction Register

The JTAGC block uses a 5-bit instruction register as shown in Figure 33-2. The instruction register allows instructions to be loaded into the block to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the IDCODE instruction. During the Capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

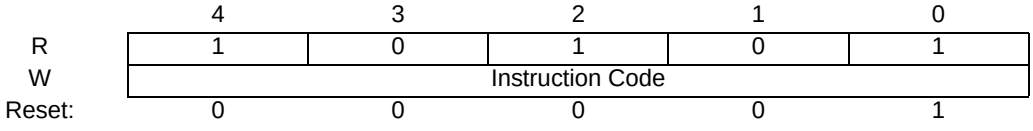


Figure 33-2. 5-Bit Instruction Register

33.3.1.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ, or reserve instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

33.3.1.3 Device Identification Register

The device identification register, shown in Figure 33-3, allows the revision number, part number, manufacturer, and design center responsible for the design of the part to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the Capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the Update-DR state. The part revision number (PRN) and part identification number (PIN) fields are system plugs, and the manufacturer identity code (MIC) is a constant value assigned to the manufacturer by the JEDEC.

The shift register LSB is forced to logic 1 on the rising edge of TCK following entry into the Capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are forced to the value of the device identification register on the rising edge of TCK following entry into the Capture-DR state.

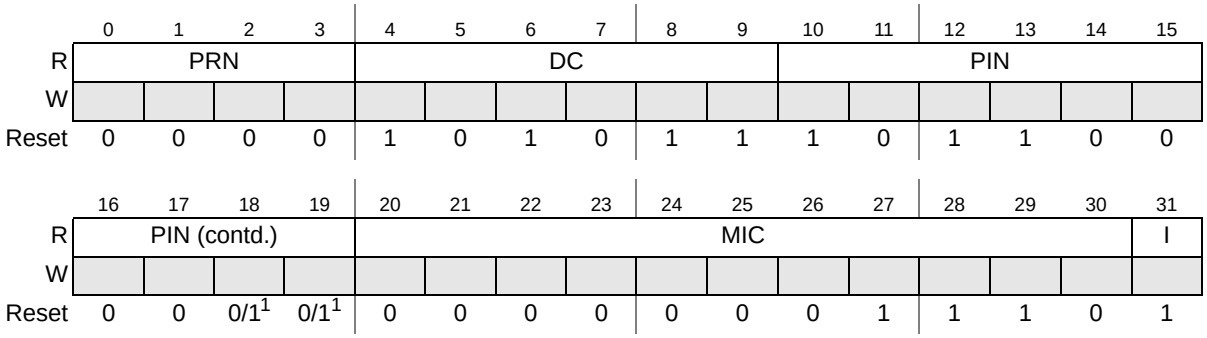


Figure 33-3. Device Identification Register

[1] Dependent on device version.

Table 33-2. Device Identification Register field descriptions

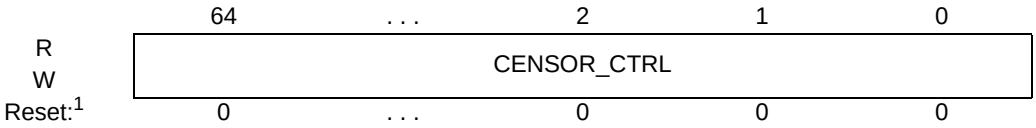
Field	Description
PRN	Part Revision Number. Bits [31:28] contain the revision number of the part.
DC	Design Center. Bits [27:22] indicate the design center. 0x2B (0b101011) for MPC5675K.
PIN	Part Identification Number for MPC5675K. Bits [21:12] contain the part number of the device. Cut1 0x2C0 (0b1011000000) Cut2 0x2C1 (0b1011000001)
MIC	Manufacturer Identity Code. Bits [11:1] contain the reduced Joint Electron Device Engineering Council (JEDEC) ID. 0x00E for Freescale.

Table 33-2. Device Identification Register field descriptions (continued)

Field	Description
I	IDCODE Register ID. When set to 1, bit I identifies this register as the device identification register and not the bypass register.

33.3.1.4 CENSOR_CTRL Register

The CENSOR_CTRL register is a 65-bit shift register path from TDI to TDO selected when the CENSOR_CTRL instruction is active. The default reset value of the CENSOR_CTRL register is 65’b0. The CENSOR_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. Once the CENSOR_CTRL instruction is executed, the register value remains valid until a JTAG reset occurs.



1 The reset value of CENSOR_CTRL is 65’b0.

Figure 33-4. CENSOR_CTRL Register

The CENSOR_CTRL bits are used to control device censorship functions.

33.3.1.5 TEST_CTRL Register

The TEST_CTRL register is a 10-bit shift register path from TDI to TDO selected when the TEST_CTRL instruction is active. The size of the TEST_CTRL register is defined by the TEST_CTRL_SIZE parameter. The default reset value of the TEST_CTRL register is defined by the TEST_CTRL_RST parameter. The TEST_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. The parallel hold register bits TEST_CTRL correspond directly to the JTAGC output control signals jtag_test_ctrl[9:0]. The jtag_test_ctrl signals are used to control device-dependent test features. Once the TEST_CTRL instruction is executed, jtag_test_ctrl remains valid until a JTAGC reset occurs.

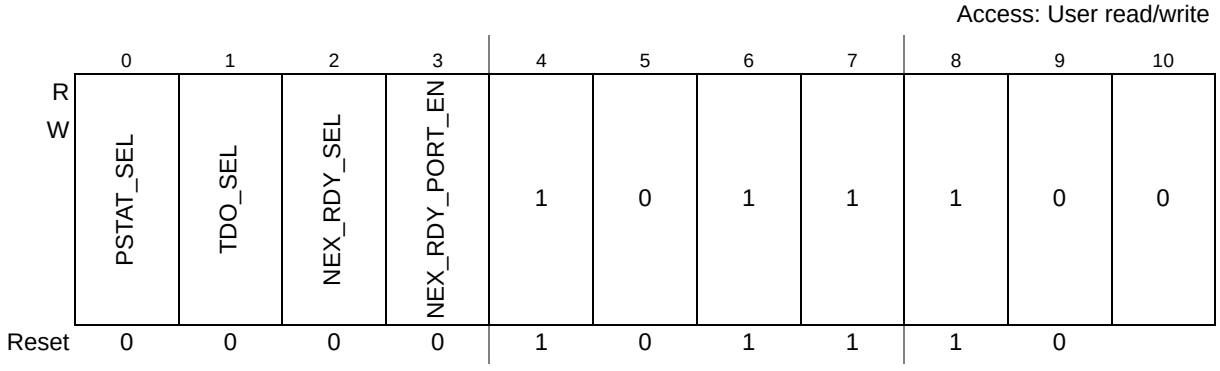


Figure 33-5. TEST_CTRL Register

Table 33-3. TEST_CTRL Register field descriptions

Field	Description
PSTAT_SEL	Processor Status Select Selects whether the Core_0 or the Core_1 processor status outputs appear on the Nexus MDO pins when Processor Status Mode is enabled in the NPC Port Configuration Register. 0 Core_0 PSTAT outputs selected. 1 Core_1 PSTST outputs selected.
TDO_SEL	TDO Select Selects whether the Core_0 or Core_1 TDO output is observable. 0 Core_0 TDO output selected. 1 Core_1 TDO output selected.
NEX_RDY_SEL	Nexus RDY select Selects whether RDY from Core_0 or Core_1 is driven as an output. 0 Core_0 RDY output selected. 1 Core_1 RDY output selected.
NEX_RDY_PORT_EN	Nexus RDY Port enable Used as hard enable for Nexus RDY signal in IOMUX 0 RDY not selected to chip top RDY. 1 RDY selected through IOMUX also IOMUXC PCR settings.

33.3.1.6 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in Section 33.4.6, Boundary scan.

33.4 Functional description

33.4.1 JTAGC reset configuration

While in reset, the TAP controller is forced into the Test-Logic-Reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

33.4.2 IEEE 1149.1-2001 (JTAG) test access port

The JTAGC block uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to Section 33.4.5.9, ACCESS_AUX_TAP_x instructions.

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in Figure 33-6. This applies for the instruction register, test data registers, and the bypass register.

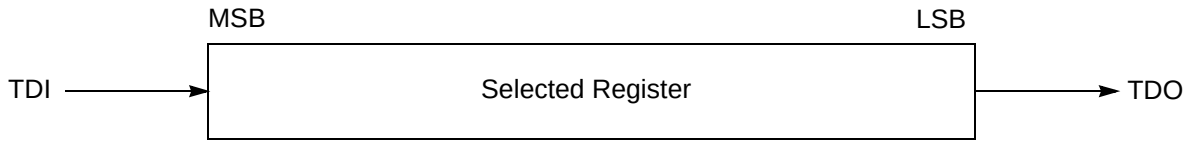
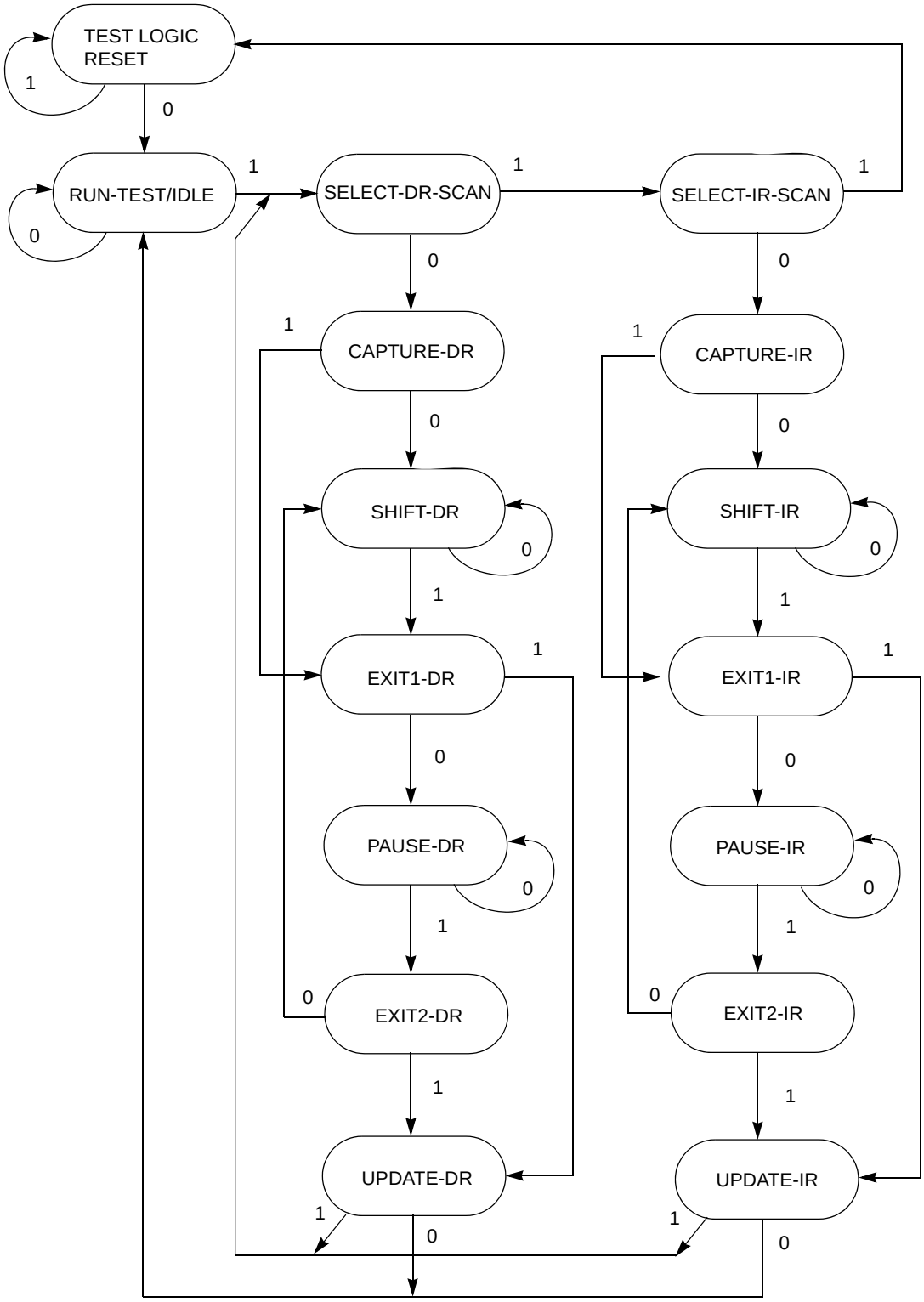


Figure 33-6. Shifting data through a register

33.4.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. Figure 33-7 shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal. As Figure 33-7 shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the Test-Logic-Reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 33-7. IEEE 1149.1-2001 tap Controller Finite State Machine

33.4.3.1 Enabling the TAP controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

33.4.3.2 Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is achieved by loading the instruction register with any of the JTAGC block instructions while the JTAGC is enabled. Instructions are shifted in via the Select-IR-Scan path and loaded in the Update-IR state. At this point, all data register access is performed via the Select-DR-Scan path.

The Select-DR-Scan path reads or writes the register data by shifting in the data (LSB first) during the Shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the Capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the Update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

33.4.4 Enabling debug of a censored device

When a device is in a censored state, the debug port (JTAG/Nexus) is disabled and only JTAG BSDL commands can be used. Access to the Nexus/JTAG clients on a censored device requires inputting the proper password into the JTAG Censorship Control Register during reset.

NOTE

When the debug port is enabled on a censored device, it is enabled only until the next reset.

Figure 33-8 shows the logic that enables access to Nexus clients in a censored device using the JTAG port.

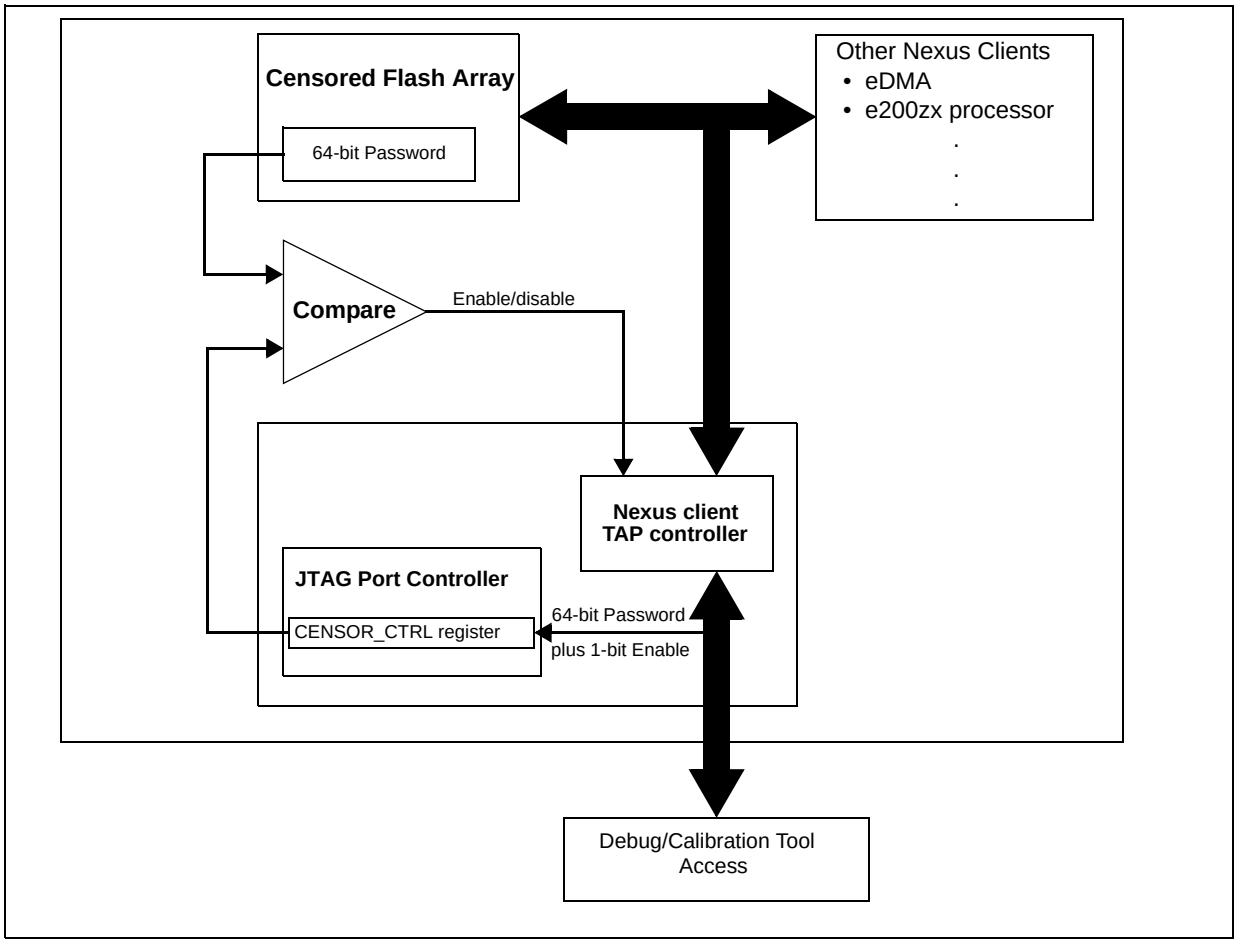


Figure 33-8. Enabling JTAG/Nexus port access on a censored device

The steps to enable the debug port on a censored device are as follows:

1. After the $\overline{\text{RSTOUT}}$ pin is negated, hold the device in system reset state using a debugger or other tool.
2. While the device is being held in system reset state, shift the 64-bit password into the CENSOR_CTRL register (see Section 33.3.1.4, CENSOR_CTRL Register) via the JTAG port using the JTAG ENABLE_CENSOR_CTRL instruction. The JTAG serial password is compared to the serial boot flash password from the flash shadow block. The MPC5675K implements an extra enable bit in the CENSOR_CTRL register for allowing access via a debug tool. This is shown in the above figure. The enable bit must be set (0b1) as the last bit in the JTAG scan data (on the 65th clock of the JTAG clock [TCK] in the Shift-DR state). For more information on the JTAG state machine, see the device Reference Manual.
3. If a match is detected, the Nexus client TAP controller enters normal operation mode and the DISNEX flag in the SIU_CCR register is negated, indicating Nexus is enabled. Upon negation of reset the debug / calibration tool is able to access the device via NEXUS port and JTAG. If the JTAG serial password does not match the serial boot flash password or the serial boot flash password is an illegal password, then the debug / calibration tool is not able to access the device.

After the debug port is enabled, the tool can access the censored device and can erase and reprogram the shadow flash block in order to uncensor the device.

NOTE

If the shadow flash block is erased without reprogramming a new valid password before a reset, it will contain an illegal password and the debug port will be inaccessible.

4. Subsequent resets will clear the JTAG censor password register and the Nexus client TAP controller will hold in reset again. Therefore, the tool must resend the JTAG serial password, as described above, in order to enable the Nexus client TAP controller again.

33.4.5 JTAGC block instructions

The JTAGC block implements the IEEE 1149.1-2001 defined instructions listed in Table 33-4. This section gives an overview of each instruction; refer to the IEEE 1149.1-2001 standard for more details. All undefined opcodes are reserved.

Table 33-4. JTAG instructions

Instruction	Code[4:0]	Instruction summary
IDCODE	00001	Selects device identification register for shift.
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation.
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation.
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset.
TEST_CTRL	00110	Selects TEST_CTRL register.
CENSOR_CTRL	00111	Selects CENSOR_CTRL register.
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset.
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset.
ACCESS_AUX_TAP_*x*	10000–11110	Grants one of the auxiliary TAP controllers ownership of the TAP as shown in the cells below. The number of auxiliary TAP controllers sharing the port is SHARE_CNT.
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller.
ACCESS_AUX_TAP_CORE_0	10001	Enables access to Core_0 TAP controller only.
ACCESS_AUX_TAP_NXSS_LSM	10110	Enables access to Nexus Crossbar Slave SRAM Data Trace TAP Controller. Both SRAM Data Trace modules receive TDI input data. TDO output data source is selected via the TEST_CTRL[TDO_SEL] bit.
ACCESS_AUX_TAP_NXSS0	10111	Enables access to Nexus Crossbar Slave Data Trace 0 TAP.
ACCESS_AUX_TAP_NXSS1	11000	Enables access to Nexus Crossbar Slave Data Trace 1 TAP.
ACCESS_AUX_TAP_CORE_1	11001	Enables access to Core_1 TAP controller only.
ACCESS_AUX_TAP_LSM	11010	Enables access to Core TAP controllers in LS mode. Both cores receive TDI input.

Table 33-4. JTAG instructions (continued)

Instruction	Code[4:0]	Instruction summary
ACCESS_AUX_TAP_MULTI	11100	Enables access to and serializes the Core_0 and Core_1 TAPs.
BYPASS	11111	Selects bypass register for data operations.
Factory debug reserved	00101, 00110, 01010, 00111, 10001, 10010, 10011, 10100, 10101, 11011	Intended for factory debug only.
Reserved[1]	All other opcodes	Decoded to select bypass register.

1 The manufacturer reserves the right to change the decoding of reserved instruction codes in the future.

33.4.5.1 IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC block is reset.

33.4.5.2 SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the Capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the Shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the Update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

33.4.5.3 SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the Capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the Update-DR state. Both the data capture and the shift operation are transparent to system operation.

33.4.5.4 EXTEST—External Test instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

33.4.5.5 TEST_CTRL instruction

The TEST_CTRL instruction selects the TEST_CTRL register for connection as the shift path between TDI and TDO.

33.4.5.6 CENSOR_CTRL instruction

The CENSOR_CTRL instruction selects the CENSOR_CTRL register for connection as the shift path between TDI and TDO.

33.4.5.7 HIGHZ instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

33.4.5.8 CLAMP instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

33.4.5.9 ACCESS_AUX_TAP_*x* instructions

The JTAGC is configurable to allow up to fifteen other TAP controllers on the device to share the port with it. This is done by providing ACCESS_AUX_TAP_x instructions for each of these TAP controllers. When this instruction is loaded, control of the JTAG pins are transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive. Instructions not used to access an auxiliary TAP controller on a device are treated like the BYPASS instruction.

33.4.5.10 BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

33.4.6 Boundary scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

33.5 Initialization/application information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC block and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to the JTAGC enable value, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

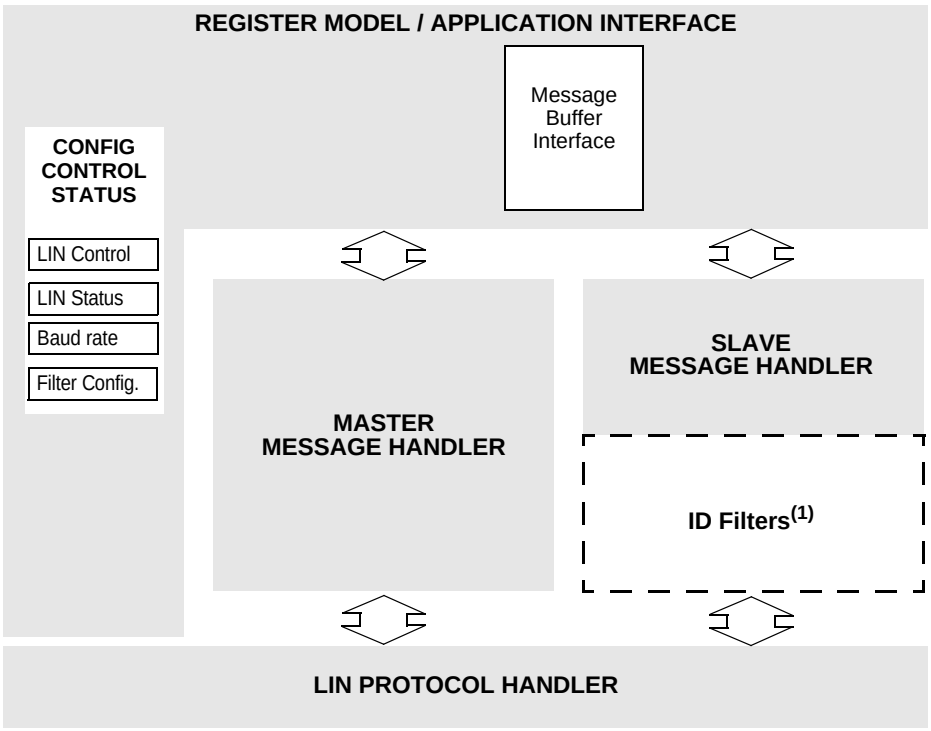
This page is intentionally left blank.

Chapter 34
LIN Controller (LINFlexD)

34.1 Introduction

The LINFlexD (Local Interconnect Network Flexible with DMA support) controller interfaces the LIN network and supports the LIN protocol versions 1.3, 2.0, 2.1, and J2602 in both master and slave modes. LINFlexD includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load, and allow slave node resynchronization.

Figure 34-1 shows the LINFlexD block diagram.



1 Filter activation optional

Figure 34-1. LINFlexD block diagram

34.2 Main features

The LINFlexD controller can operate in several modes, each of which has a distinct set of features. These distinct features are described in the following sections.

In addition, the LINFlexD controller has several features common to all modes:

- Fractional baud rate generator

- 3 operating modes for power saving and configuration registers lock
  - Initialization
  - Normal
  - Sleep
- 2 test modes
  - Loopback
  - Self-test
- Maskable interrupts

34.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1, and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for in-application programming purposes
- Wakeup event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response, and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with internal RC oscillator as clock source
- Identifier filters for autonomous message handling in slave mode

34.2.2 UART mode features

- Full-duplex communication
- Selectable frame size:
  - 8-bit frame
  - 9-bit frame
  - 16-bit frame
  - 17-bit frame
- Selectable parity:
  - Even
  - Odd
  - 0
  - 1

- 4-byte buffer for reception, 4-byte buffer for transmission
- 12-bit counter for timeout management

34.2.3 Debug mode

When the MCU is in debug mode, the LINFlexD behavior is unaffected and remains dictated by the mode of the LINFlexD.

34.3 The LIN protocol

The LIN (Local Interconnect Network) is a serial communication protocol. The topology of a LIN network is shown in Figure 34-2. A LIN network consists of:

- One master task
- Several slave tasks
- The LIN bus

A master node contains the master task as well as a slave task — all other nodes contain a slave task only. The master node decides when and which frame shall be transferred on the bus. The slave task provides the data to be transported by the frame.

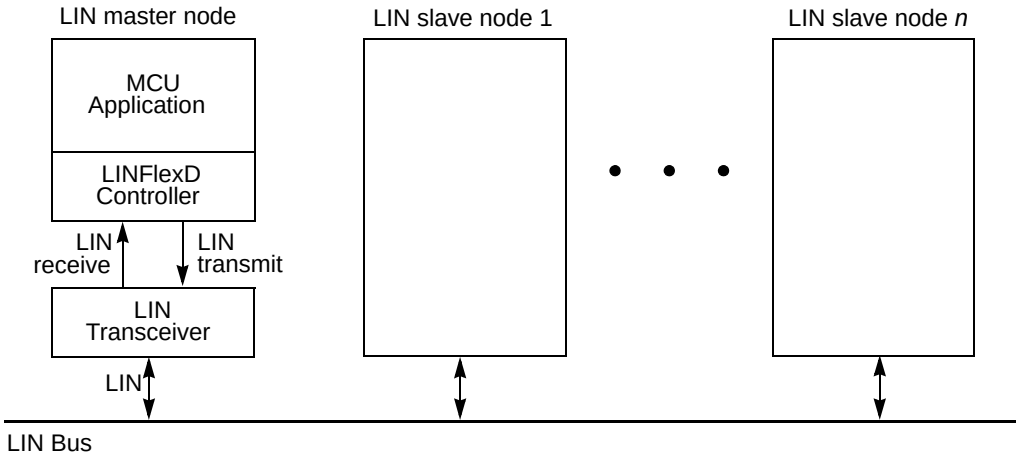


Figure 34-2. LIN network topology

34.3.1 Dominant and recessive logic levels

The LIN bus defines two logic levels, dominant and recessive, as follows:

- Dominant: logical low level (0)
- Recessive: logical high level (1)

34.3.2 LIN frames

A frame consists of a header provided by the master task and a response provided by the slave task, as shown in Figure 34-3.

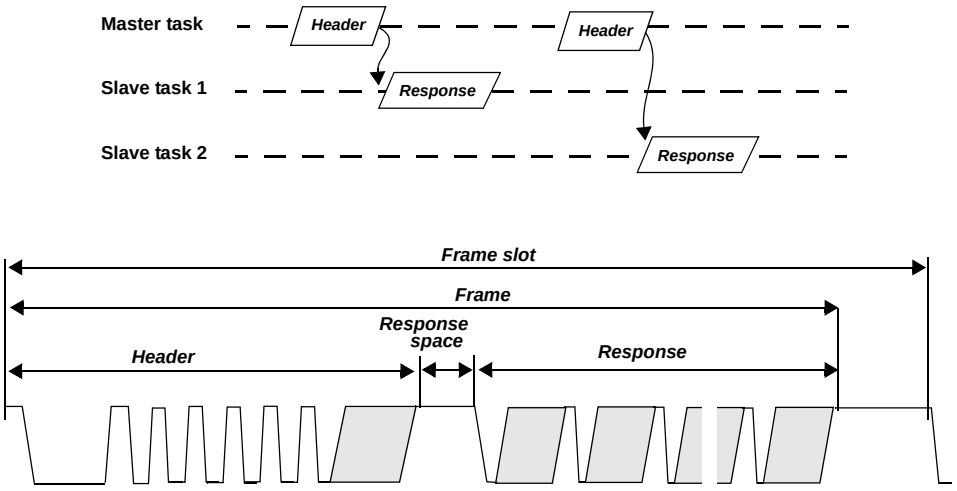


Figure 34-3. LIN frame structure

34.3.3 LIN header

The header consists of:

- A break field (described in Section 34.3.3.1, Break field)
- A sync pattern (described in Section 34.3.3.2, Sync pattern)
- An identifier (described in Section 34.3.4.2, Identifier)

The slave task associated with the identifier provides the response.

34.3.3.1 Break field

The break field, shown in Figure 34-4, is used to signal the beginning of a new frame. It is always generated by the master and consists of:

- At least 13 dominant bits, including the start bit
- At least one recessive bit that functions as break delimiter

Figure 34-4. Break field

34.3.3.2 Sync pattern

The sync pattern is a byte consisting of alternating dominant and recessive bits as shown in Figure 34-5. It forms a data value of 0x55.

Figure 34-5. Sync pattern

34.3.4 Response

The response consists of:

- A data field (described in Section 34.3.4.1, Data field)
- A checksum (described in Section 34.3.4.3, Checksum)

The slave task interested in the data associated with the identifier receives the response and verifies the checksum.

34.3.4.1 Data field

The structure of the data field transmitted on the LIN bus is shown in Figure 34-6. The LSB of the data is sent first and the MSB last. The start bit is encoded as a dominant bit and the stop bit is encoded as a recessive bit.

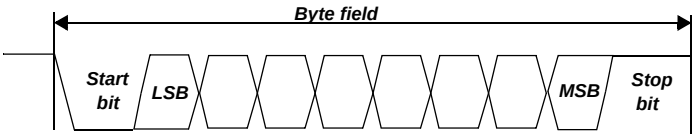


Figure 34-6. Structure of the data field

34.3.4.2 Identifier

The identifier, shown in Figure 34-7, consists of two sub-fields:

- The identifier value (in bits 0–5)
- The identifier parity (in bits 6–7)

The parity bits P0 and P1 are defined as follows:

- P0 = ID0 XOR ID1 XOR ID2 XOR ID4
- P1 = NOT(ID1 XOR ID3 XOR ID4 XOR ID5)

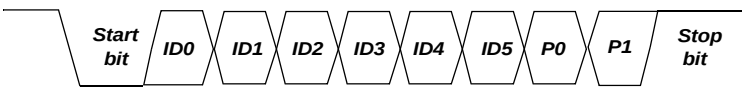


Figure 34-7. Identifier

34.3.4.3 Checksum

The checksum contains the inverted 8-bit sum (with carry) over one of two possible groups:

- The classic checksum sums all data bytes, and is used for communication with LIN 1.3 slaves.
- The enhanced checksum sums all data bytes and the identifier, and is used for communication with LIN 2.0 (or later) slaves.

34.4 LINFlexD and software intervention

The increasing number of communication peripherals embedded on microcontrollers (for example, CAN, LIN, SPI) requires more and more CPU resources for the communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 kbps is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as is usually the case.

To minimize the CPU load in master mode, LINFlexD handles the LIN messages autonomously.

In master mode, after the software has triggered the header transmission, LINFlexD does not request any software (that is, application) intervention until either of the following occurs:

- The next header transmission request in transmission mode
- The checksum reception in reception mode

To minimize the CPU load in slave mode, LINFlexD requires software intervention only to:

- Trigger transmission or reception or data discard, depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for slave mode, LINFlexD requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode).

The software uses the control, status, and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

34.5 Summary of operating modes

The LINFlexD controller has three operating modes:

- Normal
- Initialization
- Sleep

After a hardware reset, the LINFlexD controller is in sleep mode to reduce power consumption.

The transitions between these modes are shown in Figure 34-8. The software instructs LINFlexD to enter initialization mode or sleep mode by setting LINCR1[INIT] or LINCR1[SLEEP], respectively.

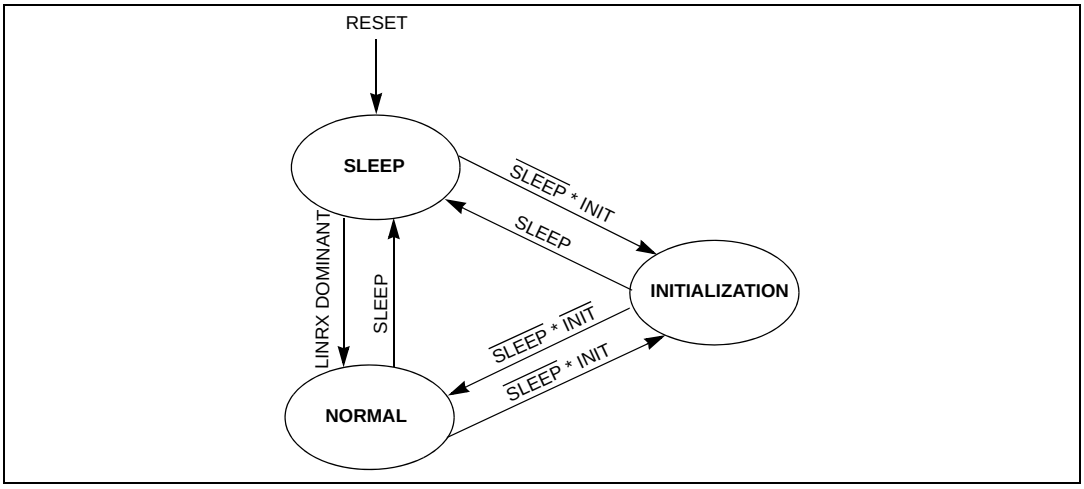


Figure 34-8. LINFlexD controller operating modes

In addition to these controller-level operating modes, the LINFlexD controller also supports several protocol-level modes:

- LIN mode:
  - Master mode
  - Slave mode
  - Slave mode with identifier filtering
  - Slave mode with automatic resynchronization
- UART mode
- Test modes:
  - Loopback mode
  - Self-test mode

These modes are discussed in detail in subsequent sections.

34.6 Controller-level operating modes

34.6.1 Initialization mode

The software initialization can be done while the hardware is in initialization mode. To enter or exit this mode, the software sets or clears LINCR1[INIT], respectively.

In initialization mode, all message transfers to and from the LIN bus are stopped and the LIN bus output (LINTX) is recessive.

Entering initialization mode does not change any of the configuration registers.

To initialize the LINFlexD controller, the software must:

- Select the desired mode (master, slave, or UART).
- Set up the baud rate register.
- If LIN slave mode with filter activation is selected, initialize the identifier list.

34.6.2 Normal mode

After initialization is complete, the software must clear LINCR1[INIT] to put the LINFlexD controller into normal mode.

34.6.3 Sleep (low-power) mode

To reduce power consumption, LINFlexD has a low-power mode called sleep mode. In this mode, the LINFlexD clock is stopped. Consequently, the LINFlexD will not update the status bits, but software can still access the LINFlexD registers.

To enter this mode, the software must set LINCR1[SLEEP].

LINFlexD can be awakened (exit sleep mode) in one of two ways:

- The software clears LINCR1[SLEEP].
- Automatic wakeup is enabled (LINCR1[AWUM] is set) and LINFlexD detects LIN bus activity (that is, if a wakeup pulse of 150 µs is detected on the LIN bus).

On LIN bus activity detection, hardware automatically performs the wakeup sequence by clearing LINCR1[SLEEP] if LINCR1[AWUM] is set. To exit from sleep mode if LINCR1[AWUM] is cleared, the software must clear LINCR1[SLEEP] when a wakeup event occurs.

34.7 LIN modes

34.7.1 Master mode

In master mode, the software uses the message buffer to handle the LIN messages.

Master mode is selected when LINCR1[MME] is set.

34.7.1.1 LIN header transmission

According to the LIN protocol, any communication on the LIN bus is triggered by the master sending a header. The header is transmitted by the master task while the data is transmitted by the slave task of a node.

To transmit a header with LINFlexD the application must set up the identifier, set up the data field length, and configure the message (direction and checksum type) in the BIDR register before requesting the header transmission by setting LINCR2[HTRQ].

34.7.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the software must provide the data to LINFlexD before requesting the header transmission. The software stores the data in the message buffer BDR. According to the data field length LINFlexD transmits the data and the checksum. The software uses the BIDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the software sets this bit the response is sent by LINFlexD (publisher). Clearing this bit configures the message buffer as subscriber.

34.7.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlexD stores the data received from the slave in the message buffer and stores the message status in the LINSR.

34.7.1.4 Error detection and handling

LINFlexD is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

Table 34-1 lists the errors detected in master mode and the LINFlexD controller's response to these errors.

Table 34-1. Errors in master mode

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value.	• Stops the transmission of the frame after the corrupted bit. • Generates an interrupt if LINIER[BEIE] is set. • Returns to idle state.
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field).	If encountered during reception: • Discards the current frame. • Generates an interrupt if LINIER[FEIE] is set. • Returns immediately to idle state.
Checksum error	The computed checksum does not match the received checksum.	If encountered during reception: • Discards the current frame. • Generates an interrupt if LINIER[CEIE] is set. • Returns to idle state.
Response and frame timeout	See Section 34.12.1, 8-bit timeout counter, for more details.	

34.7.1.5 Overrun

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINCR1[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.

34.7.2 Slave mode

In slave mode the software uses the message buffer to handle the LIN messages.

Slave mode is selected when the LINCR1[MME] is cleared.

34.7.2.1 Data transmission (transceiver as publisher)

When LINFlexD receives the identifier, a receive interrupt is generated. The software must:

- Read the received ID in BIDR.
- Fill the BDRs.
- Specify the data field length using the BIDR[DFL] field.
- Trigger the data transmission by setting LINCR2[DTRQ].

One or several identifier filters can be configured for transmission by setting the DIR bits in the corresponding IFCRs and activated by setting one or several bits in IFER.

When at least one identifier filter is configured in transmission and activated, and if the received ID matches the filter, a specific transmit interrupt is generated.

Typically, the software has to copy the data from RAM locations to BDRL and BDRM. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter which matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer which points to the right data array in the RAM area and copy this data to BDRL and BDRM (see Figure 34-10).

Using a filter avoids the software having to configure the direction, data field length, and checksum type in the BDIR register. The software fills BDRL and BDRM and triggers the data transmission by setting LINCR2[DTRQ].

If LINFlexD cannot provide enough transmit identifier filters to handle all identifiers for which the software has to transmit data, then a filter can be configured in mask mode (refer to Section 34.7.3, Slave mode with identifier filtering) in order to manage several identifiers with one filter only.

34.7.2.2 Data reception (transceiver as subscriber)

When LINFlexD receives the identifier, a receive interrupt is generated. The software must:

- Read the received ID in BIDR.
- Specify the data field length using the BIDR[DFL] field before the reception of the stop bit of the first byte of data field.

When the checksum reception is completed, a receive interrupt is generated to allow the software to read the received data in the BDRs.

One or several identifier filters can be configured for reception by clearing the DIR bit in the corresponding IFCRs and activated by clearing one or several bits in IFER.

When at least one identifier filter is configured in reception and activated, and if the received ID matches the filter, a receive interrupt is generated after the checksum reception only.

Typically, the software has to copy the data from BDRL and BDRM to RAM locations. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the RAM area and copy this data from BDRL and BDRM to the RAM (see Figure 34-10).

Using a filter avoids the software reading the ID value in BIDR, and configuring the direction, data field length, and checksum type in BIDR.

If LINFlexD cannot provide enough receive identifier filters to handle all identifiers for which the software has to receive the data, then a filter can be configured in mask mode (see Section 34.7.3, Slave mode with identifier filtering) in order to manage several identifiers with one filter only.

34.7.2.3 Data discard

When LINFlexD receives the identifier, a receive interrupt is generated. If the received identifier does not concern the node, the software must set LINCR2[DDRQ]. LINFlexD returns to idle state.

34.7.2.4 Error detection and handling

Table 34-2 lists the errors detected in slave mode and the LINFlexD controller's response to these errors.

Table 34-2. Errors in slave mode

Error	Description	LINFlexD response to error
Bit error	During transmission, the value read back from the bus differs from the transmitted value.	• Stops the transmission of the frame after the corrupted bit. • Generates an interrupt if LINIER[BEIE] is set. • Returns to idle state.
Framing error	A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field).	If encountered during reception: • Discards the current frame. • Generates an interrupt if LINIER[FEIE] is set. • Returns immediately to idle state.

Table 34-2. Errors in slave mode (continued)

Error	Description	LINFlexD response to error
Checksum error	The computed checksum does not match the received checksum.	If encountered during reception: • Discards the received frame. • Generates an interrupt if LINIER[CEIE] is set. • Returns to idle state.
Header error	An error occurred during header reception (break delimiter error, inconsistent sync field, header timeout).	If encountered during header reception, a break field error, an inconsistent sync field, or a timeout: • Discards the header. • Generates an interrupt if LINIER[HEIE] is set. • Returns to idle state.

34.7.2.5 Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid break field and break delimiter come before the end of the current header, or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

34.7.2.6 Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

34.7.2.7 Overrun

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the receive message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINCR1[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.

34.7.3 Slave mode with identifier filtering

In the LIN protocol, the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. When a slave node receives a header, it decides — depending on the identifier value — whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlexD controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources which would otherwise be needed by software for filtering.

The filtering is accomplished through the use of IFCRs. These registers have the names IFCR0 through IFCR7. This section also uses the nomenclature $IFCR_{2n}$ and $IFCR_{2n+1}$; in this nomenclature, *n* is an integer, and the corresponding IFCR is calculated using the formula in the subscript.

34.7.3.1 Filter submodes

Usually each of the eight IFCRs is used to filter one dedicated identifier, but this means that the LINFlexD controller could filter a maximum of eight identifiers. In order to be able to handle more identifiers, the filters can be configured to operate as masks.

Table 34-3 describes the two available filter submodes.

Table 34-3. Filter submodes

Submode	Description
Identifier list	Both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register. This is the default submode for the LINFlexD controller.
Mask	The identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care.”

The bit mapping and register organization in these two submodes is shown in Figure 34-9.

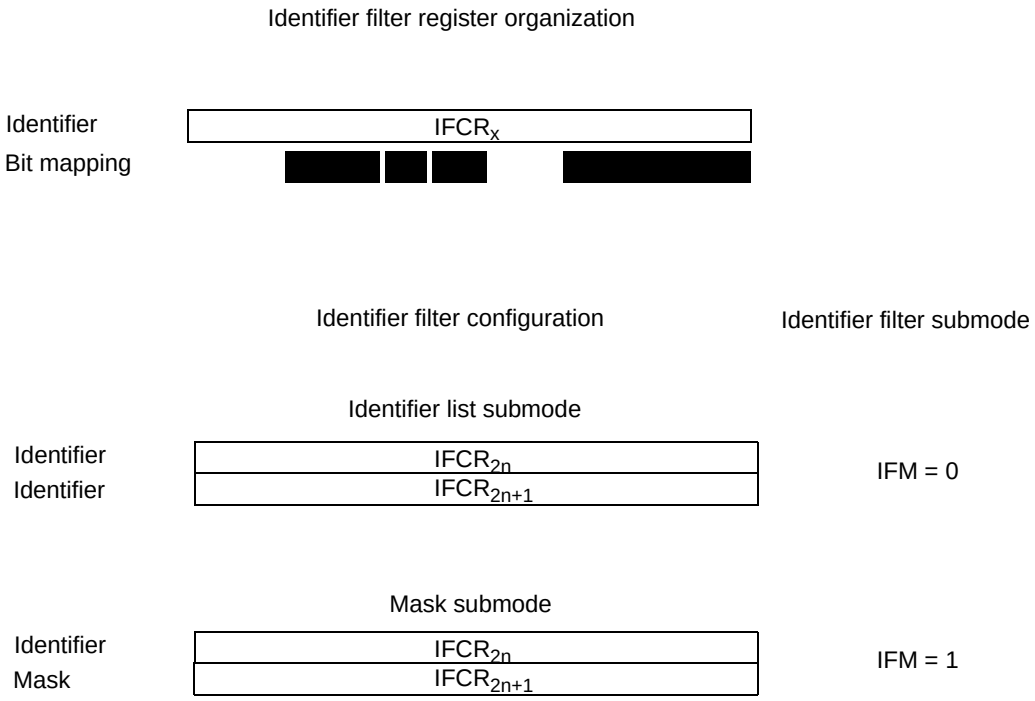


Figure 34-9. Filter configuration — register organization

34.7.3.2 Identifier filter submode configuration

The identifier filters are configured in the IFCR registers. To configure an identifier filter, the filter must first be deactivated by clearing the corresponding bit in the IFER[FACT] field. The submode (identifier list or mask) for the corresponding IFCR register is configured by the IFMR[IFM] field. For each filter, the IFCR register is used to configure:

- The ID or mask
- The direction (transmit or receive)
- The data field length
- The checksum type

If no filter is active, a receive interrupt is generated on any received identifier event.

If at least one active filter is configured as transmit, all received identifiers matching this filter generate a transmit interrupt.

If at least one active filter is configured as receive, all received identifiers matching this filter generate a receive interrupt.

If no active filter is configured as receive, all received identifiers not matching transmit filter(s) generate a receive interrupt.

Further details are provided in Table 34-4 and Figure 34-10.

Table 34-4. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as transmit	Number of active filters configured as receive	Interrupt vector
0	0	0	Receive interrupt on all IDs
a (a > 0)	a	0	• Transmit interrupt on IDs matching the filters • Receive interrupt on all other IDs if BF bit is set, no receive interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	• Transmit interrupt on IDs matching the transmit filters • Receive interrupt on IDs matching the receive filters • All other IDs discarded (no interrupt)
b (b > 0)	0	b	• Receive interrupt on IDs matching the filters • Transmit interrupt on all other IDs if BF bit is set, no transmit interrupt if BF bit is reset

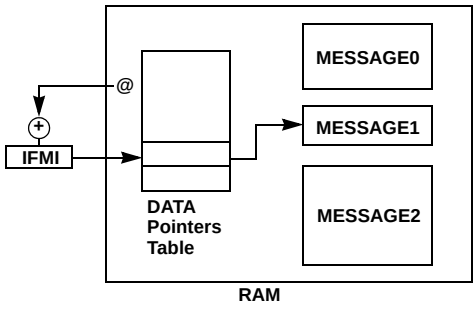


Figure 34-10. Identifier match index

34.7.4 Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in slave mode if $f_{ipg\_clock\_lin}$ tolerance is greater than 1.5%. This feature compensates an $f_{ipg\_clock\_lin}$ deviation up to 14%, as specified in the LIN standard.

This mode is similar to slave mode as described in Section 34.7.2, Slave mode, with the addition of automatic resynchronization enabled by the LINCR1[LASE] bit. In this mode LINFlexD adjusts the fractional baud rate generator after each sync field reception.

34.7.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN break, the time duration between five falling edges on RDI is sampled on $f_{ipg\_clock\_lin}$ as shown in Figure 34-11. Then the LFDIV value (and its associated LINIBRR and LINFBRR registers) are automatically updated at the end of the fifth falling edge. During LIN sync field measurement, the LINFlexD state machine is stopped and no data is transferred to the data register.

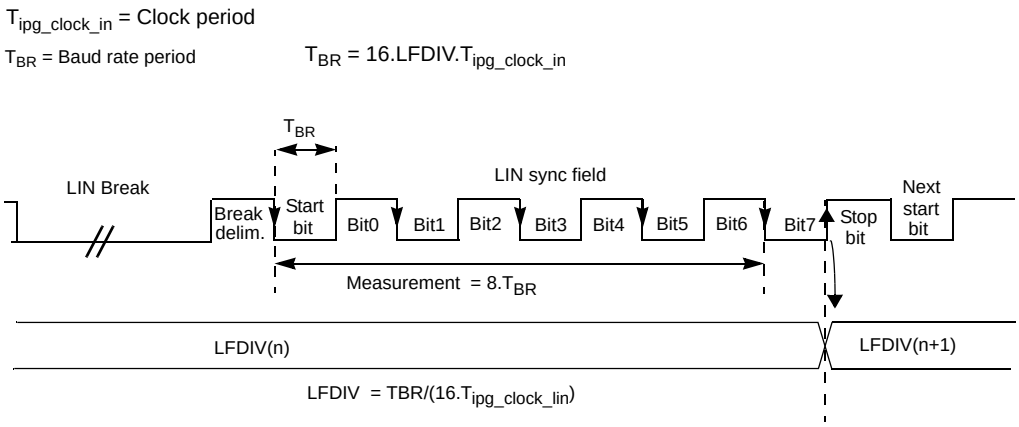


Figure 34-11. LIN sync field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 20 bits in the LINIBRR register and the fraction is coded on 4 bits in the LINFBRR register.

If LINCR1[LASE] is set, LFDIV is automatically updated at the end of each LIN sync field.

Three registers are used internally to manage the auto-update of the LINFlexD divider (LFDIV):

- LFDIV_NOM (nominal value written by software at LINIBRR and LINFBRR addresses)
- LFDIV_MEAS (results of the Field Sync measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break, or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV_NOM.

34.7.4.2 Deviation error on the sync field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN sync field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the sync field:

- If D1 > 14.84%, LHE is set.
- If D1 < 14.06%, LHE is not set.
- If 14.06% < D1 < 14.84%, LHE can be either set or reset depending on the dephasing between the signal on LINFlexD_RX pin the $f_{ipg\_clock\_lin}$ clock.

The second check is based on a measurement of time between each falling edge of the sync field:

- If D2 > 18.75%, LHE is set.
- If D2 < 15.62%, LHE is not set.
- If 15.62% < D2 < 18.75%, LHE can be either set or reset depending on the dephasing between the signal on LINFlexD_RX pin the $f_{ipg\_clock\_lin}$ clock.

Note that the LINFlexD does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full sync byte.

34.8 Test modes

The LINFlexD controller includes two test modes:

- Loopback mode
- Self-test mode

They can be selected by the LBKM and SFTM bits in the LINCR1 register. These bits must be configured while LINFlexD is in initialization mode. After one of the two test modes has been selected, LINFlexD must be started in Normal mode.

34.8.1 Loopback mode

LINFlexD can be put in loopback mode by setting LINCR1[LBKM]. In loopback mode, the LINFlexD treats its own transmitted messages as received messages. This is illustrated in Figure 34-12.

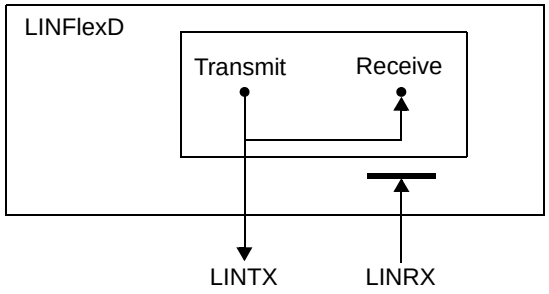


Figure 34-12. LINFlexD in loopback mode

This mode is provided for self-test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlexD performs an internal feedback from its transmit output to its receive input. The actual value of the LINRX input pin is disregarded by the LINFlexD. The transmitted messages can be monitored on the LINTX pin.

34.8.2 Self-test mode

LINFlexD can be put in self-test mode by setting LINCR1[LBKM] and LINCR1[SFTM]. This mode can be used for a hot self-test, meaning the LINFlexD can be tested as in loopback mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlexD and the LINTX pin is held recessive. This is illustrated in Figure 34-13.

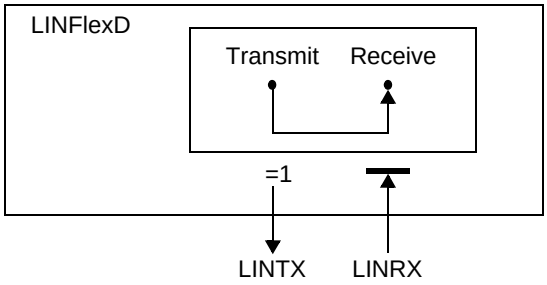


Figure 34-13. LINFlexD in self-test mode

34.9 UART mode

The main features of UART mode are presented in Section 34.2.2, UART mode features.

34.9.1 Data frame structure

34.9.1.1 8-bit data frame

The 8-bit UART data frame is shown in Figure 34-14. The 8th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

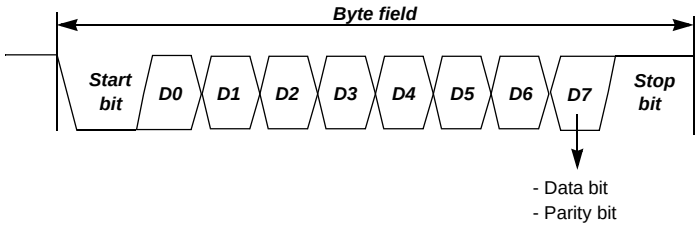


Figure 34-14. UART mode 8-bit data frame

34.9.1.2 9-bit data frame

The 9-bit UART data frame is shown in Figure 34-15. The 9th bit is a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

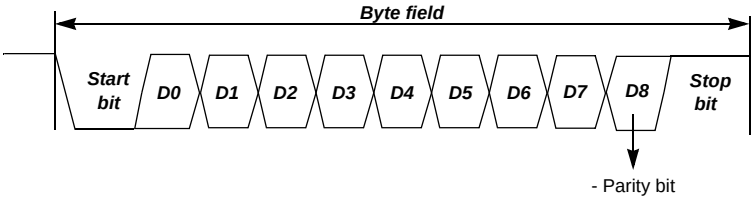


Figure 34-15. UART mode 9-bit data frame

34.9.1.3 16-bit data frame

The 16-bit UART data frame is shown in Figure 34-16. The 16th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

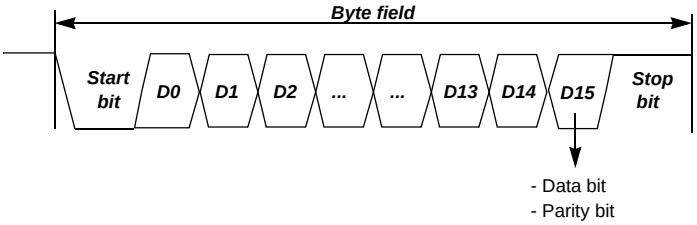


Figure 34-16. UART mode 16-bit data frame

34.9.1.4 17-bit data frame

The 17-bit UART data frame is shown in Figure 34-17. The 17th bit is the parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

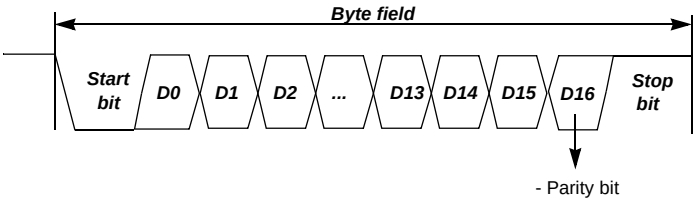


Figure 34-17. UART mode 17-bit data frame

34.9.2 Buffer

The 8-byte buffer is divided into two parts — one for receiver and one for transmitter — as shown in Table 34-5.

Table 34-5. UART buffer structure

BDR	UART mode
0	Tx0
1	Tx1
2	Tx2
3	Tx3
4	Rx0
5	Rx1
6	Rx2
7	Rx3

For 16-bit frames, the lower 8 bits are written in BDR0 and the upper 8 bits are written in BDR1.

34.9.3 UART transmitter

In order to start transmission in UART mode, the UARTCR[UART] and UARTCR[TXEN] bits must be set. Transmission starts when BDR0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by the UARTCR[TDFLTFC] field (see Table 34-18).

The transmit buffer size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 halfwords when UARTCR[WL1] = 1

Therefore, the maximum transmission that can be triggered is 4 bytes (2 halfwords). After the programmed number of bytes has been transmitted, the UARTSR[DTFTFF] flag is set. If the UARTCR[TXEN] field is cleared during a transmission, the current transmission is completed, but no further transmission can be invoked. The buffer can be configured in FIFO mode (mandatory when DMA transmit is enabled) by setting UARTCR[TFBM].

The access to the BDRL register is shown in Table 34-6.

Table 34-6. BDRL access in UART mode

Access	Mode[1]	Word length[2]	IPS operation result
Write byte0	FIFO	Byte	OK
Write byte1-2-3	FIFO	Byte	IPS transfer error
Write halfword0-1	FIFO	Byte	IPS transfer error
Write word	FIFO	Byte	IPS transfer error
Write byte0-1-2-3	FIFO	Halfword	IPS transfer error
Write halfword0	FIFO	Halfword	OK
Write halfword1	FIFO	Halfword	IPS transfer error
Write word	FIFO	Halfword	IPS transfer error

Table 34-6. BDRL access in UART mode (continued)

Access	Mode[1]	Word length[2]	IPS operation result
Read byte0-1-2-3	FIFO	Byte/halfword	IPS transfer error
Read halfword0-1	FIFO	Byte/halfword	IPS transfer error
Read word	FIFO	Byte/halfword	IPS transfer error
Write byte0-1-2-3	BUFFER	Byte/halfword	OK
Write halfword0-1	BUFFER	Byte/halfword	OK
Write word	BUFFER	Byte/halfword	OK
Read byte0-1-2-3	BUFFER	Byte/halfword	OK
Read halfword0-1	BUFFER	Byte/halfword	OK
Read word	BUFFER	Byte/halfword	OK

1 As specified by UARTCR[TFBM]

2 As specified by the WL1 and WL0 bits of UARTCR. In UART FIFO mode (UARTCR[TFBM] = 1), any read operation causes an IPS transfer error.

34.9.4 UART receiver

Reception of a data byte is started as soon as the software completes the following tasks in order:

1. Exits initialization mode.
2. Sets the UARTCR[RXEN] field.
3. Detects the start bit of the data frame.

There is a dedicated data buffer for received data bytes. Its size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 halfwords when UARTCR[WL1] = 1

After the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRFRFE] field is set. If the UARTCR[RXEN] field is cleared during a reception, the current reception is completed, but no further reception can be invoked until UARTCR[RXEN] is set again.

The buffer can be configured in FIFO mode (required when DMA receive is enabled) by setting UARTCR[RFBM].

The access to the BDRM register is shown in Table 34-7.

Table 34-7. BDRM access in UART mode

Access	Mode[1]	Word length[2]	IPS operation result
Read byte4	FIFO	Byte	OK
Read byte5-6-7	FIFO	Byte	IPS transfer error
Read halfword2-3	FIFO	Byte	IPS transfer error
Read word	FIFO	Byte	IPS transfer error

Table 34-7. BDRM access in UART mode (continued)

Access	Mode[1]	Word length[2]	IPS operation result
Read byte4-5-6-7	FIFO	halfword	IPS transfer error
Read halfword2	FIFO	halfword	OK
Read halfword3	FIFO	halfword	IPS transfer error
Read word	FIFO	halfword	IPS transfer error
Write byte4-5-6-7	FIFO	Byte/halfword	IPS transfer error
Write halfword2-3	FIFO	Byte/halfword	IPS transfer error
Write word	FIFO	Byte/halfword	IPS transfer error
Read byte4-5-6-7	BUFFER	Byte/halfword	OK
Read halfword2-3	BUFFER	Byte/halfword	OK
Read word	BUFFER	Byte/halfword	OK
Write byte4-5-6-7	BUFFER	Byte/halfword	IPS transfer error
Write halfword2-3	BUFFER	Byte/halfword	IPS transfer error
Write word	BUFFER	Byte/halfword	IPS transfer error

1 As specified by UARTCR[RFBM]

2 As specified by the WL1 and WL0 bits of UARTCR

Table 34-8 lists some common scenarios, controller responses, and suggestions when the LINFlexD controller is acting as a UART receiver.

Table 34-8. UART receiver scenarios

Scenario	Responses and suggestions
The software does not know (in advance) how many bytes will be received.	Do not program UARTCR[RDFLRFC] in advance. When this field is zero (as it is after reset), reception occurs on a byte-by-byte basis. Therefore, the state machine will move to IDLE state after each byte is received.
UARTCR[RDFLRFC] is programmed for a certain number of bytes received, but the actual number of bytes received is smaller.	The reception will hang. In this case, the software must monitor the UARTSR[TO] field, and move to IDLE state by setting LINCR1[SLEEP].
A STOP request arrives before the reception is completed.	The request is acknowledged only after the programmed number of data bytes are received. In other words, the STOP request is not serviced immediately. In this case, the software must monitor the UARTSR[TO] field and move the state machine to IDLE state as appropriate. The stop request will be serviced only after this is complete.
A parity error occurs during the reception of a byte.	The corresponding UARTSR[PE*n*] field is set. No interrupt is generated.

Table 34-8. UART receiver scenarios (continued)

Scenario	Responses and suggestions
A framing error occurs during the reception of a byte.	• UARTSR[FE] is set. • If LINIER[FEIE] = 1, an interrupt is generated. This interrupt is helpful in identifying which byte has the framing error, since there is only one register bit for framing errors.
A new byte has been received, but the last received frame has not been read from the buffer (UARTSR[RMB] has not yet been cleared by the software).	• An overrun error will occur (UARTSR[BOF] will be set). • One message will be lost (depending on the setting of LINCR[RBLM]). • An interrupt is generated if LINIER[BOIE] is set.

NOTE

When the LINFlexD is configured in UART Receive (Rx) FIFO mode, the Buffer Overrun Flag (BOF) bit of the UART Mode Status Register (UARTSR) register is cleared in the subsequent clock cycle after being asserted.

User software cannot poll the BOF to detect an overflow.

The LINFlexD Error Combined Interrupt can still be triggered by the buffer overrun. This interrupt is enabled by setting the Buffer Overrun Error Interrupt Enable (BOIE) bit in the LIN Interrupt enable register (LINIER). However, the BOF bit will be cleared when the interrupt routine is entered, preventing the user from identifying the source of error.

Buffer overrun errors in UART FIFO mode can be detected by enabling only the Buffer Overrun Interrupt Enable (BOIE) in the LIN interrupt enable register (LINIER).

34.10 Memory map and register description

Table 34-9 shows the base addresses for the LINFlexD modules. These addresses are not affected by Lock Step Mode (LSM) or Decoupled Parallel Mode (DPM). Table 34-10 shows the LINFlexD memory/register map.

Table 34-9. LINFlexD module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM) Decoupled Parallel Mode (DPM)	LINFlexD_0	0xFFE4_0000
	LINFlexD_1	0xFFE4_4000
	LINFlexD_2	0xFFE4_8000
	LINFlexD_3	0xFFE4_C000

Table 34-10. LINFlexD memory map

Offset from LINFlexD_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	LIN Control Register 1 (LINCR1)	R/W	0x0000_00U2[3]	on page 1237
0x0004	LIN Interrupt Enable Register (LINIER)	R/W	0x0000_0000	on page 1240
0x0008	LIN Status Register (LINSR)	R/W	0x0000_0040	on page 1241
0x000C	LIN Error Status Register (LINESR)	R/W	0x0000_0000	on page 1244
0x0010	UART Mode Control Register (UARTCR)	R/W	0x0000_0000	on page 1245
0x0014	UART Mode Status Register (UARTSR)	R/W	0x0000_0000	on page 1247
0x0018	LIN Timeout Control Status Register (LINTCSR)	R/W	0x0000_0200	on page 1249
0x001C	LIN Output Compare Register (LINOCR)	R/W	0x0000_FFFF	on page 1250
0x0020	LIN Timeout Control Register (LINTOCR)	R/W	0x0000_0EUB[3]	on page 1251
0x0024	LIN Fractional Baud Rate Register (LINFBRR)	R/W	0x0000_0000	on page 1251
0x0028	LIN Integer Baud Rate Register (LINIBRR)	R/W	0x0000_0000	on page 1252
0x002C	LIN Checksum Field Register (LINCFR)	R/W	0x0000_0000	on page 1253
0x0030	LIN Control Register 2 (LINCR2)	R/W	0x0000_U000[3]	on page 1253
0x0034	Buffer Identifier Register (BIDR)	R/W	0x0000_0000	on page 1254
0x0038	Buffer Data Register Least Significant (BDRL)	R/W	0x0000_0000	on page 1255
0x003C	Buffer Data Register Most Significant (BDRM)	R/W	0x0000_0000	on page 1256
0x0040	Identifier Filter Enable Register (IFER)	R/W	0x0000_0000	on page 1257
0x0044	Identifier Filter Match Index (IFMI)	R/W	0x0000_0000	on page 1257
0x0048	Identifier Filter Mode Register (IFMR)	R/W	0x0000_0000	on page 1258
0x004C	Identifier Filter Control Register 0 (IFCR0)	R/W	0x0000_0000	on page 1258
0x0050	Identifier Filter Control Register 1 (IFCR1)	R/W	0x0000_0000	on page 1258
0x0054	Identifier Filter Control Register 2 (IFCR2)	R/W	0x0000_0000	on page 1258
0x0058	Identifier Filter Control Register 3 (IFCR3)	R/W	0x0000_0000	on page 1258
0x005C	Identifier Filter Control Register 4 (IFCR4)	R/W	0x0000_0000	on page 1258
0x0060	Identifier Filter Control Register 5 (IFCR5)	R/W	0x0000_0000	on page 1258
0x0064	Identifier Filter Control Register 6 (IFCR6)	R/W	0x0000_0000	on page 1258
0x0068	Identifier Filter Control Register 7 (IFCR7)	R/W	0x0000_0000	on page 1258
0x006C	Identifier Filter Control Register 8 (IFCR8)	R/W	0x0000_0000	on page 1258
0x0070	Identifier Filter Control Register 9 (IFCR9)	R/W	0x0000_0000	on page 1258
0x0074	Identifier Filter Control Register 10 (IFCR10)	R/W	0x0000_0000	on page 1258
0x0078	Identifier Filter Control Register 11 (IFCR11)	R/W	0x0000_0000	on page 1258
0x007C	Identifier Filter Control Register 12 (IFCR12)	R/W	0x0000_0000	on page 1258

Table 34-10. LINFlexD memory map (continued)

Offset from LINFlexD_BASE	Register	Access[1]	Reset Value[2]	Location
0x0080	Identifier Filter Control Register 13 (IFCR13)	R/W	0x0000_0000	on page 1258
0x0084	Identifier Filter Control Register 14 (IFCR14)	R/W	0x0000_0000	on page 1258
0x0088	Identifier Filter Control Register 15 (IFCR15)	R/W	0x0000_0000	on page 1258
0x008C	Global Control Register (GCR)	R/W	0x0000_0000	on page 1259
0x0090	UART Preset Timeout Register (UARTPTO)	R/W	0x0000_0FFF	on page 1260
0x0094	UART Current Timeout Register (UARTCTO)	R	0x0000_0000	on page 1261
0x0098	DMA Transmit Enable Register (DMATXE)	R/W	0x0000_0000	on page 1261
0x009C	DMA Receive Enable Register (DMARXE)	R/W	0x0000_0000	on page 1262
0x00A0–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 One or more bits (indicated by “U”) are of indeterminate value at Reset. See register for more information.

34.10.1 LIN Control Register 1 (LINCR1)

Address: Base + 0x00 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CCD[1]	CFD[1]	LASE[1]	AWUM[1]	MBL[1]				BF[1]	SFTM[1]	LBKM[1]	MME[1]	SBDT[1]	RBLM[1]	SLEEP	INIT
Reset	0	0	0	0	0	0	0	0	1	0	0	0/1[2]	0	0	1	0

1 These fields are writable only in initialization mode (LINCR1[INIT] = 1).

2 Resets to 0 in slave mode and to 1 in master mode.

Figure 34-18. LIN Control Register 1 (LINCR1)

Table 34-11. LINCR1 field descriptions

Field	Description
CCD	Checksum Calculation Disable This bit disables the checksum calculation (see Table 34-12). 0 Checksum calculation is done by hardware. When this bit is reset the LINCFR register is read-only. 1 Checksum calculation is disabled. When this bit is set the LINCFR register is read/write. User can program this register to send a software-calculated CRC (provided CFD is reset). **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.

Table 34-11. LINCR1 field descriptions (continued)

Field	Description
CFD	Checksum Field Disable This bit disables the checksum field transmission (see Table 34-12). 0 Checksum field is sent after the required number of data bytes is sent. 1 No checksum field is sent. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
LASE	LIN Slave Automatic Resynchronization Enable 0 Automatic resynchronization disable. 1 Automatic resynchronization enable. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
AWUM	Automatic Wake-Up Mode This bit controls the behavior of the LINFlexD hardware during sleep mode. 0 The sleep mode is exited on software request by clearing the LINCR[SLEEP] bit. 1 The sleep mode is exited automatically by hardware on receive dominant state detection. The LINCR[SLEEP] bit is cleared by hardware whenever LINSR[WUF] = 1. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
MBL	LIN Master Break Length These bits indicate the Break length in master mode (see Table 34-13). **Note:** These bits can be written in initialization mode only. They are read-only in normal or sleep mode.
BF	Bypass filter 0 No interrupt if ID does not match any filter. 1 A receive interrupt is generated on ID not matching any filter. Notes: • If no filter is activated, this bit is reserved. • This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
SFTM	Self-test Mode This bit controls the self-test mode. For more details, see Section 34.8.2, Self-test mode. 0 Self-test mode disable. 1 Self-test mode enable. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
LBKM	Loopback Mode This bit controls the loopback mode. For more details, see Section 34.8.1, Loopback mode. 0 Loopback mode disable. 1 Loopback mode enable. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
MME	Master Mode Enable 0 Slave mode enable. 1 Master mode enable. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
SBDT	Slave Mode Break Detection Threshold 0 11-bit break. 1 10-bit break. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.

Table 34-11. LINCR1 field descriptions (continued)

Field	Description
RBLM	Receive Buffer Locked Mode 0 Receive Buffer not locked on overrun. Once the slave Receive Buffer is full the next incoming message overwrites the previous one. 1 Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. **Note:** This bit can be written in initialization mode only. It is read-only in normal or sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlexD to enter sleep mode. This bit is cleared by software to exit sleep mode or by hardware if LINCR1[AWUM] = 1 and LINSR[WUF] = 1 (see Table 34-14).
INIT	Initialization Request The software sets this bit to switch hardware into initialization mode. If the SLEEP bit is reset, LINFlexD enters normal mode when clearing the INIT bit (see Table 34-14).

Table 34-12. Checksum bits configuration

CFD	CCD	LINCFR	Checksum sent
1	1	Read/Write	None
1	0	Read-only	None
0	1	Read/Write	Programmed in LINCFR by bits CF[0:7]
0	0	Read-only	Hardware calculated

Table 34-13. LIN master break length selection

MBL	Length	MBL	Length
0000	10-bit	1000	18-bit
0001	11-bit	1001	19-bit
0010	12-bit	1010	20-bit
0011	13-bit	1011	21-bit
0100	14-bit	1100	22-bit
0101	15-bit	1101	23-bit
0110	16-bit	1110	36-bit
0111	17-bit	1111	50-bit

Table 34-14. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

34.10.2 LIN Interrupt Enable Register (LINIER)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZIE	OCIE	BEIE	CEIE	HEIE	0	0	FEIE	BOIE	LSIE	WUIE	DBFIE	DBEIETOIE	DRIE	DTIE	HRIE
W	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-19. LIN Interrupt Enable Register (LINIER)

Table 34-15. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0 No interrupt when SZF bit in LINESR or UARTSR is set. 1 Interrupt generated when SZF bit in LINESR or UARTSR is set.
OCIE	Output Compare Interrupt Enable 0 No interrupt when OCF bit in LINESR or UARTSR is set. 1 Interrupt generated when OCF bit in LINESR or UARTSR is set.
BEIE	Bit Error Interrupt Enable 0 No interrupt when LINESR[BEF] = 1. 1 Interrupt generated when LINESR[BEF] = 1.
CEIE	Checksum Error Interrupt Enable 0 No interrupt on checksum error. 1 Interrupt generated when checksum error flag (LINESR[CEF]) is set.
HEIE	Header Error Interrupt Enable 0 No interrupt on break delimiter error, sync field error, ID field error. 1 Interrupt generated on break delimiter error, sync field error, ID field error.
FEIE	Framing Error Interrupt Enable 0 No interrupt on framing error. 1 Interrupt generated on framing error.
BOIE	Buffer Overrun Interrupt Enable 0 No interrupt on buffer overrun. 1 Interrupt generated on buffer overrun.
LSIE	LIN State Interrupt Enable 0 No interrupt on LIN state change. 1 Interrupt generated on LIN state change. This interrupt can be used for debugging purposes. It has no status flag but is reset when writing 1111 into the LIN state bits in LINSR.

Table 34-15. LINIER field descriptions (continued)

Field	Description
WUIE	Wakeup Interrupt Enable 0 No interrupt when WUF bit in LINSR or UARTSR is set. 1 Interrupt generated when WUF bit in LINSR or UARTSR is set.
DBFIE	Data Buffer Full Interrupt Enable 0 No interrupt when buffer data register is full. 1 Interrupt generated when data buffer register is full.
DBEIETOIE	Data Buffer Empty Interrupt Enable / Timeout Interrupt Enable 0 No interrupt when buffer data register is empty. 1 Interrupt generated when data buffer register is empty. **Note:** An interrupt is generated if this bit is set and one of the following is true: LINFlexD is in LIN mode and LINSR[DBEF] is set. LINFlexD is in UART mode and UARTSR[TO] is set.
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt is generated when data reception is completed. 1 An interrupt generated when the data reception complete flag LINSR[DRF] or UARTSR[DRFRFE] is set. **Note:** In UART FIFO mode, the UARTSR[DRFRFE] bit is immediately set, indicating that the receive FIFO is empty. If the DRIE bit is set, it continuously triggers the receive interrupt and causes the DMA transfer to fail. This bit should not be set in UART FIFO mode.
DTIE	Data Transmitted Interrupt Enable 0 No interrupt when data transmission is completed. 1 Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR register.
HRIE	Header Received Interrupt Enable 0 No interrupt when a valid LIN header has been received. 1 Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR register is set.

34.10.3 LIN Status Register (LINSR)

Address: Base + 0x08 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LINS				0	0	RMB	0	RBSY	RPS	WUF	DBFF	DBEF	DRF	DTF	HRF
W	w1c	w1c	w1c	w1c			w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 34-20. LIN Status Register (LINSR)

Table 34-16. LINSR field descriptions

Field	Description
LINS	LIN state LIN mode states description 0000: Sleep mode LINFlexD is in sleep mode to save power consumption. 0001 Initialization mode LINFlexD is in initialization mode. 0010 Idle This state is entered on several events: - SLEEP bit and INIT in LINCR1 register have been cleared by software - A falling edge has been received on receive pin and AWUM bit is set - The previous frame reception or transmission has been completed or aborted 0011 Break In slave mode, a falling edge followed by a dominant state has been detected. Receiving Break. In slave mode, in case of error new LIN state can be either idle or break depending on last bit state. If last bit is dominant new LIN state is break, otherwise idle. In master mode, break transmission ongoing. 0100 Break Delimiter In slave mode, a valid Break has been detected. Refer to LINCR1 register for break length configuration (10-bit or 11-bit). Waiting for a rising edge. In master mode, break transmission has been completed. break delimiter transmission is ongoing. 0101 Sync Field In slave mode, a valid break delimiter has been detected (recessive state for at least one bit time). receiving sync field. In master mode, sync field transmission is ongoing. 0110 Identifier Field In slave mode, a valid sync field has been received. Receiving ID field. In master mode, identifier transmission is ongoing. 0111 Header reception/transmission completed In slave mode, a valid header has been received and identifier field is available in BIDR. In master mode, header transmission is completed. 1000 Data reception/transmission Response reception/transmission is ongoing. 1001 Checksum Data reception/transmission completed. Checksum reception/transmission ongoing. In UART mode, only the following states are flagged by the LIN state bits: - Init - Sleep - Idle - Data transmission/reception **Note:** This LINS bit is invalid in UART mode.
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in initialization mode.
RBSY	Receiver Busy Flag 0 Receiver is Idle. 1 Reception ongoing. **Note:** In slave mode, after header reception, if DIR bit in BIDR is reset and reception starts then this bit is set. In this case, user cannot set DTRQ bit in LINCR2.

Table 34-16. LINSR field descriptions (continued)

Field	Description
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wakeup Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin when • Slave is in sleep mode, • Master is in sleep mode or idle state. This bit must be cleared by software. It is reset by hardware in initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in initialization mode. **Note:** This flag is not set in case of bit error or framing error.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in initialization mode. **Note:** This flag is not set in case of bit error if IOBE bit is reset.
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in initialization mode and at end of completed or aborted frame. **Note:** If filters are enabled, this bit is set only when identifier software filtering is required, that is to say: • All filters are inactive and BF bit in LINCR1 is set • No match in any filter and BF bit in LINCR1 is set • Transmit filter match

34.10.4 LIN Error Status Register (LINESR)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	BEF	CEF	SFEF	BDEF	IDPEF	FEF	BOF	0	0	0	0	0	0	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-21. LIN Error Status Register (LINESR)

Table 34-17. LINESR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	Output Compare Flag 0 No output compare event occurred. 1 The contents of the counter match the contents of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and LINTCSR[IOT] = 1, LINFlexD moves to Idle state. If LINTCSR[LTOM] = 1, OCF is reset by hardware in initialization mode. If LINTCSR[LTOM] = 0, then OCF maintains its status regardless of mode.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a bit error. This error can occur during response field transmission (slave and master modes) or during header transmission (in master mode). This bit is cleared by software.
CEF	Checksum error Flag This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum. This bit is cleared by software. **Note:** This bit is never set if either the LINCR1[CCD] or LINCR1[CFD] bit is set.
SFEF	Sync Field Error Flag This bit is set by hardware and indicates that a sync field error occurred (inconsistent sync field).
BDEF	Break Delimiter Error Flag This bit is set by hardware and indicates that the received break delimiter is too short (less than one bit time).
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that an identifier parity error occurred. **Note:** Header interrupt is triggered when the SFEF, BDEF, or IDPEF bit is set and LINIER[HEIE] bit in is set.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (master or slave mode) or during reception of sync field or identifier field in slave mode.

Table 34-17. LINESR field descriptions (continued)

Field	Description
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If the LINCR1[RBLM] bit is set, then the new byte received is discarded. If LINCR1[RBLM] is cleared, then the new byte overwrites the buffer. This bit can be cleared by software.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

34.10.5 UART Mode Control Register (UARTCR)

Address: Base + 0x10 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TDFLTFC[1]			RDFLRFC[1]			RFBM[2]	TFBM[2]	WL1	PC1[2]	RXEN	TXEN	PC0[2]	PCE[2]	WL0	UART
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These fields are read/write in UART buffer mode and read-only in other modes.

2 These fields are writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-22. UART Mode Control Register (UARTCR)

Table 34-18. UARTCR field descriptions

Field	Description
TDFLTFC	Transmitter data field length / transmit FIFO counter This field has one of two functions depending on the mode of operation as follows: • When LINFlexD is in UART buffer mode (TFBM = 0), TDFLTFC defines the number of bytes to be transmitted. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit): 0bX00: 1 byte 0bX01: 2 bytes 0bX10: 3 bytes 0bX11: 4 bytes When the UART data length is configured as halfword (WL = 0b10 or 0b11), the only valid values for TDFLTFC are 0b001 and 0b011. • When LINFlexD is in UART FIFO mode (TFBM = 1), TDFLTFC contains the number of entries (bytes) of the transmit FIFO. The field is read-only in this configuration. The permissible values are as follows: 0b000: Empty 0b001: 1 byte 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes All other values are reserved. **Note:** This field is meaningful and can be programmed only when the UART bit is set.
RDFLRFC	Receiver data field length / receive FIFO counter This field has one of two functions depending on the mode of operation as follows: • When LINFlexD is in UART buffer mode (RFBM = 0), RDFLRFC defines the number of bytes to be received. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit): 0bX00: 1 byte 0bX01: 2 bytes 0bX10: 3 bytes 0bX11: 4 bytes When the UART data length is configured as halfword (WL = 0b10 or 0b11), the only valid values for RDFLRFC are 0b001 and 0b011. • When LINFlexD is in UART FIFO mode (RFBM = 1), RDFLRFC contains the number of entries (bytes) of the receive FIFO. The field is read-only in this configuration. The permissible values are as follows: 0b000: Empty 0b001: 1 byte 0b010: 2 bytes 0b011: 3 bytes 0b100: 4 bytes All other values are reserved. **Note:** This field is meaningful and can be programmed only when the UART bit is set.
RFBM	Receive FIFO/buffer mode 0 Receive buffer mode enabled. 1 Receive FIFO mode enabled (mandatory in DMA receive mode). **Note:** This field can be programmed in initialization mode only when the UART bit is set.
TFBM	Transmit FIFO/buffer mode 0 Transmit buffer mode enabled. 1 Transmit FIFO mode enabled (mandatory in DMA transmit mode). **Note:** This field can be programmed in initialization mode only when the UART bit is set.

Table 34-18. UARTCR field descriptions (continued)

Field	Description
RXEN	Receiver Enable 0 Receiver disabled. 1 Receiver enabled. This field can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disabled. 1 Transmitter enabled. This field can be programmed only when the UART bit is set. **Note:** Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
PC	Parity Control 00 Parity sent is even. 01 Parity sent is odd. 10 A logical 0 is always transmitted/checked as parity bit. 11 A logical 1 is always transmitted/checked as parity bit. **Note:** This field can be programmed in initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disabled. 1 Parity transmit/check enabled. **Note:** This field can be programmed in initialization mode only when the UART bit is set.
WL	Word Length in UART Mode 00 7 data bits + parity. 01 8 data bits when PCE = 0 or 8 data bits + parity when PCE = 1. 10 15 data bits + parity. 11 16 data bits when PCE = 0 or 16 data bits + parity when PCE = 1. **Note:** This field can be programmed in initialization mode only when the UART bit is set.
UART	UART Mode Enable 0 LIN mode. 1 UART mode. This field can be programmed in initialization mode only.

34.10.6 UART Mode Status Register (UARTSR)

Address: Base + 0x14 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	TO	DRFRFE	DTFTFF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-23. UART Mode Status Register (UARTSR)

Table 34-19. UARTSR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	OCF Output Compare Flag 0 No output compare event occurred. 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. An interrupt is generated if the OCIE bit in LINIER register is set.
PE3	Parity Error Flag Rx3 This bit indicates if a parity error is detected in the corresponding received byte (Rx3). No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE2	Parity Error Flag Rx2 This bit indicates if a parity error is detected in the corresponding received byte (Rx2). No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE1	Parity Error Flag Rx1 This bit indicates if a parity error is detected in the corresponding received byte (Rx1). No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE0	Parity Error Flag Rx0 This bit indicates if a parity error is detected in the corresponding received byte (Rx0). No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in initialization mode.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit).
BOF	FIFO/Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the RMB bit is not cleared in UART buffer mode. In UART FIFO mode, this bit is set when there is a new byte and the receive FIFO is full. In UART FIFO mode, once receive FIFO is full, the new received message is discarded regardless of the value of LINCR1[RBLM]. If LINCR1[RBLM] = 1, the new byte received is discarded. If LINCR1[RBLM] = 0, the new byte overwrites buffer. This field can be cleared by writing a 1 to it. An interrupt is generated if LINIER[BOIE] is set.
RPS	LIN Receive Pin State This bit reflects the current status of LINRX pin for diagnostic purposes.

Table 34-19. UARTSR field descriptions (continued)

Field	Description
WUF	Wakeup Flag This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin in sleep mode. This bit must be cleared by software. It is reset by hardware in initialization mode. An interrupt is generated if LINIER[WUIE] = 1.
TO	Timeout The LINFlexD controller sets this field when a UART timeout occurs — that is, when the value of UARTCTO becomes equal to the preset value of the timeout (UARTPTO register setting). This field should be cleared by software. The GCR[SR] field should be used to reset the receiver FSM to idle state in case of UART timeout for UART reception depending on the application in both buffer and FIFO mode. An interrupt is generated when LINIER[DBEIETOIE] is set on the Error interrupt line in UART mode.
DRFRFE	Data reception completed flag / receive FIFO empty flag The LINFlexD controller sets this field as follows: • In UART buffer mode (RFBM = 0), it indicates that the number of bytes programmed in RDFL has been received. This field should be cleared by software. An interrupt is generated if LINIER[DRIE] is set. This field is set in case of framing error, parity error, or overrun. This field reflects the same value as in LINESR when in initialization mode and UART bit is set. • In UART FIFO mode (RFBM = 1), it indicates that the receive FIFO is empty. This field is a read-only field used internally by the DMA receive interface.
DTFTFF	Data transmission completed flag / transmit FIFO full flag The LINFlexD controller sets this field as follows: • In UART buffer mode (TFBM = 0), it indicates that the data transmission is completed. This field should be cleared by software. An interrupt is generated if LINIER[DTIE] is set. This field reflects the same value as in LINESR when in initialization mode and UART bit is set. • In UART FIFO mode (TFBM = 1), it indicates that the transmit FIFO is full. This field is a read-only field used internally by the DMA transmit interface.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

34.10.7 LIN Timeout Control Status Register (LINTCSR)

Address: Base + 0x18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	LTOM[1]	IOT[1]	TOCE[1]	CNT							
W						w1c	w1c	w1c								
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

1 These fields are writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-24. LIN Timeout Control Status Register (LINTCSR)

Table 34-20. LINTCSR field descriptions

Name	Description
LTOM	LIN timeout mode 0 LIN timeout mode (header, response, and frame timeout detection). 1 Output compare mode. **Note:** This bit can be set/cleared in initialization mode only.
IOT	Idle on Timeout 0 LIN state machine not reset to Idle on timeout event. 1 LIN state machine reset to Idle on timeout event. **Note:** This bit can be set/cleared in initialization mode only.
TOCE	Timeout counter enable 0 Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event. 1 Timeout counter enable. OCF bit is set if an output compare event occurs. **Note:** The TOCE bit is configurable by software in initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit.
CNT	Counter Value These bits indicate the LIN Timeout counter value.

34.10.8 LIN Output Compare Register (LINOCR)

Address: Base + 0x1C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OC2[1]								OC1[1]							
W	w1c[1]								w1c[1]							
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1 If LINTCSR[LTOM] = 1, these fields are read-only.

Figure 34-25. LIN Output Compare Register (LINOCR)

Table 34-21. LINOCR field descriptions

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of LINTCSR[CNT].
OC1	Output compare 1 value These bits contain the value to be compared to the value of LINTCSR[CNT].

34.10.9 LIN Timeout Control Register (LINTOCR)

Address: Base + 0x20 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	RTO				0	HTO[3]						
W																
Reset	0	0	0	0	1	1	1	0	0	0	0/1[1]	0/1[2]	1	1	0	0

1 Resets to 1 in slave mode and to 0 in master mode.
2 Resets to 0 in slave mode and to 1 in master mode.
3 HTO field can only be written in slave mode, LINCR1[MME] = 0.

Figure 34-26. LIN Timeout Control Register (LINTOCR)

Table 34-22. LINTOCR field descriptions

Field	Description
RTO	Response timeout value This register contains the response timeout duration (in bit time) for 1 byte. The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$.
HTO	Header timeout value This register contains the header timeout duration (in bit time). This value does not include the first 11 dominant bits of the break. The reset value depends on which mode LINFlexD is in. HTO can be written only for slave mode.

34.10.10 LIN Fractional Baud Rate Register (LINFBRR)

Address: Base + 0x24 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_F[1]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 This field is writable only in initialization mode, LINCR1[INIT] = 1.

Figure 34-27. LIN Fractional Baud Rate Register (LINFBRR)

Table 34-23. LINFBRR field descriptions

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlexD divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F/ 16. This register can be written in initialization mode only, LINCR1[INIT] = 1.

34.10.11 LIN Integer Baud Rate Register (LINIBRR)

Address: Base + 0x28 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_M[1]																			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] This field is writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-28. LIN Integer Baud Rate Register (LINIBRR)

Table 34-24. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa These bits define the LINFlexD divider (LFDIV) mantissa value (see Table 34-25). This register can be written in initialization mode only.

Table 34-25. Integer baud rate selection

DIV_M	Mantissa
0x0	LIN clock disabled
0x1	1
...	...
0xFFFFE	1048574
0xFFFFF	1048575

34.10.12 LIN Checksum Field Register (LINCFR)

Address: Base + 0x2C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-29. LIN Checksum Field Register (LINCFR)

Table 34-26. LINCFR field descriptions

Field	Description
CF	Checksum When LINCR1[CCD] is cleared, this field is read-only. When LINCR1[CCD] is set, this field is read/write. See Table 34-12.

34.10.13 LIN Control Register 2 (LINCR2)

Address: Base + 0x30 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	IOBE[1]	IOPE[1]	WURQ	DDRQ	DTRQ	ABRQ	HTRQ	0	0	0	0	0	0	0	0
W				w1c	w1c	w1c	w1c	w1c								
Reset	0	1	0/1[2]	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These fields are writable only in initialization mode (LINCR1[INIT] = 1).

2 Resets to 1 in slave mode and to 0 in master mode

Figure 34-30. LIN Control Register 2 (LINCR2)

Table 34-27. LINCR2 field descriptions

Field	Description
IOBE	Idle on Bit Error 0 Bit error does not reset LIN state machine. 1 Bit error reset LIN state machine. This bit can be set/cleared in initialization mode only (LINCR1[INIT] = 1).

Table 34-27. LINCR2 field descriptions (continued)

Field	Description
IOPE	Idle on Identifier Parity Error 0 Identifier Parity error does not reset LIN state machine. 1 Identifier Parity error reset LIN state machine. This bit can be set/cleared in initialization mode only (LINCR1[INIT] = 1).
WURQ	Wakeup Generation Request Setting this bit generates a wakeup pulse. It is reset by hardware when the wakeup character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in sleep mode. Software has to exit sleep mode before requesting a wakeup. Bit error is not checked when transmitting the wakeup request.
DDRQ	Data Discard Request Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlexD has moved to idle state. In slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.
DTRQ	Data Transmission Request Set by software in slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set. Cleared by hardware when the request has been completed or aborted or on an error condition. In master mode, this bit is set by hardware when DIR bit in BIDR is set and header transmission is completed.
ABRQ	Abort Request Set by software to abort the current transmission. Cleared by hardware when the transmission has been aborted. LINFlexD aborts the transmission at the end of the current bit. This bit can also abort a wakeup request. It can also be used in UART mode.
HTRQ	Header Transmission Request Set by software to request the transmission of the LIN header. Cleared by hardware when the request has been completed or aborted. This bit has no effect in UART mode.

34.10.14 Buffer Identifier Register (BIDR)

The Buffer Identifier Register (BIDR) contains the fields that identify a transaction and provide other information related to it.

All the fields in this register must be updated when an ID filter (enabled) in slave mode (transmit or receive) matches the ID received.

Address: Base + 0x34 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL						DIR	CCS	0	0	ID					
W							w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-31. Buffer Identifier Register (BIDR)

Table 34-28. BIDR field descriptions

Field	Description
DFL	Data Field Length These bits define the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1. Normally, LIN uses only DFL[0:2] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[0:2] and DFL[0:5]. DFL[3:5] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0 LINFlexD receives the data and copies it in the BDRs. 1 LINFlexD transmits the data from the BDRs.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

34.10.15 Buffer Data Register Least Significant (BDRL) register

Address: Base + 0x38 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA3								DATA2							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DATA1								DATA0							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-32. Buffer Data Register Least Significant (BDRL) register

Table 34-29. BDRL field descriptions

Field	Description
DATA3	Data Byte 3 Data byte 3 of the data field
DATA2	Data Byte 2 Data byte 2 of the data field
DATA1	Data Byte 1 Data byte 1 of the data field
DATA0	Data Byte 0 Data byte 0 of the data field

34.10.16 Buffer Data Register Most Significant (BDRM) register

Address: Base + 0x3C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA7								DATA6							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA5								DATA4							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-33. Buffer data register most significant (BDRM) register

Table 34-30. BDRM field descriptions

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field
DATA6	Data Byte 6 Data byte 6 of the data field
DATA5	Data Byte 5 Data byte 5 of the data field
DATA4	Data Byte 4 Data byte 4 of the data field

34.10.17 Identifier Filter Enable Register (IFER)

Address: Base + 0x40 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	FACT[1]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] This field is writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-34. Identifier Filter Enable Register (IFER)

Table 34-31. IFER field descriptions

Field	Description
FACT	Filter active The software sets the bit FACT[*x*] to activate the filter x in identifier list mode. In identifier mask mode bits FACT(2*n* + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2*n*. These bits can be set/cleared in initialization mode only.

34.10.18 Identifier Filter Match Index (IFMI) register

Address: Base + 0x44 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFMI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-35. Identifier Filter Match Index (IFMI) register

Table 34-32. IFMI field descriptions

Field	Description
IFMI	Filter match index This register contains the index corresponding to the received ID. It can be used to directly write or read the data in RAM (see Section 34.7.2, Slave mode, for more details). When no filter matches, IFMI = 0. When Filter *n* is matching, IFMI = *n* + 1.

34.10.19 Identifier Filter Mode Register (IFMR)

Address: Base + 0x48 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFM[1]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 This field is writable only in initialization mode (LINCR1[INIT] = 1)

Figure 34-36. Identifier Filter Mode Register (IFMR)

Table 34-33. IFMR field descriptions

Field	Description
IFM	Filter mode 0 Filters 2*n* and 2*n* + 1 are in identifier list mode. 1 Filters 2*n* and 2*n* + 1 are in mask mode (filter 2*n* + 1 is the mask for the filter 2*n*).

34.10.20 Identifier Filter Control Registers (IFCR0–IFCR15)

The function of these registers is different depending on which mode the LINFlexD controller is in, as described in Table 34-34.

Table 34-34. IFCR functionality based on mode

Mode	IFCR functionality
Identifier list	Each IFCR register acts as a filter.
Identifier mask	If *a* = (number of filters) / 2, and *n* = 0 to (*a* – 1), then IFCR[2*n*] acts as a filter and IFCR[2*n* + 1] acts as the mask for IFCR[2*n*].

Address: Base + 0x4C (IFCR0)....Base + 0x88 (IFCR15) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL[1]						DIR[1]	CCS[1]	0	0	ID[1]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 These fields are writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-37. Identifier Filter Control Registers (IFCR0–IFCR15)

Table 34-35. IFCR field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame.
DIR	Direction This bit controls the direction of the data field. 0 LINFlexD receives the data and copies it in the BDRL and BDRM registers. 1 LINFlexD transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below.
ID	Identifier Identifier part of the identifier field without the identifier parity.

34.10.21 Global Control Register (GCR)

The GCR can be programmed only in initialization mode. The configuration specified in this register applies in both LIN and UART modes.

Address: Base + 0x8C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	TDFBM[1]	RDFBM[1]	TDLIS[1]	RDLIS[1]	STOP[1]	0
W																SR[1]
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 This field is writable only in initialization mode (LINCR1[INIT] = 1).

Figure 34-38. Global Control Register (GCR)

Table 34-36. GCR field descriptions

Field	Description
TDFBM	Transmit data first bit MSB This field controls the first bit of transmitted data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of transmitted data is LSB – that is, the first bit transmitted is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of transmitted data is MSB – that is, the first bit transmitted is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).

Table 34-36. GCR field descriptions (continued)

Field	Description
RDFBM	Received data first bit MSB This field controls the first bit of received data (payload only) as MSB/LSB in both UART and LIN modes. 0 The first bit of received data is LSB – that is, the first bit received is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)). 1 The first bit of received data is MSB – that is, the first bit received is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).
TDLIS	Transmit data level inversion selection This field controls the data inversion of transmitted data (payload only) in both UART and LIN modes. 0 Transmitted data is not inverted. 1 Transmitted data is inverted.
RDLIS	Received data level inversion selection This field controls the data inversion of received data (payload only) in both UART and LIN modes. 0 Received data is not inverted. 1 Received data is inverted.
STOP	Stop bit configuration This field controls the number of stop bits in transmitted data in both UART and LIN modes. The stop bit is configured for all the fields (delimiter, sync, ID, checksum, and payload). 0 One stop bit. 1 Two stop bits.
SR	Soft reset If the software writes a 1 to this field, the LINFlexD controller executes a soft reset in which the FSMs, FIFO pointers, counters, timers, status registers, and error registers are reset but the configuration registers are unaffected. This field always reads “0”.

34.10.22 UART Preset Timeout (UARTPTO) register

The UART Preset Timeout (UARTPTO) register contains the preset timeout value in UART mode, and monitors the IDLE state of the reception line. The timeout detection uses this register and the UARTCTO register described in Section 34.10.23, UART Current Timeout (UARTCTO) register.

Address: Base + 0x90 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PTO											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 34-39. UART Preset Timeout (UARTPTO) register

Table 34-37. UARTPTO field descriptions

Field	Description
PTO	Preset value of the timeout counter Do not set PTO = 0 (otherwise, UARTSR[TO] would immediately be set).

34.10.23 UART Current Timeout (UARTCTO) register

The UART Current Timeout (UARTCTO) register contains the current timeout value in UART mode, and is used in conjunction with the UARTPTO register (see Section 34.10.22, UART Preset Timeout (UARTPTO) register) to monitor the IDLE state of the reception line. UART timeout works in both CPU and DMA modes.

The timeout counter:

- Starts at zero and counts upward
- Is clocked with the baud rate clock prescaled by a hard-wired scaling factor of 16
- Is automatically enabled when UARTCR[RXEN] = 1

Address: Base + 0x94 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	CTO											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-40. UART Current Timeout (UARTCTO) register

Table 34-38. UARTCTO field descriptions

Field	Description
CTO	Current value of the timeout counter This field is reset whenever one of the following occurs: • A new value is written to the UARTPTO register • The value of this field matches the value of UARTPTO[PTO] • A hard or soft reset occurs • New incoming data is received When CTO matches the value of UARTPTO[PTO], UARTSR[TO] is set.

34.10.24 DMA Transmit Enable (DMATXE) register

The DMATXE register enables the DMA transmit interface.

Address: Base + 0x98 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTE15	DTE14	DTE13	DTE12	DTE11	DTE10	DTE9	DTE8	DTE7	DTE6	DTE5	DTE4	DTE3	DTE2	DTE1	DTE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-41. DMA Transmit Enable (DMATXE) register

Table 34-39. DMATXE field descriptions

Field	Description
DTE*n*	DMA transmit channel *n* enable 0 DMA transmit channel *n* disabled. 1 DMA transmit channel *n* enabled. **Note:** When DMATXE = 0x0, the DMA transmit interface FSM is forced (soft reset) into the IDLE state.

NOTE

For the UART mode and the LIN master mode, only the DMATXE[DTE0] bit is writable. In these modes, the DMATXE[DTE*n*] bits (where $n \geq 1$) are read-only bits set at 0, and the other 15 channels are inactive.

In the LIN slave modes, the DMA transmit channels 1 through 15 can be used (when enabled in the identifier filter registers), with 1 to *n* DMA channels where *n* = the maximum number of identifier filters enabled. For example, if three ID filters are used, then DMATXE[DTE2:0] bits are writable.

34.10.25 DMA Receive Enable (DMARXE) register

The DMARXE register enables the DMA receive interface.

Address: Base + 0x9C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DRE15	DRE14	DRE13	DRE12	DRE11	DRE10	DRE9	DRE8	DRE7	DRE6	DRE5	DRE4	DRE3	DRE2	DRE1	DRE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-42. DMA Receive Enable (DMARXE) register

Table 34-40. DMARXE field descriptions

Field	Description
DRE*n*	DMA receive channel *n* enable 0 DMA receive channel *n* disabled 1 DMA receive channel *n* enabled **Note:** When DMARXE = 0x0, the DMA receive interface FSM is forced (soft reset) into the IDLE state.

NOTE

For the UART mode and the LIN master mode, only the DMARXE[DRE0] bit is writable. In these modes, the DMARXE[DRE*n*] bits (where $n \geq 1$) are read-only bits set at 0, and the other 15 channels are inactive.

In the LIN slave modes, the DMA receive channels 1 through 15 can be used (when enabled in the identifier filter registers), with 1 to *n* DMA channels where *n* = the maximum number of identifier filters enabled. For example, if 3 ID filters are used, then DMARXE[DRE2:0] bits are writable.

34.11 DMA interface

The LINFlexD DMA interface offers a parametric and programmable solution with the following features:

- LIN master node, transmit mode: single DMA channel
- LIN master node, receive mode: single DMA channel
- LIN slave node, transmit mode: 1 to *n* DMA channels where *n* = max number of ID filters
- LIN slave node, receive mode: 1 to *n* DMA channels where *n* = max number of ID filters
- UART node, transmit mode: single DMA channel
- UART node, receive mode: single DMA channel + timeout

The LINFlexD controller interacts with an enhanced direct memory access (eDMA) controller; see the description of that controller for details on its operation and the transfer control descriptors (TCDs) referenced in this section.

34.11.1 Master node, transmit mode

On a master node in transmit mode, the DMA interface requires a single transmit channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-43.

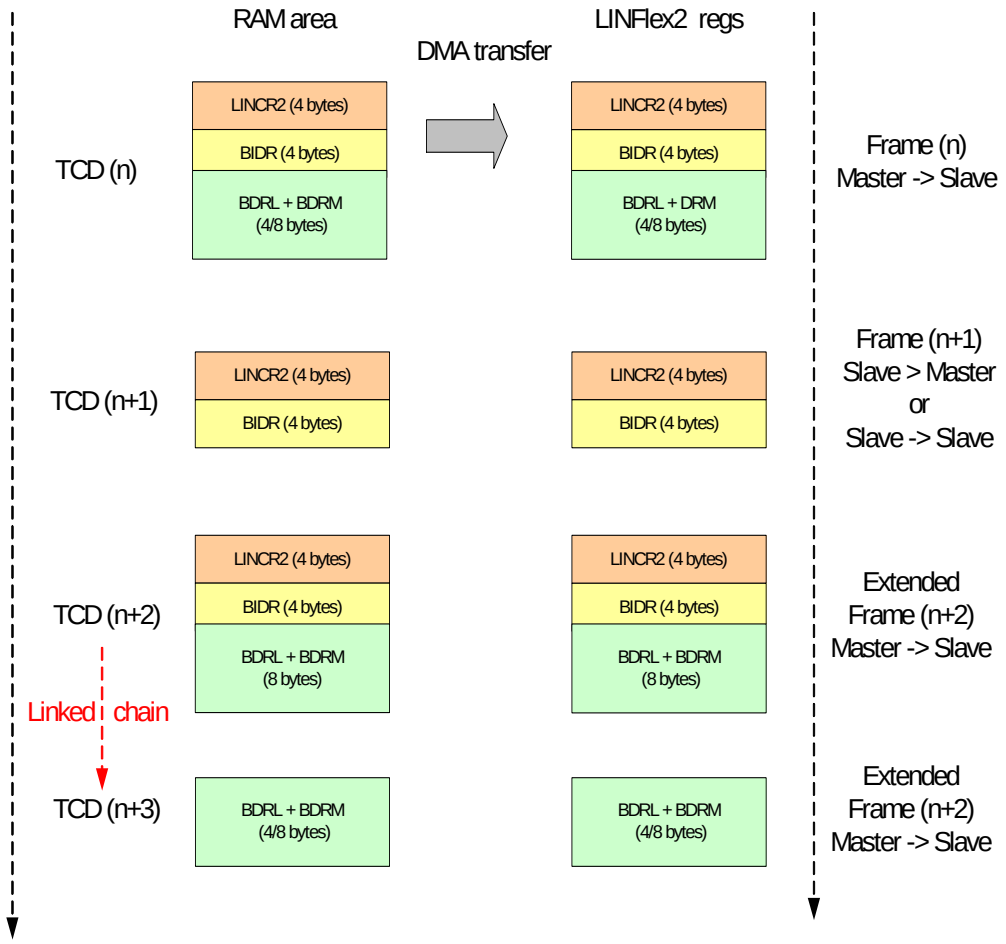


Figure 34-43. TCD chain memory map (master node, transmit mode)

The TCD chain of the DMA transmit channel on a master node supports:

- Master to slave: transmission of the entire frame (header + data)
- Slave to master: transmission of the header (the data reception is controlled by the receive channel on the master node)
- Slave to slave: transmission of the header

The register settings for the LINCR2 and BIDR registers for each class of LIN frame are shown in Table 34-41.

Table 34-41. Register settings (master node, transmit mode)

LIN frame	LINCR2	BIDR
Master to slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 1 (transmit)
Slave to master	DDRQ=0 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (receive)
Slave to slave	DDRQ=1 DTRQ=0 HTRQ=0	DFL = payload size ID = address CCS = checksum DIR = 0 (receive)

The concept FSM to control the DMA transmit interface is shown in Figure 34-44. The DMA transmit FSM will move to IDLE state immediately at next clock edge if DMATXE[0] = 0.

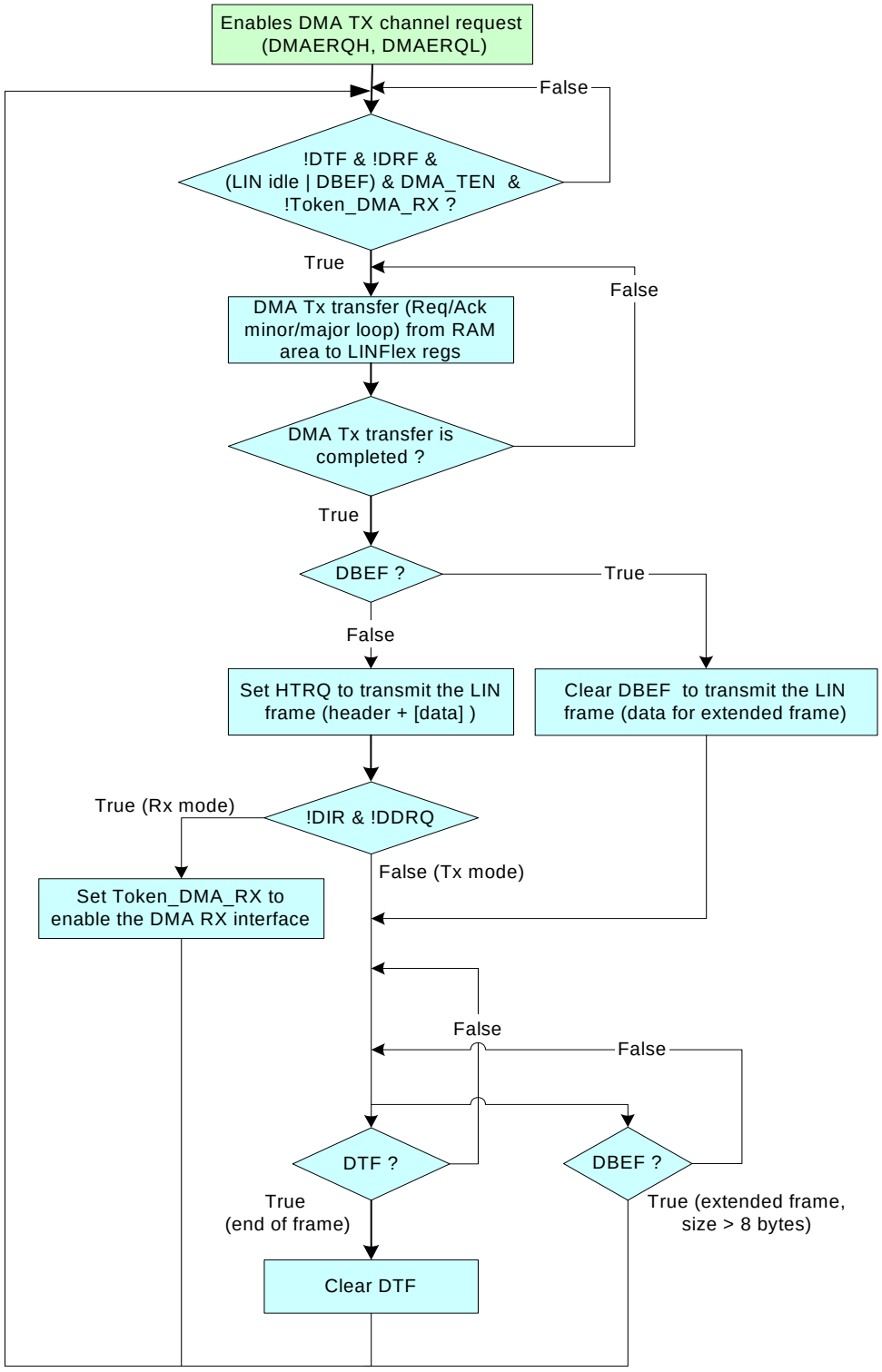


Figure 34-44. FSM to control the DMA transmit interface (master node)

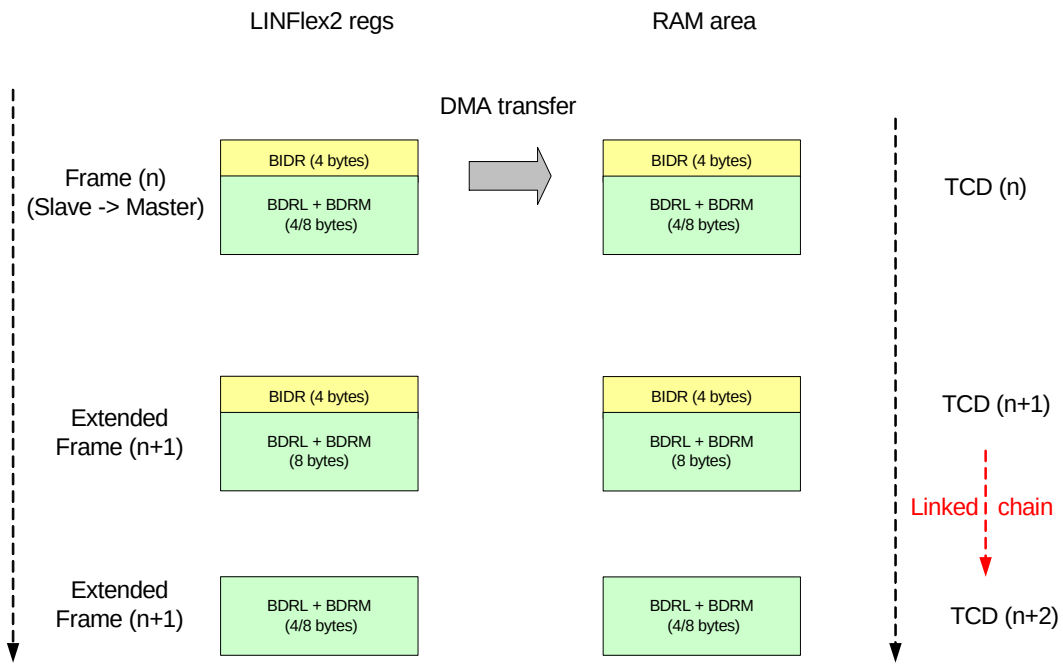
The TCD settings (word transfer) are shown in Table 34-42. All other TCD fields are equal to 0. TCD settings based on halfword or byte transfers are allowed.

Table 34-42. TCD settings (master node, transmit mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for major loop
BITER[14:0]	1	Single iteration for major loop
NBYTES[31:0]	[4 + 4] + 0/4/8 = N	Data buffer stuffed with dummy bytes if length is not word-aligned LINCR2 + BIDR + BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	LINCR2 address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on the last TCD of the chain
START	0	No software request

34.11.2 Master node, receive mode

On a master node in receive mode, the DMA interface requires a single receive channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-45.



1 DMA RX channel (TCD single and/or linked-chain)

Figure 34-45. TCD chain memory map (master node, receive mode)

The TCD chain of the DMA receive channel on a master node supports slave-to-master reception of the data field.

The BIDR register is optionally copied into the RAM area. This BIDR field (part of FIFO data) contains the ID of each message to allow the CPU to figure out which ID was received by the LINFlexD DMA if only the “one DMA channel” setup is used.

The concept FSM to control the DMA receive interface is shown in Figure 34-46. The DMA receive FSM will move to IDLE state immediately at next clock edge if DMARXE[0]=0.

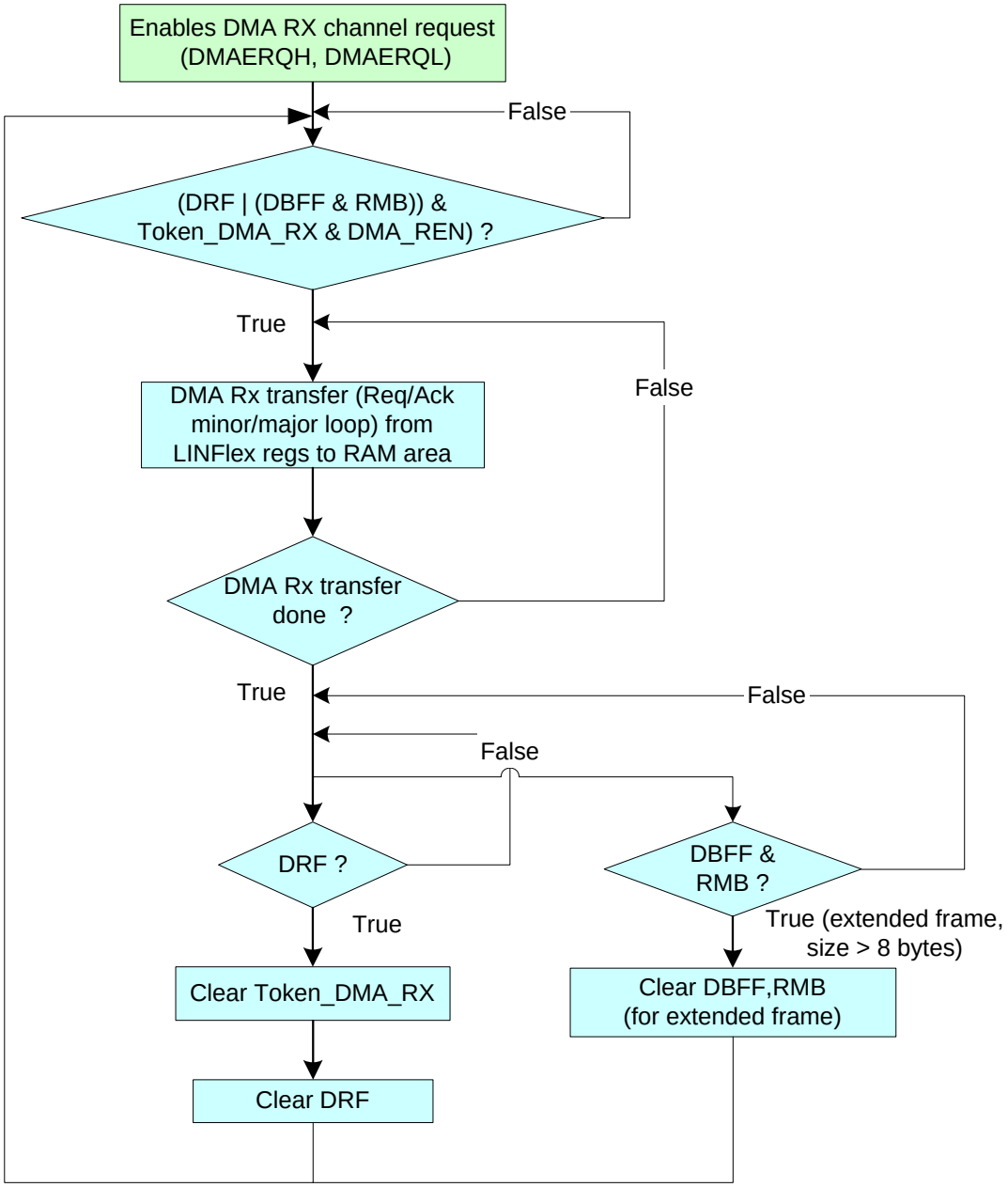


Figure 34-46. FSM to control the DMA receive interface (master node)

The TCD settings (word transfer) are shown in Table 34-43. All other TCD fields are equal to 0. TCD settings based on halfword or byte transfer are allowed.

Table 34-43. TCD settings (master node, receive mode)

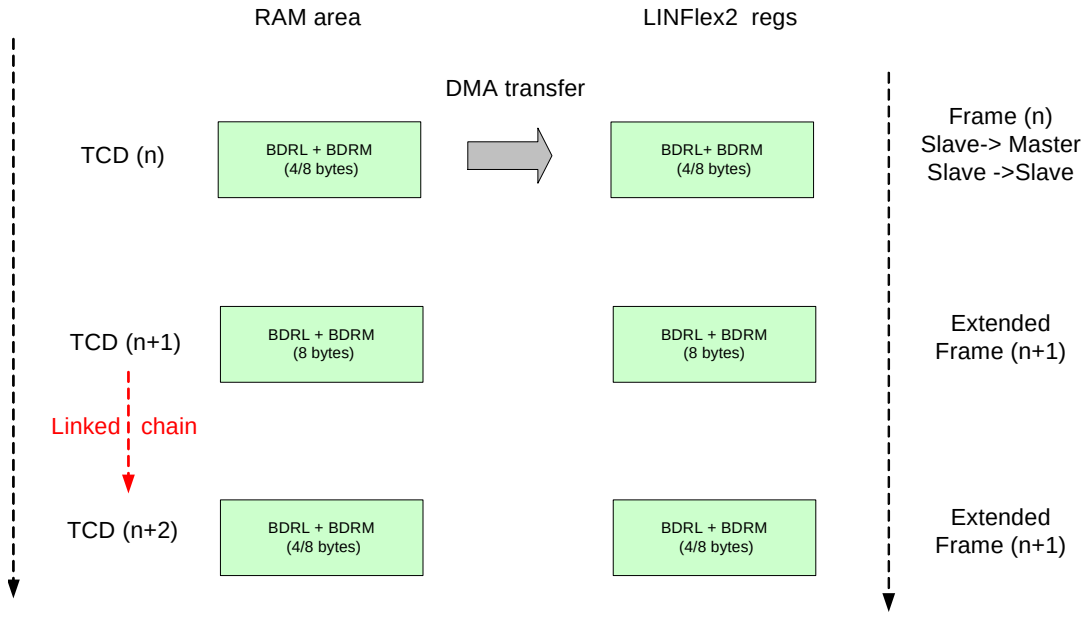
TCD field	Value	Description
CITER[14:0]	1	Single iteration for major loop
BITER[14:0]	1	Single iteration for major loop

Table 34-43. TCD settings (master node, receive mode) (continued)

TCD field	Value	Description
NBYTES[31:0]	[4] + 4/8 = N	Data buffer stuffed with dummy bytes if length is not word-aligned BIDR + BDRL + BDRM
SADDR[31:0]	BIDR address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on last TCD of chain
START	0	No software request

34.11.3 Slave node, transmit mode

On a slave node in transmit mode, the DMA interface requires a DMA transmit channel for each ID filter programmed in transmit mode. In case a single DMA transmit channel is available, a single ID field filter must be programmed in transmit mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-47.



1 DMA TX channel/ filter (TCD single and/or linked chain)

Figure 34-47. TCD chain memory map (slave node, transmit mode)

The TCD chain of the DMA transmit channel on a slave node supports:

- Slave to master: transmission of data field
- Slave to slave: transmission of data field

The register settings of the LINCR2, IFER, IFMR, and IFCR registers are shown in Table 34-44.

Table 34-44. Register settings (slave node, transmit mode)

LIN frame	LINCR2	IFER	IFMR	IFCR
Slave to master or slave to slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (transmit mode) for each DMA transmit channel	• Identifier list mode • Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 1(transmit)

The concept FSM to control the DMA transmit interface is shown in Figure 34-48. DMA transmit FSM will move to idle state if DMATXE[x] = 0, where x = IFMI – 1.

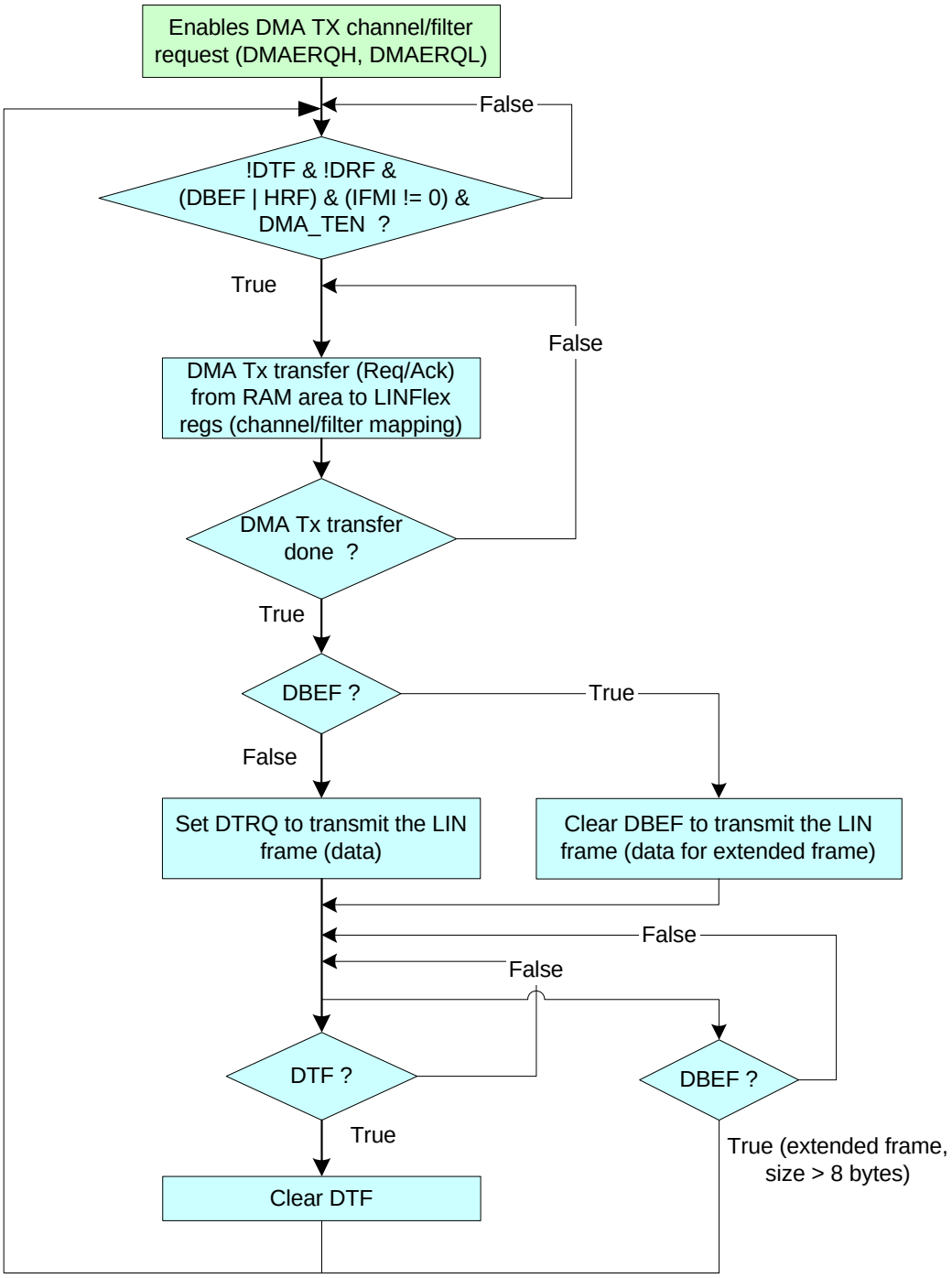


Figure 34-48. FSM to control DMA transmit interface (slave node)

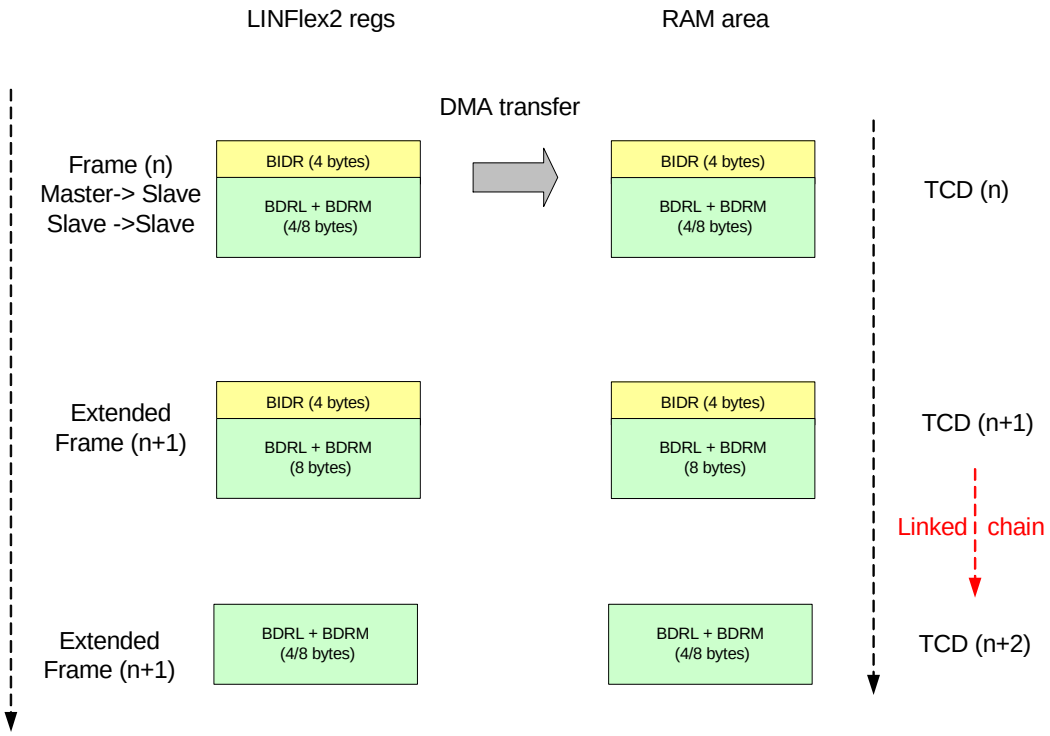
The TCD settings (word transfer) are shown in Table 34-45. All other TCD fields are equal to 0. TCD settings based on halfword or byte transfer are allowed.

Table 34-45. TCD settings (slave node, transmit mode)

TCD field	Value	Description
CITER[14:0]	1	Single iteration for major loop
BITER[14:0]	1	Single iteration for major loop
NBYTES[31:0]	4/8 = N	Data buffer stuffed with dummy bytes if length is not word-aligned BDRL + BDRM
SADDR[31:0]	RAM address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	BDRL address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on last TCD of chain
START	0	No software request

34.11.4 Slave node, receive mode

On a slave node in receive mode, the DMA interface requires a DMA receive channel for each ID filter programmed in receive mode. In case a single DMA receive channel is available, a single ID field filter must be programmed in receive mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-49.



1 DMA RX channel/ filter (TCD single and/or linked chain)

Figure 34-49. TCD chain memory map (slave node, receive mode)

The TCD chain of the DMA receive channel on a slave node supports:

- Master to slave: reception of data field
- Slave to slave: reception of data field

The register setting of the LINCR2, IFER, IFMR, and IFCR registers are given in Table 34-46.

Table 34-46. Register settings (slave node, receive mode)

LIN frame	LINCR2	IFER	IFMR	IFCR
Master to slave or slave to slave	DDRQ = 0 DTRQ = 0 HTRQ = 0	To enable an ID filter (receive mode) for each DMA receive channel	• Identifier list mode • Identifier mask mode	DFL = payload size ID = address CCS = checksum DIR = 0 (receive)

The concept FSM to control the DMA receive interface is shown in Figure 34-50. DMA receive FSM will move to idle state if DMARXE[x] = 0 where x = IFMI – 1.

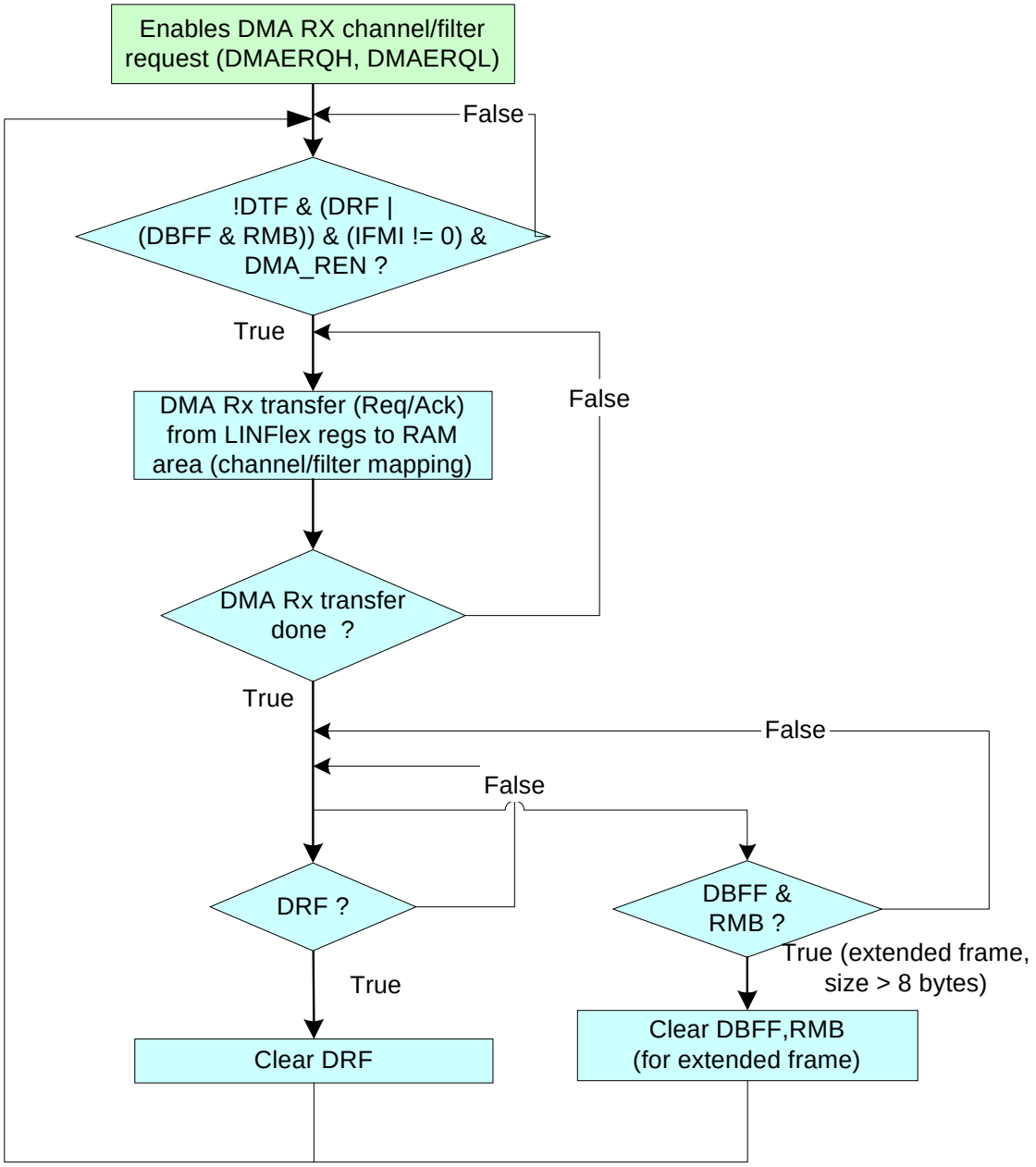


Figure 34-50. FSM to control the DMA receive interface (slave node)

The TCD settings (word transfer) are shown in Table 34-47. All other TCD fields = 0. TCD settings based on halfword or byte transfer are allowed.

Table 34-47. TCD settings (slave node, receive mode)

TCD Field	Value	Description
CITER[14:0]	1	Single iteration for major loop
BITER[14:0]	1	Single iteration for major loop

Table 34-47. TCD settings (slave node, receive mode) (continued)

TCD Field	Value	Description
NBYTES[31:0]	[4] + 4/8 = N	Data buffer stuffed with dummy bytes if length is not word-aligned BIDR + BDRL + BDRM
SADDR[31:0]	BDRL address	
SOFF[15:0]	4	Word increment
SSIZE[2:0]	2	Word transfer
SLAST[31:0]	–N	
DADDR[31:0]	RAM address	
DOFF[15:0]	4	Word increment
DSIZE[2:0]	2	Word transfer
DLAST_SGA[31:0]	–N	No scatter/gather processing
INT_MAJ	0/1	Interrupt disabled/enabled
D_REQ	1	Only on last TCD of chain
START	0	No software request

34.11.5 UART node, transmit mode

In UART transmit mode, the DMA interface requires a DMA transmit channel. A single TCD can control the transmission of an entire transmit buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-51.

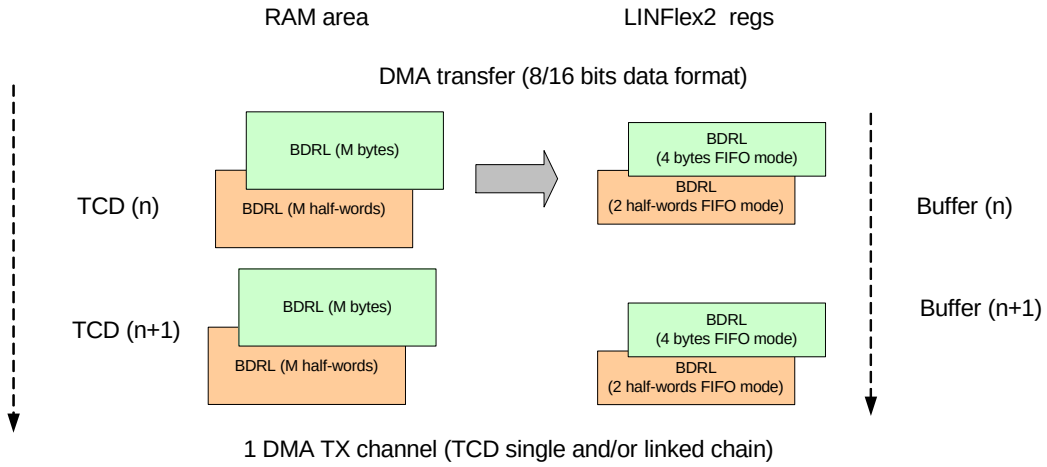


Figure 34-51. TCD chain memory map (UART node, transmit mode)

The UART transmit buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD

- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from RAM to FIFO
- Use low-priority DMA channels
- Support the UART baud rate (2 Mb/s) without underrun events

The transmit FIFO size is:

- 4 bytes in 8-bit data format
- 2 halfwords in 16-bit data format

A DMA request is triggered by FIFO-not-full (transmit) status signals.

The concept FSM to control the DMA transmit interface is shown in Figure 34-52. DMA transmit FSM will move to idle state if DMATXE[0] = 0.

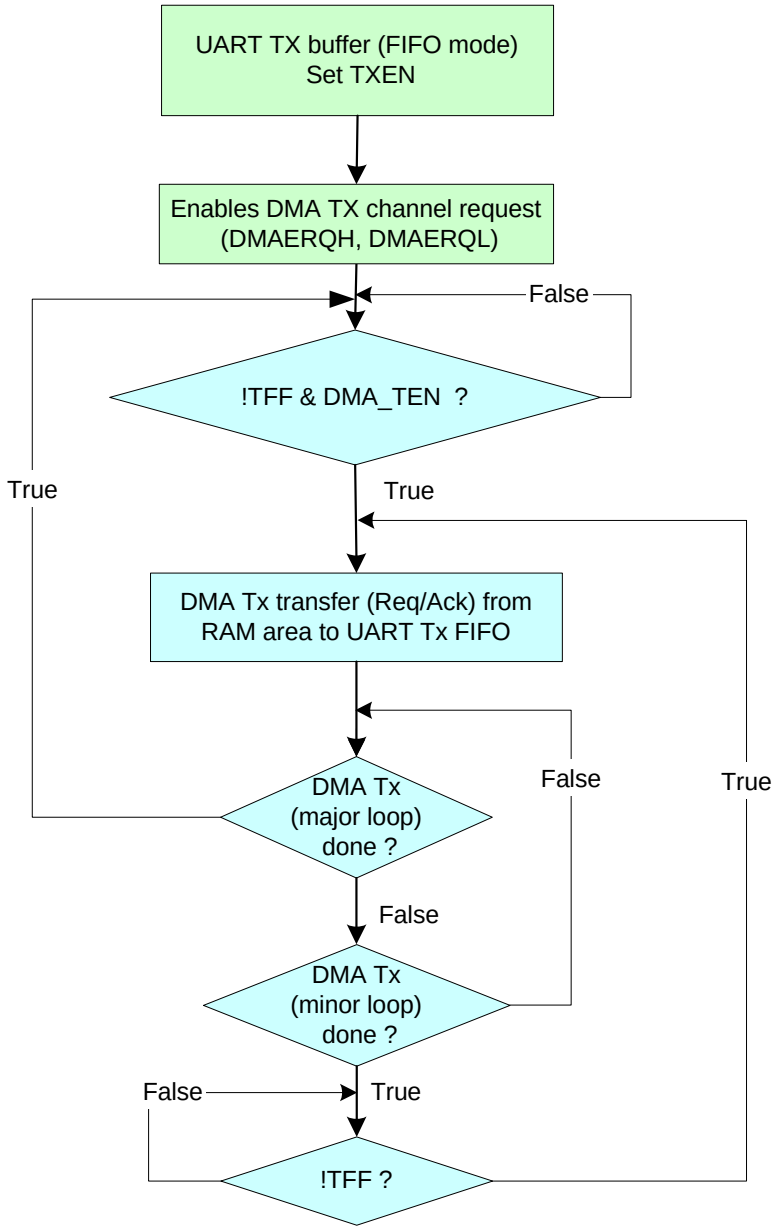


Figure 34-52. FSM to control DMA transmit interface (UART node)

The TCD settings (typical case) are shown in Table 34-48. All other TCD fields = 0. The minor loop transfers a single byte/halfword as soon a free entry is available in the transmit FIFO.

Table 34-48. TCD settings (UART node, transmit mode)

TCD Field	Value		Description
	8-bit data	16-bit data	
CITER[14:0]	M		Multiple iterations for major loop
BITER[14:0]	M		Multiple iterations for major loop

Table 34-48. TCD settings (UART node, transmit mode) (continued)

TCD Field	Value		Description
	8-bit data	16-bit data	
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	RAM address		
SOFF[15:0]	1	2	Byte/halfword increment
SSIZE[2:0]	0	1	Byte/halfword transfer
SLAST[31:0]	–M	–M × 2	
DADDR[31:0]	BDRL address		DADDR = BDRL + 0x3 for byte transfer DADDR = BDRL + 0x2 for halfword transfer
DOFF[15:0]	0		No increment (FIFO)
DSIZE[2:0]	0	1	Byte/halfword transfer
DLAST_SGA[31:0]	0		No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on last TCD of chain
START	0		No software request

34.11.6 UART node, receive mode

In UART receive mode, the DMA interface requires a DMA receive channel. A single TCD can control the reception of an entire receive buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in Figure 34-53.

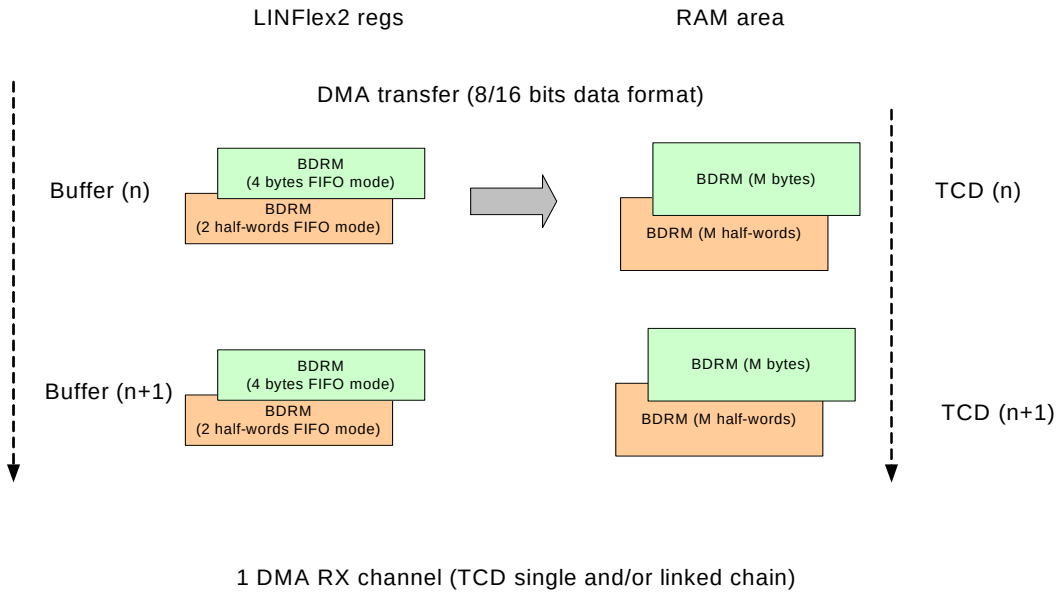


Figure 34-53. TCD chain memory map (UART node, receive mode)

The UART receive buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from FIFO to RAM
- Use low-priority DMA channels
- Support high UART baud rate (at least 2 Mb/s) without overrun events

The receive FIFO size is:

- 4 bytes in 8-bit data format
- 2 halfwords in 16-bit data format

This is sufficient because just one byte allows a reaction time of about 3.8 µs (at 2 Mbit/s), corresponding to about 450 clock cycles at 120 MHz, before the transmission is affected. A DMA request is triggered by FIFO-not-empty (receive) status signals.

The concept FSM to control the DMA receive interface is shown in Figure 34-54. DMA receive FSM will move to idle state if DMARXE[0] = 0.

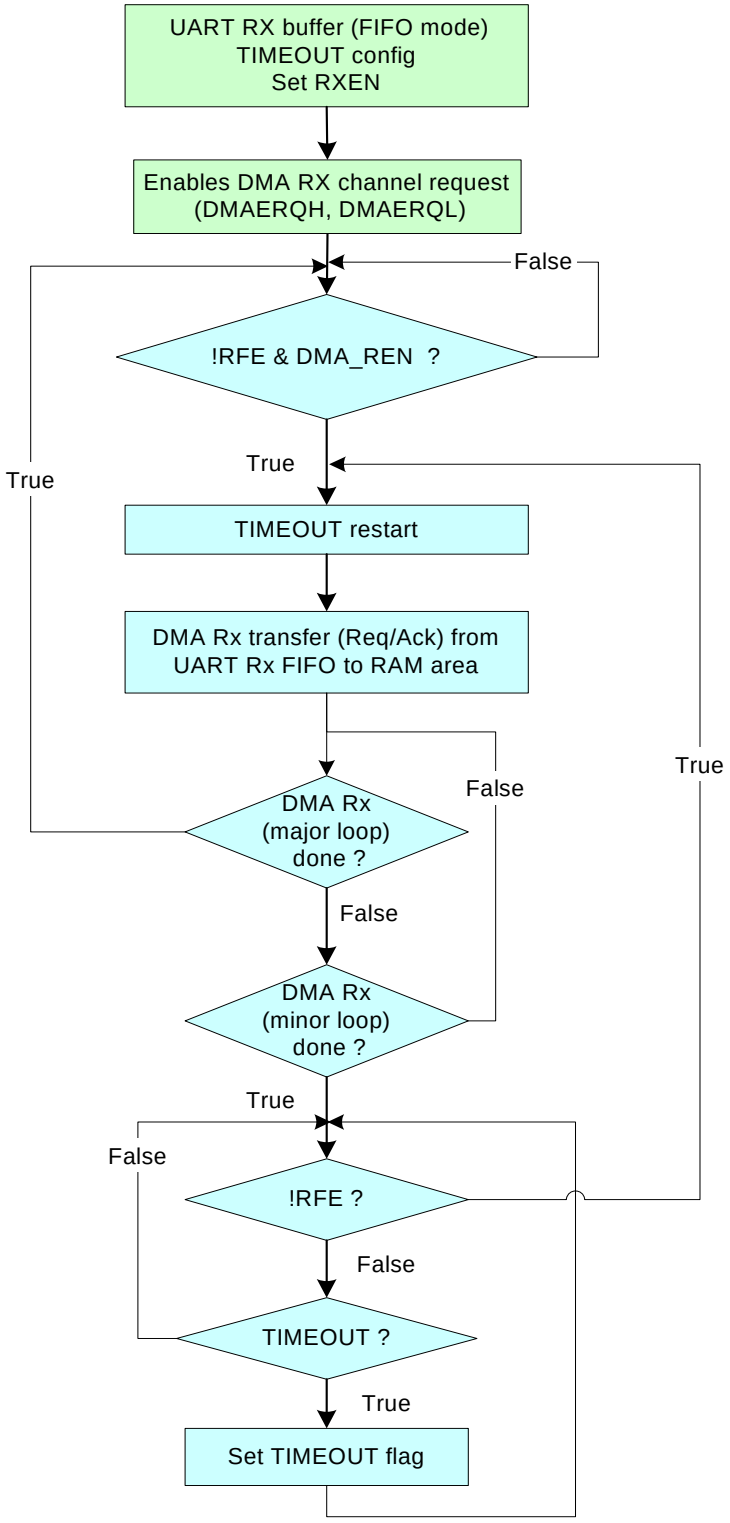


Figure 34-54. FSM to control the DMA receive interface (UART node)

The TCD settings (typical case) are shown in Table 34-49. All other TCD fields = 0. The minor loop transfers a single byte/halfword as soon an entry is available in the receive FIFO. A new software reset bit

is required that allows the LINFlexD FSMs to be reset in case this timeout state is reached or in any other case. Timeout counter can be rewritten by software at any time to extend timeout period.

Table 34-49. TCD settings (UART node, receive mode)

TCD Field	Value		Description
	8-bit data	16-bit data	
CITER[14:0]	M		Multiple iterations for major loop
BITER[14:0]	M		Multiple iterations for major loop
NBYTES[31:0]	1	2	Minor loop transfer = 1 or 2 bytes
SADDR[31:0]	BDRM address		SADDR = BDRM + 0x3 for byte transfer SADDR = BDRM + 0x2 for halfword transfer
SOFF[15:0]	0		No increment (FIFO)
SSIZE[2:0]	0	1	Byte/halfword transfer
SLAST[31:0]	0		
DADDR[31:0]	RAM address		
DOFF[15:0]	1	2	Byte/halfword increment
DSIZE[2:0]	0	1	Byte/halfword transfer
DLAST_SGA[31:0]	–M	–M × 2	No scatter/gather processing
INT_MAJ	0/1		Interrupt disabled/enabled
D_REQ	1		Only on last TCD of chain
START	0		No software request

34.11.7 Use cases and limitations

- In LIN slave mode, the DMA capability can be used only if the ID filtering mode is activated. The number of ID filters enabled must be equal to the number of DMA channels enabled. The correspondence between channel number and ID filter is based on IFMI (identifier filter match index).
- In LIN master mode both the DMA channels (transmit and receive) must be enabled in case DMA capability is required.
- In UART mode, DMA capability can be used only if the UART transmit/receive buffers are configured as FIFOs.
- DMA and CPU operating modes are mutually exclusive for the data/frame transfer on a UART or LIN node. Once a DMA transfer is finished the CPU can handle subsequent accesses.
- Error management must always be executed via CPU enabling the related error interrupt sources. The DMA capability does not provide support for error management. Error management means checking status bits, handling IRQs, and potentially canceling DMA transfers.
- The DMA programming model must be coherent with the TCD setting defined in this document.

34.12 Functional description

34.12.1 8-bit timeout counter

34.12.1.1 LIN timeout mode

The timeout counter can be used to monitor LIN frame timing as well as problems on the LIN bus, such as:

- Bus stuck at dominant state
- Header timeout
- No bus activity
- Slave not responding error (in master mode)
- Wakeup timeout

The header timeout and response timeout values can be tuned in the LINTOCR register. The LINTOCR reset value corresponds to T_{Header_max}, $T_{Response\_max}$, and T_{Frame_max} defined in the LIN Standard.

If the LINTCSR[LTOM] = 1, OC1 and OC2 output compare values in the LINOCR register are automatically updated.[1]

OC1 is used to check T_{Header} and $T_{Response}$, and OC2 is used to check T_{Frame} (see Figure 34-55).

When LINFlexD moves from Break state to Break Delimiter state (refer to LINSR register):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + HTO).
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + HTO + RTO × 9 (frame timeout value for an 8-byte frame)).
- The TOCE bit is set

On the start bit of the first data byte of the response (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO × 9 (response timeout value for an 8-byte frame)).

When BIDR[DFL] is updated (and if no error occurred during the header reception), OC1 and OC2 are automatically updated to check $T_{Response}$ and T_{Frame} according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is cleared.

This configuration is used to detect header timeout and response timeout.

1. The LINOCR register is read-only.

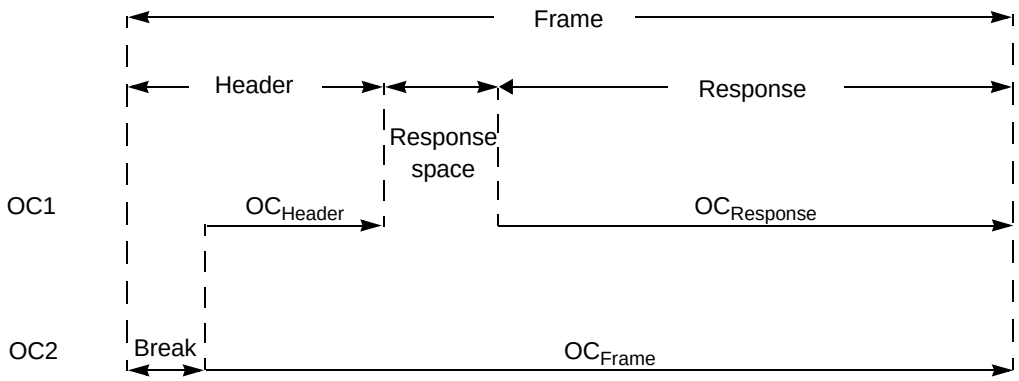


Figure 34-55. Header and response timeout

34.12.1.2 Output compare mode

The timeout counter can also be used as an Output Compare function. In this case, the LTOM bit must be reset, and output compare values can be updated in LINTOCR by software.

34.12.2 Interrupts

Table 34-50. LINFlexD interrupt control

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header received interrupt	HRF	HRIE	RXI [1]
Data transmitted interrupt	DTF	DTIE	TXI
Data received interrupt	DRF	DRIE	RXI
Data buffer empty interrupt	DBEF	DBEIE	TXI
Data buffer full interrupt	DBFF	DBFIE	RXI
Wakeup interrupt	WUPF	WUPIE	RXI
LIN state interrupt [2]	LSF	LSIE	RXI
Buffer overrun interrupt	BOF	BOIE	ERR
Framing error interrupt	FEF	FEIE	ERR
Header error interrupt	HEF	HEIE	ERR
Checksum error interrupt	CEF	CEIE	ERR
Bit error interrupt	BEF	BEIE	ERR
Output compare interrupt	OCF	OCIE	ERR
Stuck at zero interrupt	SZF	SZIE	ERR

1 In slave mode, if at least one filter is configured as transmit and enabled, then the value of the identifier received indicates whether the header received interrupt vector is RXI or TXI.

2 For debug and validation purposes.

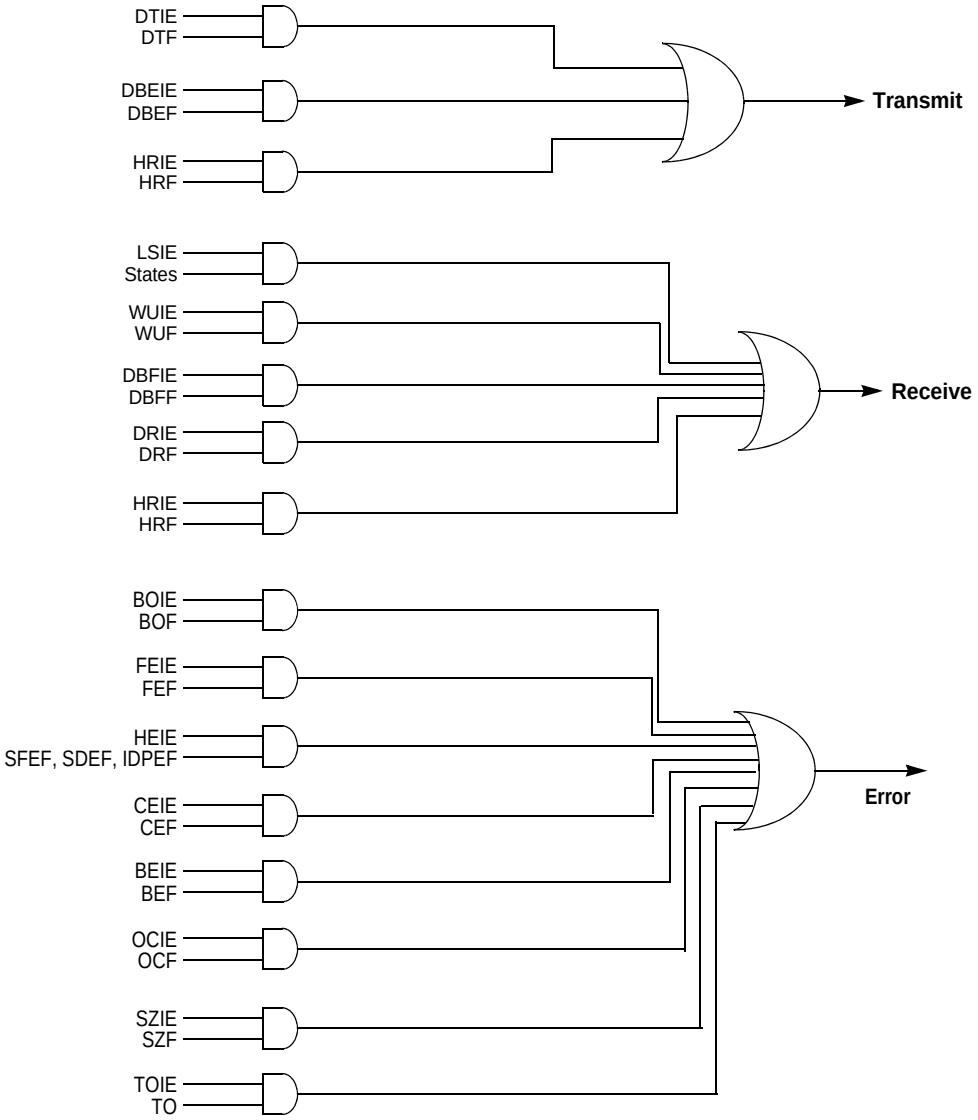


Figure 34-56. Interrupt diagram

34.12.3 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in LINIBRR (for mantissa) and LINFBRR (for fraction).

Eqn. 34-1

$$\text{Transmit/receive baud} = \frac{f_{ipg\_clock\_lin}}{(16 * \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 20-bit mantissa is coded in the LINIBRR register and the fraction is coded in LINFBRR.

The following examples show how to derive LFDIV from LINIBRR and LINFBRR values:

Example 34-1.

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12 / 16 = 0.75d

Therefore LFDIV = 27.75d

Example 34-2.

To program LFDIV = 25.62d,

LINFBRR = 16 * 0.62 = 9.92, nearest real number 10d = Ah

LINIBRR = mantissa(25.620d) = 25d = 19h

NOTE

The baud counters are updated with the new value of the baud registers after a write to LINIBRR. Hence the baud register value must not be changed during a transaction. The LINFBRR (containing the fraction bits) must be programmed before LINIBRR.

NOTE

LFDIV must be greater than or equal to 1.5d, for example, LINIBRR = 1 and LINFBRR = 8. Therefore, the maximum possible baud rate is fperiph_set_1_clk / 24.

Table 34-51. Error calculation for programmed baud rates

Baud rate	$f_{ipg\_clock\_lin}$ = 64 MHz				$f_{ipg\_clock\_lin}$ = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) Baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) Baud rate / Desired baud rate
		LINIBRR	LINFBRR			LINIBRR	LINFBRR	
2400	2399.97	1666	11	–0.001	2399.88	416	11	–0.005
9600	9599.52	416	11	–0.005	9598.08	104	3	–0.02
10417	10416.7	383	16	–0.003	10416.7	95	16	–0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	–0.08
115200	115108	34	12	–0.08	115108	8	11	–0.08

Table 34-51. Error calculation for programmed baud rates (continued)

Baud rate	$f_{ipg\_clock\_lin}$ = 64 MHz				$f_{ipg\_clock\_lin}$ = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) Baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) Baud rate / Desired baud rate
		LINIBRR	LINFBRR			LINIBRR	LINFBRR	
230400	230216	17	6	-0.08	231884	4	5	0.644
460800	460432	8	11	–0.08	457143	2	3	–0.794
921600	927536	4	5	0.644	941176	1	1	2.124

34.13 Programming considerations

This section describes the various configurations in which the LINFlexD can be used.

34.13.1 Master node

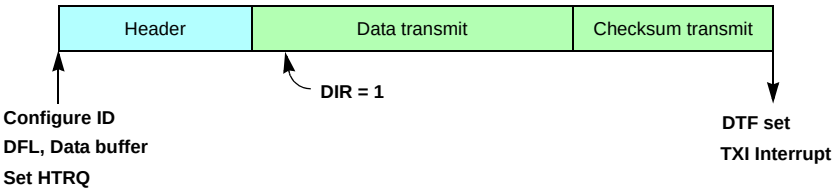


Figure 34-57. Programming consideration: master node, transmitter

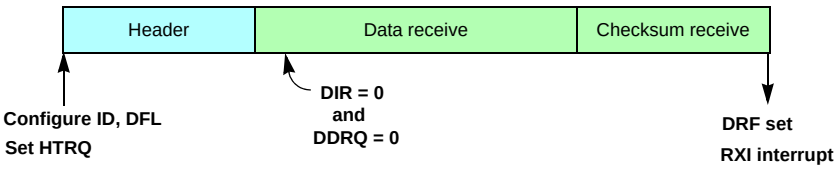


Figure 34-58. Programming consideration: master node, receiver

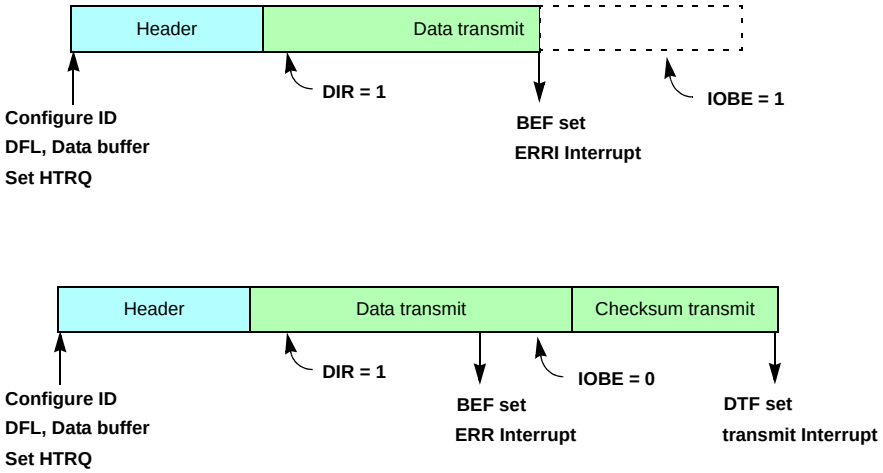


Figure 34-59. Programming consideration: master node, transmitter, bit error

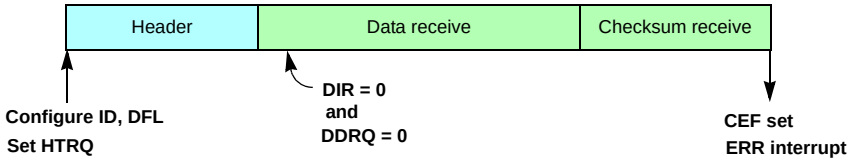


Figure 34-60. Programming consideration: master node, receiver, checksum error

34.13.2 Slave node

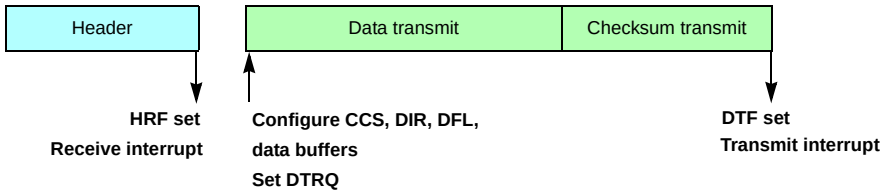


Figure 34-61. Programming consideration: slave node, transmitter, no filters

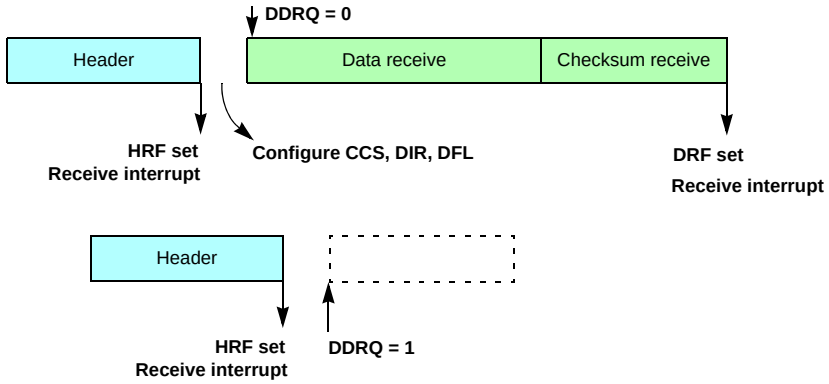


Figure 34-62. Programming consideration: slave node, receiver, no filters

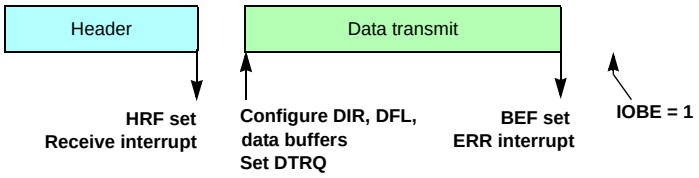


Figure 34-63. Programming consideration: slave node, transmitter, no filters, bit error

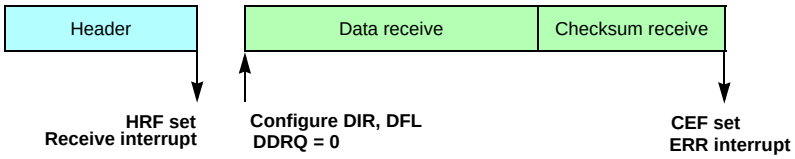
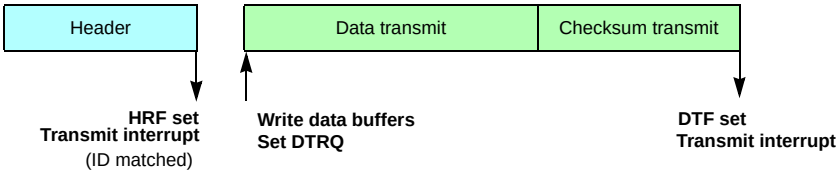


Figure 34-64. Programming consideration: slave node, receiver, no filters, checksum error



Note: This configuration can be used in case the slave never receives data (for example, with a sensor).

Figure 34-65. Programming consideration: slave node, at least one transmit filter, BF is reset, ID matches filter

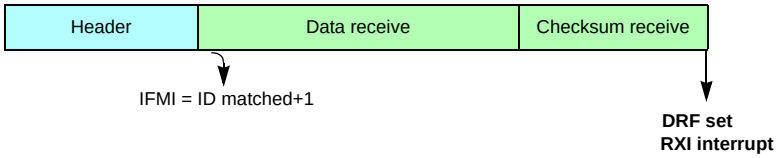


Figure 34-66. Programming consideration: slave node, at least one receive filter, BF is reset, ID matches filter

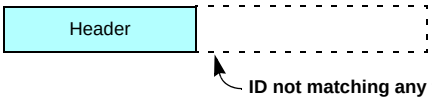
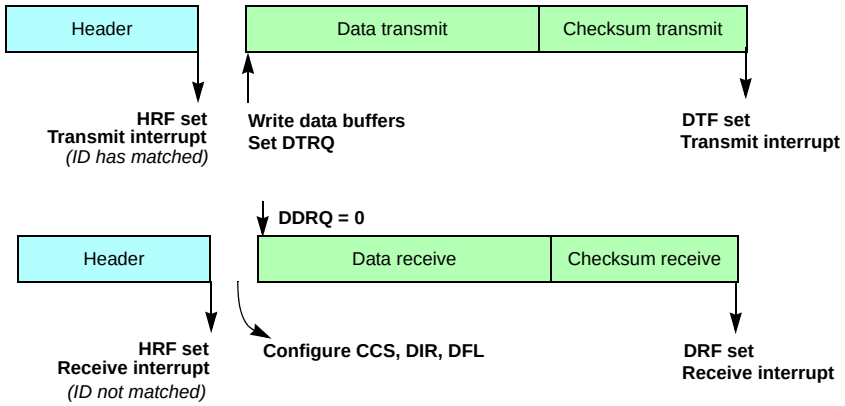


Figure 34-67. Programming consideration: slave node, receive only, transmit only, receive and transmit filters, ID not matching filter, BF is reset



Note: This configuration is used when:
a) All transmit IDs are managed by filters
b) The number of other filters is not enough to manage all reception IDs

Figure 34-68. Programming consideration: slave node, transmit filter, BF is set

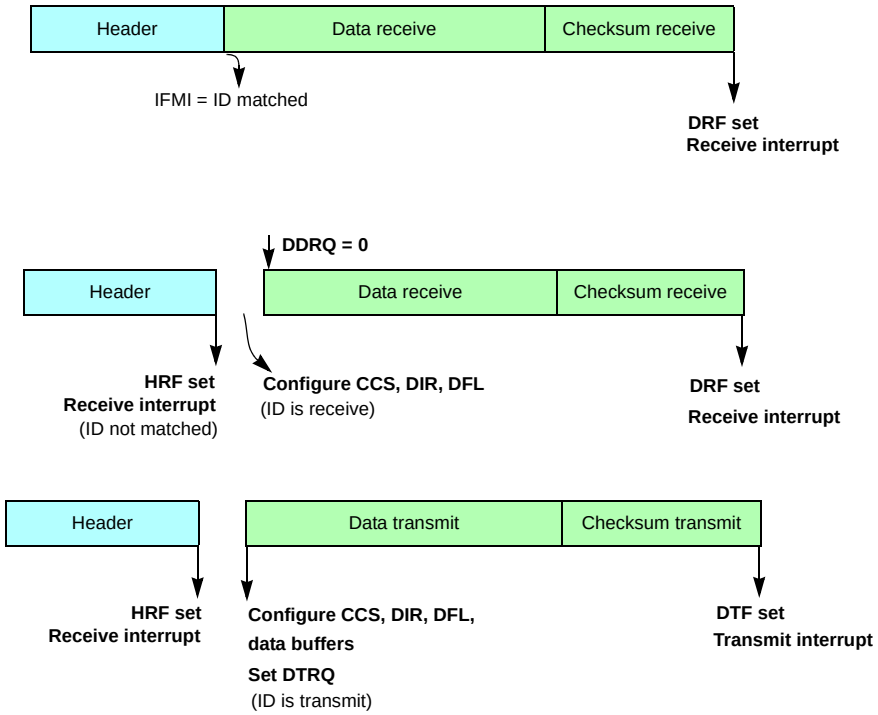
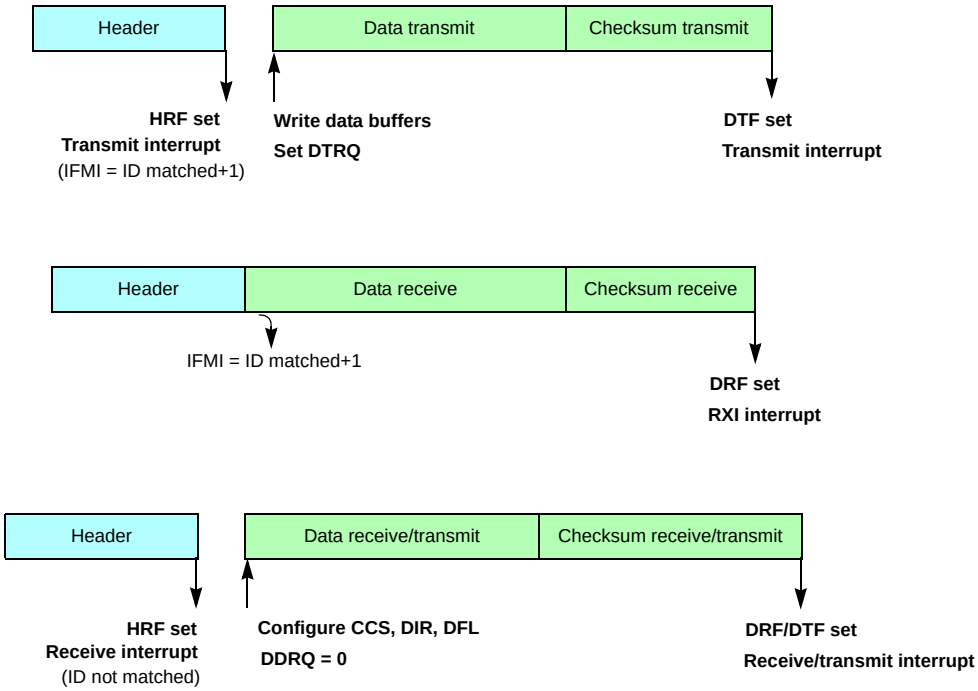


Figure 34-69. Programming consideration: slave node, receive filter, BF is set



Note: This configuration is used when:
a) The number of filters is not enough
b) Filters are used for most frequently used IDs to reduce CPU usage

Figure 34-70. Programming consideration: slave node, transmit filter, receive filter, BF is set

The following three steps should be followed:

1) Set the MODE bit in the LIN Time-Out Control Status Register (LINTCSR[MODE]) to '0'.

2) Set Idle on Timeout in the LINTCSR[IOT] register to '1'.

3) Configure master to wait until the occurrence of the Output Compare flag in LIN Error Status Register (LINESR[OCF]) before sending the next header. This flag causes the LIN Slave to go to an IDLE state before the next header arrives, which will be accepted without any framing error.

34.13.3 Extended frames

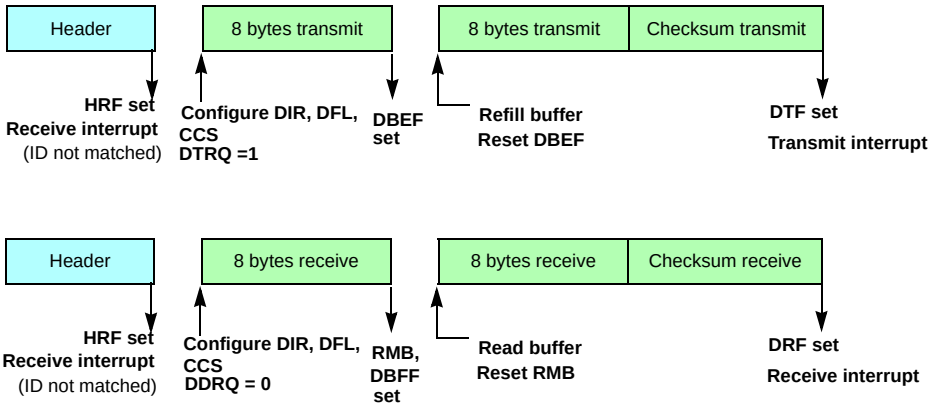


Figure 34-71. Programming consideration: extended frames

34.13.4 Timeout

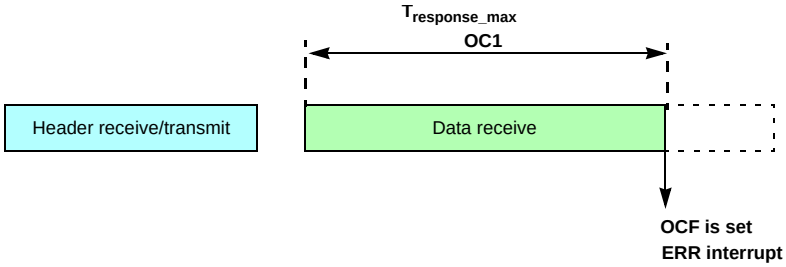


Figure 34-72. Programming consideration: response timeout

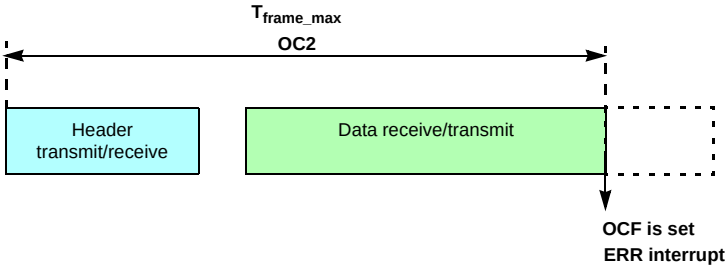


Figure 34-73. Programming consideration: frame timeout

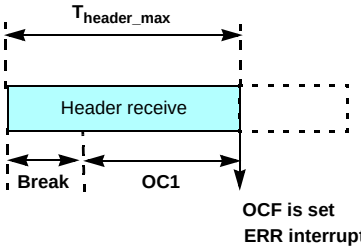


Figure 34-74. Programming consideration: header timeout

34.13.5 UART mode

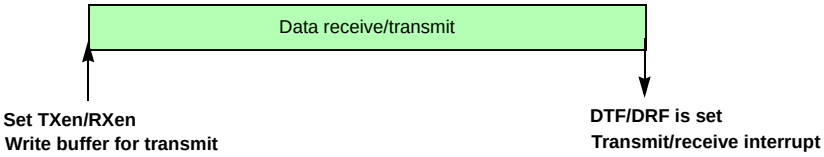


Figure 34-75. Programming consideration: UART mode

This page is intentionally left blank.

Chapter 35
Mode Entry Module (MC_ME)

35.1 Introduction

35.1.1 Overview

The MC_ME controls the MPC5675K mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

Figure 35-1 shows the MC_ME block diagram.

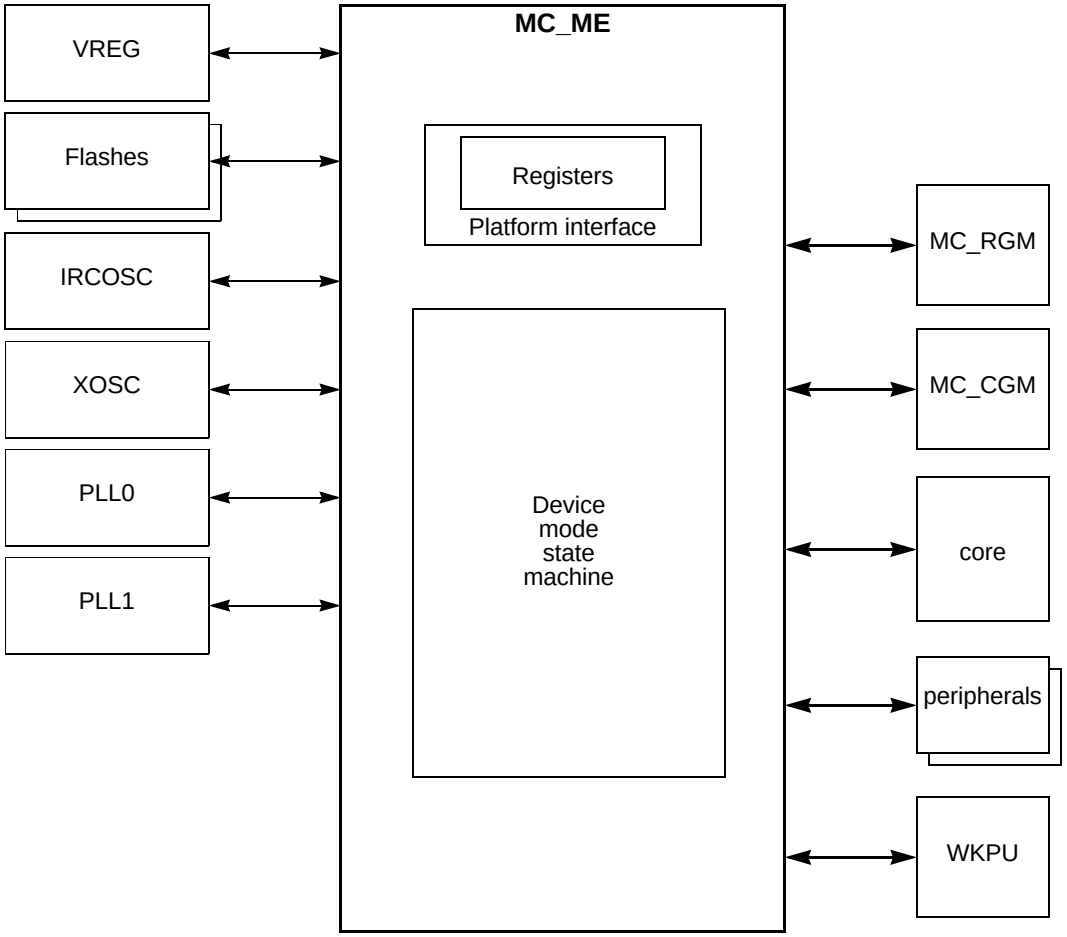


Figure 35-1. MC_ME block diagram

35.1.2 Features

The MC_ME includes the following features:

- Control of the available modes by the ME_ME register
- Definition of various device mode configurations by the ME_<*mode*>_MC registers

- Control of the actual device mode by the ME_MCTL register
- Capture of the current mode and various resource status information within the contents of the ME_GS register
- Optional generation of various mode transition interrupts
- Status bits for each cause of invalid mode transitions
- Peripheral clock gating control based on the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL*n* registers
- Capture of current peripheral clock gated/enabled status

35.1.3 Modes of operation

The MC_ME is based on several device modes corresponding to different usage models of the device. Each mode is configurable and can define a policy for energy and processing power management to fit particular system requirements. An application can easily switch from one mode to another depending on the current needs of the system. The operating modes controlled by the MC_ME are divided into system and user modes. The system modes are modes such as RESET, DRUN, SAFE, and TEST. These modes aim to ease the configuration and monitoring of the system. The user modes are modes such as RUN0…3, HALT0, and STOP0, which can be configured to meet the application requirements in terms of energy management and available processing power. The modes DRUN, SAFE, TEST, and RUN0…3 are the device software running modes.

Table 35-1 describes the MC_ME modes.

Table 35-1. MC_ME mode descriptions

Name	Description	Entry	Exit
RESET	This is a chip-wide virtual mode during which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the device. It manages hardware initialization of chip configuration, voltage regulators, clock sources, and flash modules.	System reset assertion from MC_RGM	System reset deassertion from MC_RGM
DRUN	This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter user modes. BAM when present is executed in DRUN mode.	System reset deassertion from MC_RGM, software request from SAFE, TEST and RUN0…3	System reset assertion, RUN0…3, TEST via software, SAFE via software or hardware failure.
SAFE	This is a chip-wide service mode that may be entered on the detection of a recoverable error. It forces the system into a pre-defined safe configuration from which the system may try to recover.	Hardware failure from any mode, software request from DRUN, TEST, and RUN0…3	System reset assertion, DRUN via software
TEST	This is a chip-wide service mode that is intended to provide a control environment for device software testing.	Software request from DRUN	System reset assertion, SAFE via software or hardware failure, DRUN via software

Table 35-1. MC_ME mode descriptions (continued)

Name	Description	Entry	Exit
RUN0...3	These are software running modes where most processing activity is done. These various run modes allow enabling different clock and power configurations of the system with respect to each other.	Software request from DRUN or other RUN0…3, interrupt event from HALT0, interrupt or wakeup event from STOP0	System reset assertion, SAFE via software or hardware failure, DRUN, other RUN0…3 modes, HALT0, STOP0 via software
HALT0	This is a reduced-activity low-power mode during which the clock to the core is disabled. It can be configured to switch off analog peripherals like clock sources, flash, main regulator, etc., for efficient power management at the cost of higher wakeup latency.	Software request from RUN0…3	System reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event
STOP0	This is an advanced low-power mode during which the clock to the core is disabled. It may be configured to switch off most of the peripherals, including clock sources for efficient power management, at the cost of higher wakeup latency.	Software request from RUN0…3	System reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event or wakeup event

35.2 External signal description

The MC_ME has no connections to any external pins.

35.3 Memory map and register description

The MC_ME contains registers for:

- Mode selection and status reporting
- Mode configuration
- Mode transition interrupt status and mask control
- Scalable number of peripheral sub-mode selection and status reporting

35.3.1 Memory map

Table 35-2. MC_ME register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C000	ME_GS	Global Status	word	read	read	read	on page 1307
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	on page 1309

Table 35-2. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	on page 1310
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	on page 1312
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	on page 1313
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	on page 1314
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	on page 1315
0xC3FD_C020	ME_RESET_MC	RESET Mode Configuration	word	read	read	read	on page 1318
0xC3FD_C024	ME_TEST_MC	TEST Mode Configuration	word	read	read/write	read/write	on page 1319
0xC3FD_C028	ME_SAFE_MC	SAFE Mode Configuration	word	read	read/write	read/write	on page 1319
0xC3FD_C02C	ME_DRUN_MC	DRUN Mode Configuration	word	read	read/write	read/write	on page 1320
0xC3FD_C030	ME_RUN0_MC	RUN0 Mode Configuration	word	read	read/write	read/write	on page 1320
0xC3FD_C034	ME_RUN1_MC	RUN1 Mode Configuration	word	read	read/write	read/write	on page 1320
0xC3FD_C038	ME_RUN2_MC	RUN2 Mode Configuration	word	read	read/write	read/write	on page 1320
0xC3FD_C03C	ME_RUN3_MC	RUN3 Mode Configuration	word	read	read/write	read/write	on page 1320
0xC3FD_C040	ME_HALT0_MC	HALT0 Mode Configuration	word	read	read/write	read/write	on page 1321
0xC3FD_C048	ME_STOP0_MC	STOP0 Mode Configuration	word	read	read/write	read/write	on page 1321
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	on page 1323
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	on page 1324
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	on page 1324
0xC3FD_C06C	ME_PS3	Peripheral Status 3	word	read	read	read	on page 1325

Table 35-2. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	on page 1325
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	on page 1325
0xC3FD_C088	ME_RUN_PC2	Run Peripheral Configuration 2	word	read	read/write	read/write	on page 1325
0xC3FD_C08C	ME_RUN_PC3	Run Peripheral Configuration 3	word	read	read/write	read/write	on page 1325
0xC3FD_C090	ME_RUN_PC4	Run Peripheral Configuration 4	word	read	read/write	read/write	on page 1325
0xC3FD_C094	ME_RUN_PC5	Run Peripheral Configuration 5	word	read	read/write	read/write	on page 1325
0xC3FD_C098	ME_RUN_PC6	Run Peripheral Configuration 6	word	read	read/write	read/write	on page 1325
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	on page 1325
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	on page 1326
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	on page 1326
0xC3FD_C0A8	ME_LP_PC2	Low-Power Peripheral Configuration 2	word	read	read/write	read/write	on page 1326
0xC3FD_C0AC	ME_LP_PC3	Low-Power Peripheral Configuration 3	word	read	read/write	read/write	on page 1326
0xC3FD_C0B0	ME_LP_PC4	Low-Power Peripheral Configuration 4	word	read	read/write	read/write	on page 1326
0xC3FD_C0B4	ME_LP_PC5	Low-Power Peripheral Configuration 5	word	read	read/write	read/write	on page 1326
0xC3FD_C0B8	ME_LP_PC6	Low-Power Peripheral Configuration 6	word	read	read/write	read/write	on page 1326
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	on page 1326
0xC3FD_C0C4	ME_PCTL4	DSPI0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0C5	ME_PCTL5	DSPI1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0C6	ME_PCTL6	DSPI2 Control	byte	read	read/write	read/write	on page 1327

Table 35-2. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0D0	ME_PCTL16	FlexCAN0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0D1	ME_PCTL17	FlexCAN1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0D2	ME_PCTL18	FlexCAN2 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0D3	ME_PCTL19	FlexCAN3 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0D8	ME_PCTL24	FlexRay Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E0	ME_PCTL32	ADC0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E1	ME_PCTL33	ADC1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E3	ME_PCTL35	CTU0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E6	ME_PCTL38	eTimer0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E7	ME_PCTL39	eTimer1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E8	ME_PCTL40	eTimer2 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0E9	ME_PCTL41	FlexPWM0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0EA	ME_PCTL42	FlexPWM1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0EC	ME_PCTL44	I2C_DMA0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0ED	ME_PCTL45	I2C_DMA1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0EE	ME_PCTL46	I2C_DMA2 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0F0	ME_PCTL48	LIN_FLEX0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0F1	ME_PCTL49	LIN_FLEX1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0F2	ME_PCTL50	LIN_FLEX2 Control	byte	read	read/write	read/write	on page 1327

Table 35-2. MC_ME register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C0F3	ME_PCTL51	LIN_FLEX3 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C0FA	ME_PCTL58	CRC0 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C101	ME_PCTL65	EBI Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C106	ME_PCTL70	DRAMC Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C11C	ME_PCTL92	PIT_RTI Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C130	ME_PCTL112	PDI Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C134	ME_PCTL116	ADC2 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C135	ME_PCTL117	ADC3 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C137	ME_PCTL119	CTU1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C138	ME_PCTL120	CRC1 Control	byte	read	read/write	read/write	on page 1327
0xC3FD_C13D	ME_PCTL125	FlexPWM2 Control	byte	read	read/write	read/write	on page 1327

NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

Table 35-3. MC_ME memory map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	1	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R	0	0	0	0	0	0	0	0	S_PLL1	S_PLL0	S_XOSC	S_IRCOSC	S_SYSCLK			
		W																
0xC3FD_C004	ME_MCTL	R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
		W	KEY															
0xC3FD_C008	ME_ME	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	RESET_DEST	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC
		W																
0xC3FD_C00C	ME_IS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	I_ICONF_CU	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C010	ME_IM	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	M_ICONF_CU	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																

Table 35-3. MC_ME memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C018	ME_DMTS	R	PREVIOUS_MODE				0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
		W																
		R	0	VREG_CSRC_SC	CSRC_CSRC_SC	IRCOSC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
		W																
0xC3FD_C01C	Reserved																	
0xC3FD_C020	ME_RESET_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C024	ME_TEST_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																

Table 35-3. MC_ME memory map (continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	27 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C030 … 0xC3FD_C03C	ME_RUN0…3_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C044	Reserved																	

Table 35-3. MC_ME memory map (continued)

Address	Name		0 16	1 17	2 18	3 19	27 20	5 21	6 22	7 23	8 24	9 25	10 26	11 27	12 28	13 29	14 30	15 31
0xC3FD_C048	ME_STOP0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
		W																
0xC3FD_C04C ... 0xC3FD_C05C	Reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	0	0	S_FlexRay	0	0	0	0	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
		W																
		R	0	0	0	0	0	0	0	0	0	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0	0	0	S_CRC0	0	0	0	0	0	0	S_LIN_FLEX3	S_LIN_FLEX2	S_LIN_FLEX1	S_LIN_FLEX0
		W																
		R	0	S_I2C_DMA2	S_I2C_DMA1	S_I2C_DMA0	0	S_FlexPWM1	S_FlexPWM0	S_eTimer2	S_eTimer1	S_eTimer0	0	0	S_CTU0	0	S_ADC1	S_ADC0
		W																

Table 35-3. MC_ME memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT_RTI	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	S_DRAMC	0	0	0	0	S_EBI	0
		W																
0xC3FD_C06C	ME_PS3	R	0	0	S_FlexPWM2	0	0	0	0	S_CRC1	S_CTU1	0	S_ADC3	S_ADC2	0	0	0	S_PDI
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FD_C070 … 0xC3FD_C07C	Reserved																	
0xC3FD_C080 … 0xC3FD_C09C	ME_RUN_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		W																
0xC3FD_C0A0 … 0xC3FD_C0BC	ME_LP_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
		W																

Table 35-3. MC_ME memory map (continued)

Address	Name		0 16	1 17	2 18	3 19	27 20	5 21	6 22	7 23	8 24	9 25	10 26	11 27	12 28	13 29	14 30	15 31
0xC3FD_C0C0 … 0xC3FD_C14C	ME_PCTL0…143[1]	R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
		R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
0xC3FD_C150 … 0xC3FD_FFFC	Reserved																	

[1] There is space in the register map for 144 peripherals. Please refer to Table 35-2 (MC_ME register description) for the ME_PCTL*n* locations actually occupied. The unoccupied locations contain a read-only byte value of 0x00.

35.3.2 Register Description

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered in big-endian format. For example, the ME_RUN_PC0 register may be accessed as a word at address 0xC3FD_C080, as a half-word at address 0xC3FD_C082, or as a byte at address 0xC3FD_C083. Some fields may be read-only, and their reset value of ‘1’ or ‘0’ and the corresponding behavior cannot be changed.

35.3.2.1 Global Status Register (ME_GS)

Address 0xC3FD_C000 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CURRENT_MODE				S_MTRANS	1	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	S_PLL1	S_PLL0	S_XOSC	S_IRCOSC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-2. Global Status Register (ME_GS)

This register contains global mode status.

Table 35-4. Global Status Register (ME_GS) field descriptions

Field	Description
S_CURRENT_MODE	Current device mode status 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT0 1001 Reserved 1010 STOP0 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Reserved
S_MTRANS	Mode transition status 0 Mode transition process is not active 1 Mode transition is ongoing
S_PDO	Output power-down status — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and the pads power sequence driver is disabled. The inputs are level unchanged. In STOP0 mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
S_MVR	Main voltage regulator status 0 Main voltage regulator is not ready. 1 Main voltage regulator is ready for use.
S_DFLA	Data flash availability status 00 Data flash is not available. 01 Data flash is in power-down mode. 10 Data flash is not available. 11 Data flash is in normal mode and available for use.
S_CFLA	Code flash availability status 00 Code flash is not available. 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode and available for use.
S_SSCLK1	Secondary system clock source 1 status 0 Secondary system clock source 1 is not stable. 1 Secondary system clock source 1 is providing a stable clock.
S_PLL1	Secondary PLL status 0 Secondary PLL is not stable. 1 Secondary PLL is providing a stable clock.
S_PLL0	System PLL status 0 System PLL is not stable. 1 System PLL is providing a stable clock.

Table 35-4. Global Status Register (ME_GS) field descriptions (continued)

Field	Description
S_XOSC	4-40 MHz crystal oscillator status 0 4-40 MHz crystal oscillator is not stable. 1 4-40 MHz crystal oscillator is providing a stable clock.
S_IRCOSC	16 MHz internal RC oscillator status 0 16 MHz internal RC oscillator is not stable. 1 16 MHz internal RC oscillator is providing a stable clock.
S_SYSCLK	System clock switch status — These bits specify the system clock currently used by the system. 0000 16 MHz internal RC oscillator 0001 Reserved. 0010 Reserved. 0011 Reserved. 0100 System PLL. 0101 Reserved. 0110 Reserved. 0111 Reserved. 1000 Reserved. 1001 Reserved. 1010 Reserved. 1011 Reserved. 1100 Reserved. 1101 Reserved. 1110 Reserved. 1111 System clock is disabled.

35.3.2.2 Mode Control Register (ME_MCTL)

Address 0xC3FD_C004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
W	KEY															
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

Figure 35-3. Mode Control Register (ME_MCTL)

This register is used to trigger software-controlled mode changes. Depending on the modes as enabled by ME_ME register bits, configurations corresponding to unavailable modes are reserved, and access to ME_<*mode*>_MC registers must respect this for successful mode requests.

NOTE

Byte and half-word write accesses are not allowed for this register because a predefined key is required to change its value.

Table 35-5. Mode Control Register (ME_MCTL) field descriptions

Field	Description
TARGET _MODE	Target device mode — These bits provide the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: the first time with a key, and the second time with an inverted key. These bits are automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT0 and STOP0 modes on hardware exit events, these are updated with the appropriate RUN0…3 mode value. 0000 RESET. 0001 TEST. 0010 SAFE. 0011 DRUN. 0100 RUN0. 0101 RUN1. 0110 RUN2. 0111 RUN3. 1000 HALT0. 1001 Reserved. 1010 STOP0. 1011 Reserved. 1100 Reserved. 1101 Reserved. 1110 Reserved. 1111 Software Destructive RESET.
KEY	Control key — These bits enable write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return inverted key. KEY:0101101011110000 (0x5AF0) INVERTED KEY:1010010100001111 (0xA50F)

35.3.2.3 Mode Enable Register (ME_ME)

Address 0xC3FD_C008 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RESET_DEST	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET_FUNC
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 35-4. Mode Enable Register (ME_ME)

This register allows a way to disable the device modes that are not required for a given device. RESET_FUNC, SAFE, DRUN, RUN0, and RESET_DEST modes are always enabled.

Table 35-6. Mode Enable Register (ME_ME) field descriptions

Field	Description
RESET_DEST	RESET_DEST mode enable 0 RESET_DEST mode is disabled. 1 RESET_DEST mode is enabled. **Note:** This read-only bit is always set to 1.
STOP0	STOP0 mode enable 0 STOP0 mode is disabled. 1 STOP0 mode is enabled.
HALT0	HALT0 mode enable 0 HALT0 mode is disabled. 1 HALT0 mode is enabled.
RUN3	RUN3 mode enable 0 RUN3 mode is disabled. 1 RUN3 mode is enabled.
RUN2	RUN2 mode enable 0 RUN2 mode is disabled. 1 RUN2 mode is enabled.
RUN1	RUN1 mode enable 0 RUN1 mode is disabled. 1 RUN1 mode is enabled.
RUN0	RUN0 mode enable 0 RUN0 mode is disabled. 1 RUN0 mode is enabled.
DRUN	DRUN mode enable 0 DRUN mode is disabled. 1 DRUN mode is enabled.
SAFE	SAFE mode enable 0 SAFE mode is disabled. 1 SAFE mode is enabled.
TEST	TEST mode enable 0 TEST mode is disabled. 1 TEST mode is enabled.
RESET_FUNC	RESET_FUNC mode enable 0 RESET_FUNC mode is disabled. 1 RESET_FUNC mode is enabled. **Note:** This read-only bit is always set to 1.

35.3.2.4 Interrupt Status Register (ME_IS)

Address 0xC3FD_C00C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	I_ICONF_CU	I_ICONF	I_IMODE	I_SAFE	I_MTC
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-5. Interrupt Status Register (ME_IS)

This register provides the current interrupt status.

Table 35-7. Interrupt Status Register (ME_IS) field descriptions

Field	Description
I_ICONF_CU	Invalid mode configuration interrupt (Clock Usage) — This bit is set during a mode transition if a clock that is required to be on by an enabled peripheral is configured to be turned off. It is cleared by writing a 1 to this bit. 0 No invalid mode configuration (clock usage) interrupt occurred. 1 Invalid mode configuration (clock usage) interrupt is pending
I_ICONF	Invalid mode configuration interrupt — This bit is set whenever a write operation to ME_<*mode*>_MC registers with invalid mode configuration is attempted. It is cleared by writing a 1 to this bit. 0 No invalid mode configuration interrupt occurred. 1 Invalid mode configuration interrupt is pending.
I_IMODE	Invalid mode interrupt — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a 1 to this bit. 0 No invalid mode interrupt occurred. 1 Invalid mode interrupt is pending.
I_SAFE	SAFE mode interrupt — This bit is set whenever the device enters SAFE mode on hardware requests generated in the system. It is cleared by writing a 1 to this bit. 0 No SAFE mode interrupt occurred. 1 SAFE mode interrupt is pending.
I_MTC	Mode transition complete interrupt — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a 1 to this bit. This mode transition interrupt bit will not be set while entering low-power modes HALT0 or STOP0. 0 No mode transition complete interrupt occurred. 1 Mode transition complete interrupt is pending.

35.3.2.5 Interrupt Mask Register (ME_IM)

Address 0xC3FD_C010 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	M_ICONF_CU	M_ICONF	M_IMODE	M_SAFE	M_MTC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-6. Interrupt Mask Register (ME_IM)

This register controls whether an event generates an interrupt or not.

Table 35-8. Interrupt Mask Register (ME_IM) field descriptions

Field	Description
M_ICONF_CU	Invalid mode configuration (clock usage) interrupt mask 0 Invalid mode interrupt (clock usage) is masked. 1 Invalid mode interrupt (clock usage) is enabled.
M_ICONF	Invalid mode configuration interrupt mask 0 Invalid mode interrupt is masked. 1 Invalid mode interrupt is enabled.
M_IMODE	Invalid mode interrupt mask 0 Invalid mode interrupt is masked. 1 Invalid mode interrupt is enabled.
M_SAFE	SAFE mode interrupt mask 0 SAFE mode interrupt is masked. 1 SAFE mode interrupt is enabled.
M_MTC	Mode transition complete interrupt mask 0 Mode transition complete interrupt is masked. 1 Mode transition complete interrupt is enabled.

35.3.2.6 Invalid Mode Transition Status Register (ME_IMTS)

Address 0xC3FD_C014 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-7. Invalid Mode Transition Status Register (ME_IMTS)

This register provides the status bits for the possible causes of an invalid mode interrupt.

Table 35-9. Invalid Mode Transition Status Register (ME_IMTS) field descriptions

Field	Description
S_MTI	Mode Transition Illegal status — This bit is set whenever a new mode is requested while some other mode transition process is active (S_MTRANS is 1). Please refer to Section 35.4.5, Mode transition interrupts, for the exceptions to this behavior. It is cleared by writing a 1 to this bit. 0 Mode transition requested is not illegal. 1 Mode transition requested is illegal.
S_MRI	Mode Request Illegal status — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a 1 to this bit. 0 Target mode requested is not illegal with respect to current mode. 1 Target mode requested is illegal with respect to current mode.
S_DMA	Disabled Mode Access status — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a 1 to this bit. 0 Target mode requested is not a disabled mode. 1 Target mode requested is a disabled mode.
S_NMA	Non-existing Mode Access status — This bit is set whenever the target mode requested is one of those non-existing modes determined by ME_ME register. It is cleared by writing a 1 to this bit. 0 Target mode requested is an existing mode. 1 Target mode requested is a non-existing mode.
S_SEA	SAFE Event Active status — This bit is set whenever the device is in SAFE mode, the SAFE event bit is pending, and a new mode has been requested other than RESET/SAFE modes. It is cleared by writing a 1 to this bit. 0 No new mode requested other than RESET/SAFE while SAFE event is pending. 1 New mode requested other than RESET/SAFE while SAFE event is pending.

35.3.2.7 Debug Mode Transition Status Register (ME_DMTS)

Address 0xC3FD_C018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PREVIOUS_MODE				0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	VREG_CSRC_SC	CSRC_CSRC_SC	IRCOSC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	CDP_PRPH_96_127	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-8. Debug Mode Transition Status Register (ME_DMTS)

This register provides the status of different factors that influence mode transitions. It is used to give an indication of why a mode transition indicated by ME_GS.S_MTRANS may be taking longer than expected.

NOTE

The ME_DMTS register does not indicate whether a mode transition is ongoing. Therefore, some ME_DMTS bits may still be asserted after the mode transition has completed.

Table 35-10. Debug Mode Transition Status Register (ME_DMTS) field descriptions

Field	Description
PREVIOUS_MODE	Previous device mode — These bits show the mode the device was in before the change to the current mode. 0000 RESET. 0001 TEST. 0010 SAFE. 0011 DRUN. 0100 RUN0. 0101 RUN1. 0110 RUN2. 0111 RUN3. 1000 HALT0. 1001 Reserved. 1010 STOP0. 1011 Reserved. 1100 Reserved. 1101 Reserved. 1110 Reserved. 1111 Reserved.
MPH_BUSY	MC_ME/MC_PCU Handshake Busy indicator — This bit is set if the MC_ME has requested a mode change from the MC_PCU and the MC_PCU has not yet responded. It is cleared when the MC_PCU has responded. 0 Handshake is not busy. 1 Handshake is busy.
PMC_PROG	MC_PCU Mode Change in Progress indicator — This bit is set if the MC_PCU is in the process of powering up or down power domains. It is cleared when all power-up/power-down processes have completed. 0 Power-up/down transition is not in progress. 1 Power-up/down transition is in progress.
CORE_DBG	Processor is in Debug mode indicator — This bit is set while the processor is in debug mode. 0 The processor is not in debug mode. 1 The processor is in debug mode.
SMR	SAFE mode request from MC_RGM is active indicator — This bit is set if a hardware SAFE mode request has been triggered. It is cleared when the hardware SAFE mode request has been cleared. 0 A SAFE mode request is not active. 1 A SAFE mode request is active.
VREG_CSRC_SC	Main VREG dependent Clock Source State Change during mode transition indicator — This bit is set when a clock source that depends on the main voltage regulator to be powered-up is requested to change its power up/down state. It is cleared when the clock source has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
CSRC_CSRC_SC	(Other) Clock Source dependent Clock Source State Change during mode transition indicator — This bit is set when a clock source that depends on another clock source to be powered-up is requested to change its power up/down state. It is cleared when the clock source has completed its state change. 0 No state change is taking place. 1 A state change is taking place.

Table 35-10. Debug Mode Transition Status Register (ME_DMTS) field descriptions (continued)

Field	Description
IRCOSC_SC	IRCOSC State Change during mode transition indicator — This bit is set when the 16 MHz internal RC oscillator is requested to change its power up/down state. It is cleared when the 16 MHz internal RC oscillator has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
SCSRC_SC	Secondary Clock Sources State Change during mode transition indicator — This bit is set when a secondary clock source is requested to change its power up/down state. It is cleared when all secondary system clock sources have completed their state changes. (A secondary clock source is a clock source other than IRCOSC.) 0 No state change is taking place. 1 A state change is taking place.
SYSCLK_SW	System Clock Switching pending status 0 No system clock source switching is pending. 1 A system clock source switching is pending.
DFLASH_SC	DFLASH State Change during mode transition indicator — This bit is set when the DFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
CFLASH_SC	CFLASH State Change during mode transition indicator — This bit is set when the CFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0…143 — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals that have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.
CDP_PRPH_96_127	Clock Disable Process Pending status for Peripherals 96…127 — This bit is set when any peripheral appearing in ME_PS3 has been requested to have its clock disabled. It is cleared when all these peripherals that have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95 — This bit is set when any peripheral appearing in ME_PS2 has been requested to have its clock disabled. It is cleared when all these peripherals that have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.

Table 35-10. Debug Mode Transition Status Register (ME_DMTS) field descriptions (continued)

Field	Description
CDP_PRPH_32_63	Clock Disable Process Pending status for Peripherals 32…63 — This bit is set when any peripheral appearing in ME_PS1 has been requested to have its clock disabled. It is cleared when all these peripherals that have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.
CDP_PRPH_0_31	Clock Disable Process Pending status for Peripherals 0…31 — This bit is set when any peripheral appearing in ME_PS0 has been requested to have its clock disabled. It is cleared when all these peripherals that have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.

35.3.2.8 RESET Mode Configuration Register (ME_RESET_MC)

Address 0xC3FD_C020 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-9. RESET Mode Configuration Register (ME_RESET_MC)

This register configures system behavior during RESET mode. Please refer to Table 35-11 for details.

35.3.2.9 TEST Mode Configuration Register (ME_TEST_MC)

Address 0xC3FD_C024 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-10. TEST Mode Configuration Register (ME_TEST_MC)

This register configures system behavior during TEST mode. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

35.3.2.10 SAFE Mode Configuration Register (ME_SAFE_MC)

Address 0xC3FD_C028 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-11. SAFE Mode Configuration Register (ME_SAFE_MC)

This register configures system behavior during SAFE mode. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

35.3.2.11 *DRUN* Mode Configuration Register (ME_DRUN_MC)

Address 0xC3FD_C02C — Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-12. DRUN Mode Configuration Register (ME_DRUN_MC)

This register configures system behavior during DRUN mode. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

35.3.2.12 *RUN0…3* Mode Configuration Registers (ME_RUN0…3_MC)

Address 0xC3FD_C030 - 0xC3FD_C03C — Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-13. RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

This register configures system behavior during RUN0…3 modes. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

35.3.2.13 *HALT0* Mode Configuration Register (ME_HALT0_MC)

Address 0xC3FD_C040 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-14. HALT0 Mode Configuration Register (ME_HALT0_MC)

This register configures system behavior during HALT0 mode. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

35.3.2.14 *STOP0* Mode Configuration Register (ME_STOP0_MC)

Address 0xC3FD_C048 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSCON	IRCOSCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 35-15. STOP0 Mode Configuration Register (ME_STOP0_MC)

This register configures system behavior during STOP0 mode. Please refer to Table 35-11 for details.

NOTE

Byte write accesses are not allowed to this register.

Table 35-11. Mode Configuration Registers (ME_<*mode*>_MC) field descriptions

Field	Description
PDO	I/O output power-down control — This bit controls the output power-down of I/Os. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP0 mode, only the pad power sequence driver is disabled, but the state of the output remains functional.
MVRON	Main voltage regulator control — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 0 Reserved. 1 Main voltage regulator is switched on.
DFLAON	Data flash power-down control — This bit specifies the operating mode of the data flash after entering this mode. 00 Reserved. 01 Data flash is in power-down mode. 10 Reserved. 11 Data flash is in normal mode.
CFLAON	Code flash power-down control — This bit specifies the operating mode of the code flash after entering this mode. Code flash cannot be put into low-power or power-down mode if the data flash is in normal mode. 00 Reserved. 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode.
PLL1ON	Secondary PLL control 0 Secondary PLL is switched off. 1 Secondary PLL is switched on. **Note:** The PLL*n*ON bits can be set only if the XOSCON bit is set. The FMPLL cannot be powered up if XOSC is powered down.
PLL0ON	System PLL control 0 System PLL is switched off. 1 System PLL is switched on. **Note:** The PLL*n*ON bits can be set only if the XOSCON bit is set. The FMPLL cannot be powered up if XOSC is powered down.
XOSCON	4-40 MHz crystal oscillator control 0 4-40 MHz crystal oscillator is switched off. 1 4-40 MHz crystal oscillator is switched on.

Table 35-11. Mode Configuration Registers (ME_<*mode*>_MC) field descriptions (continued)

Field	Description
IRCOSCON	16 MHz internal RC oscillator control. This bit is meaningful only in Test Mode. 0 16 MHz internal RC oscillator is switched off. 1 16 MHz internal RC oscillator is switched on.
SYSCLK	System clock switch control — These bits specify the system clock to be used by the system. 0000 16 MHz internal RC oscillator 0001 Reserved. 0010 Reserved. 0011 Reserved. 0100 System PLL. 0101 Reserved. 0110 Reserved. 0111 Reserved. 1000 Reserved. 1001 Reserved. 1010 Reserved. 1011 Reserved. 1100 Reserved. 1101 Reserved. 1110 Reserved. 1111 System clock is disabled in TEST mode, reserved in all other modes.

35.3.2.15 Peripheral Status Register 0 (ME_PS0)

Address 0xC3FD_C060 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	S_FlexRay	0	0	0	0	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-16. Peripheral Status Register 0 (ME_PS0)

This register provides the status of the peripherals. Please refer to Table 35-12 for details.

35.3.2.16 Peripheral Status Register 1 (ME_PS1)

Address 0xC3FD_C064 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	S_CRC0	0	0	0	0	0	0	S_LIN_FLEX3	S_LIN_FLEX2	S_LIN_FLEX1	S_LIN_FLEX0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	S_I2C_DMA2	S_I2C_DMA1	S_I2C_DMA0	0	S_FlexPWM1	S_FlexPWM0	S_eTimer2	S_eTimer1	S_eTimer0	0	0	S_CTU0	0	S_ADC1	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-17. Peripheral Status Register 1 (ME_PS1)

This register provides the status of the peripherals. Please refer to Table 35-12 for details.

35.3.2.17 Peripheral Status Register 2 (ME_PS2)

Address 0xC3FD_C068 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_PIT_RTI	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	S_DRAMC	0	0	0	0	S_EBI	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-18. Peripheral Status Register 2 (ME_PS2)

This register provides the status of the peripherals. Please refer to Table 35-12 for details.

35.3.2.18 Peripheral Status Register 3 (ME_PS3)

Address 0xC3FD_C06C — Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	S_FlexPWM2	0	0	0	0	S_CRC1	S_CTU1	0	S_ADC3	S_ADC2	0	0	0	S_PDI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-19. Peripheral Status Register 3 (ME_PS3)

This register provides the status of the peripherals. Please refer to Table 35-12 for details.

Table 35-12. Peripheral Status Registers 0…4 (ME_PS0…4) field descriptions

Field	Description
S_*<periph>*	Peripheral status — These bits specify the current status of the peripherals in the system. 0 Peripheral is frozen. 1 Peripheral is active.

35.3.2.19 Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Address 0xC3FD_C080 - 0xC3FD_C09C — Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-20. Run Peripheral Configuration Registers (ME_RUN_PC0…7)

These registers configure eight different types of peripheral behavior during non-low-power modes.

Table 35-13. Run Peripheral Configuration Registers (ME_RUN_PC0…7) field descriptions

Field	Description
RUN3	Peripheral control during RUN3 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN2	Peripheral control during RUN2 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN1	Peripheral control during RUN1 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN0	Peripheral control during RUN0 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
DRUN	Peripheral control during DRUN 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
SAFE	Peripheral control during SAFE 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
TEST	Peripheral control during *TEST* 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RESET	Peripheral control during *RESET* 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

35.3.2.20 Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

Address 0xC3FD_C0A0 - 0xC3FD_C0BC Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-21. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

These registers configure eight different types of peripheral behavior during low-power modes.

Table 35-14. Low-Power Peripheral Configuration Registers (ME_LP_PC0…7) field descriptions

Field	Description
STOP0	Peripheral control during STOP0 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
HALT0	Peripheral control during HALT0 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

35.3.2.21 Peripheral Control Registers (ME_PCTL*n*)

Address 0xC3FD_C0C0 - 0xC3FD_C14F Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	0	DBG_F	LP_CFG			RUN_CFG		
W								
Reset	0	0	0	0	0	0	0	0

Figure 35-22. Peripheral Control Registers (ME_PCTL*n*)

These registers select the configurations during non-low-power and low-power modes for each peripheral.

Table 35-15. Peripheral Control Registers (ME_PCTL*n*) field descriptions

Field	Description
DBG_F	Peripheral control in debug mode — This bit controls the state of the peripheral in debug mode 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the device mode. 1 Peripheral is frozen if not already frozen in device modes. **Note:** This feature is useful to freeze the peripheral state while entering debug. For example, this may be used to prevent a reference timer from running while making a debug access.
LP_CFG	Peripheral configuration select for low-power modes — These bits associate a configuration as defined in the ME_LP_PC0...7 registers to the peripheral. 000 Selects ME_LP_PC0 configuration. 001 Selects ME_LP_PC1 configuration. 010 Selects ME_LP_PC2 configuration. 011 Selects ME_LP_PC3 configuration. 100 Selects ME_LP_PC4 configuration. 101 Selects ME_LP_PC5 configuration. 110 Selects ME_LP_PC6 configuration. 111 Selects ME_LP_PC7 configuration.
RUN_CFG	Peripheral configuration select for non-low-power modes — These bits associate a configuration as defined in the ME_RUN_PC0...7 registers to the peripheral. 000 Selects ME_RUN_PC0 configuration. 001 Selects ME_RUN_PC1 configuration. 010 Selects ME_RUN_PC2 configuration. 011 Selects ME_RUN_PC3 configuration. 100 Selects ME_RUN_PC4 configuration. 101 Selects ME_RUN_PC5 configuration. 110 Selects ME_RUN_PC6 configuration. 111 Selects ME_RUN_PC7 configuration.

NOTE

After modifying any of the ME_RUN_PC0...7, ME_LP_PC0...7, and ME_PCTL*n* registers, software must request a mode change and wait for the mode change to be completed before entering debug mode, in order to have consistent behavior between the peripheral clock control process and the clock status reporting in the ME_PS*n* registers.

35.4 Functional description

35.4.1 Mode transition request

The transition from one mode to another mode is normally handled by software accessing the mode control register ME_MCTL. But in the case of special events, the mode transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access the ME_MCTL register twice by:

1. Writing with the value of the key (0x5AF0) into the KEY bit field and the required target mode into the TARGET_MODE bit field.
2. Writing with the inverted value of the key (0xA50F) into the KEY bit field and the required target mode into the TARGET_MODE bit field.

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding ME_<*mode*>_MC register. Depending on the programmed configuration, the mode transition request may require a number of cycles, and software should check the S_CURRENT_MODE bit field and the S_MTRANS bit of the global status register ME_GS to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please refer to Section 35.4.5, Mode transition interrupts.

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately — it will be taken into account on the next request to enter this mode. This means that transition requests such as RUN0…3 → RUN0…3, DRUN → DRUN, SAFE → SAFE, and TEST → TEST are considered valid mode transition requests. As soon as the mode request is accepted as valid, the S_MTRANS bit is set until the status in the ME_GS register matches the configuration programmed in the respective ME_<*mode*>_MC register.

NOTE

It is recommended that software poll the S_MTRANS bit in the ME_GS register after requesting a transition to HALT0 or STOP0 modes.

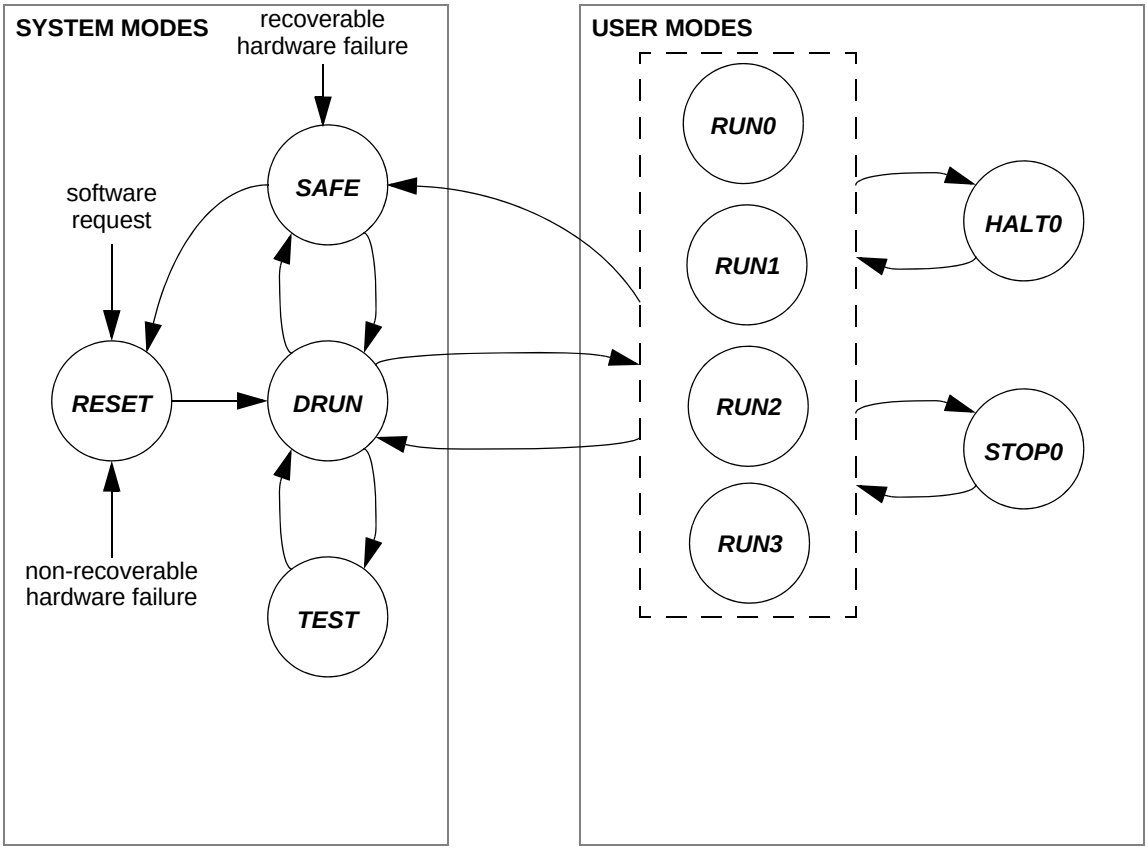


Figure 35-23. MC_ME mode diagram

35.4.2 Mode details

35.4.2.1 RESET mode

The device enters this mode on the following events:

- From SAFE, DRUN, RUN0…3, or TEST mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0000.
- From any mode due to a system reset by the MC_RGM because of some non-recoverable hardware failure in the system (see the MC_RGM chapter for details).

Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished. The mode configuration information for this mode is provided by the ME_RESET_MC register. This mode has a pre-defined configuration, and the 16 MHz internal RC oscillator is selected as the system clock.

35.4.2.2 DRUN mode

The device enters this mode on the following events:

- Automatically from RESET mode after completion of the reset sequence.

- From RUN0…3, SAFE, or TEST mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0011.

As soon as any of the above events has occurred, a DRUN mode transition request is generated. The mode configuration information for this mode is provided by the ME_DRUN_MC register. In this mode, the flashes, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the 16 MHz internal RC oscillator as the system clock.

This mode is intended to be used by software to initialize all registers as per the system needs.

NOTE

Software must ensure that the code executes from RAM before changing to this mode, if the flashes are configured in the low-power or power-down state in this mode.

35.4.2.3 SAFE mode

The device enters this mode on the following events:

- From DRUN, RUN0…3, or TEST mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0010.
- From any mode except RESET due to a SAFE mode request generated by the MC_RGM because of some potentially recoverable hardware failure in the system (see Chapter 14, Clock Generation Module (MC_CGM), for details).

As soon as any of the above events has occurred, a SAFE mode transition request is generated. The mode configuration information for this mode is provided by the ME_SAFE_MC register. This mode has a pre-defined configuration, and the 16 MHz internal RC oscillator is selected as the system clock.

If the SAFE mode is requested by software while some other mode transition process is ongoing, the new target mode becomes the SAFE mode regardless of other pending requests or new requests during the mode transition. No new mode request made during a transition to the SAFE mode will cause an invalid mode interrupt.

NOTE

If software requests a change to SAFE mode and then requests to change back to the parent mode before the mode transition is completed, the device’s final mode after mode transition will be SAFE mode.

As long as a SAFE event is active, the system remains in SAFE mode, and any software mode request during this time is ignored and lost.

This mode is intended to be used by software to assess the severity of the cause of failure and then to either:

- Re-initialize the device via DRUN mode.
- Completely reset the device via RESET mode.

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the PDO bit of the ME_SAFE_MC register should be set. The input levels remain unchanged.

35.4.2.4 TEST mode

The device enters this mode from the DRUN mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0001.

As soon as the above event has occurred, a TEST mode transition request is generated. The mode configuration information for this mode is provided by the ME_TEST_MC register. Except for the main voltage regulator, all resources of the system are configurable in this mode. The system clock to the whole system can be stopped by programming the SYSCLK bit field to 1111, and in this case, the only way to exit this mode is via a device reset.

This mode is intended to be used by software to execute software test routines.

NOTE

Software must ensure that the code executes from RAM before changing to this mode if the flashes are configured in the low-power or power-down state in this mode.

35.4.2.5 RUN0…3 modes

The device enters one of these modes on the following events:

- From the DRUN, or another RUN0…3 mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0100…0111.
- From the HALT0 mode due to an interrupt event.
- From the STOP0 mode due to an interrupt or wakeup event.

As soon as any of the above events has occurred, a RUN0…3 mode transition request is generated. The mode configuration information for these modes is provided by the ME_RUN0…3_MC registers. In these modes, the flashes, all clock sources, and the system clock configuration can be controlled by software as required.

These modes are intended to be used by software to execute application routines.

NOTE

Software must ensure that the code executes from RAM before changing to this mode if the flashes are configured in the low-power or power-down state in this mode.

35.4.2.6 HALT0 mode

The device enters this mode from one of the RUN0…3 modes when the TARGET_MODE bit field of the ME_MCTL register is written with 1000.

As soon as the above event has occurred, a HALT0 mode transition request is generated. The mode configuration information for this mode is provided by the ME_HALT0_MC register. This mode is quite configurable, and the ME_HALT0_MC register should be programmed according to the system requirements. The flashes can be put in low-power mode (code flash only) or power-down mode (code and data flashes) as needed. If a HALT0 mode request occurs while an interrupt request is active, the transition

to HALT0 is aborted. The resultant mode will be the current mode, SAFE (on SAFE mode request), or DRUN (on reset), and an invalid mode interrupt is not generated.

This mode is intended as a first-level low-power mode with:

- The core clock frozen.
- Only a few peripherals running.

This mode is used by software to wait until it is required to do something and then to react quickly (that is, within a few system clock cycles of an interrupt event).

35.4.2.7 STOP0 mode

The device enters this mode from one of the RUN0…3 modes when the TARGET_MODE bit field of the ME_MCTL register is written with 1010.

As soon as the above event has occurred, a STOP0 mode transition request is generated. The mode configuration information for this mode is provided by the ME_STOP0_MC register. This mode is configurable, and the ME_STOP0_MC register should be programmed according to the system requirements. The following clock sources are switched off in this mode:

- System PLL
- Secondary PLL

The flashes can be put in power-down mode as needed. If a STOP0 mode request occurs while any interrupt or wakeup event is active, the transition to STOP0 is aborted with the resultant mode being the current mode, SAFE (on SAFE mode request), or DRUN (on reset). An invalid mode interrupt is not generated.

This mode is intended as an advanced low-power mode with

- The core clock frozen
- Almost all peripherals stopped

This mode is to be used by software to wait until it is required to do something with no need to react quickly (for example, allow for system clock source to be re-started).

This mode can be used to stop all clock sources and thus preserve the device status. When exiting the STOP0 mode, the 16 MHz internal RC oscillator clock is selected as the system clock until the target clock is available.

35.4.3 Mode transition process

The process of mode transition proceeds on the following steps in a pre-defined manner, depending on the current device mode and the requested target mode. In many cases of mode transition, not all steps need to be executed based on the mode control information, and some steps may not be applicable according to the mode definition itself.

35.4.3.1 Target mode request

The target mode is requested by accessing the ME_MCTL register with the required keys. This mode transition request by software must be a valid request satisfying a set of pre-defined rules to initiate the process. If the request fails to satisfy these rules, it is ignored, and the TARGET_MODE bit field is not updated. An optional interrupt can be generated for invalid mode requests. Refer to Section 35.4.5, Mode transition interrupts, for details.

In the case of mode transitions occurring because of hardware events such as a reset, a SAFE mode request, or interrupt requests and wakeup events to exit from low-power modes, the TARGET_MODE bit field of the ME_MCTL register is automatically updated with the appropriate target mode. The mode change process start is indicated by the setting of the mode transition status bit S_MTRANS of the ME_GS register.

A RESET mode requested via the ME_MCTL register is passed to the MC_RGM, which generates a global system reset and initiates the reset sequence. The RESET mode request has the highest priority, and the MC_ME is kept in the RESET mode during the entire reset sequence.

The SAFE mode request has the next highest priority after reset. It can be generated either by software via the ME_MCTL register from all software running modes including DRUN, RUN0…3, and TEST, or by the MC_RGM after the detection of system hardware failures, which may occur in any mode.

35.4.3.2 Target mode configuration loading

On completion of the Target mode request step, the target mode configuration from the ME_<*target mode*>_MC register is loaded to start the resources (voltage sources, clock sources, flashes, pads, etc.) control process.

An overview of resource control possibilities for each mode is shown in Table 35-16. A ‘√’ indicates that a given resource is configurable for a given mode.

Table 35-16. MC_ME resource control overview

Resource	Mode						
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT0	STOP0
IRCOSC	on	√ on	on	on	on	√ on	on
XOSC	off	√ off	off	√ off	√ off	√ off	√ off
PLL0	off	√ off	off	√ off	√ off	√ off	off
PLL1	off	√ off	off	√ off	√ off	√ off	off

Table 35-16. MC_ME resource control overview (continued)

Resource	Mode						
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT0	STOP0
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ normal	√ power-down
MVREG	on	on	on	on	on	on	on
PDO	off	√ off	√ on	off	off	off	√ off

35.4.3.3 Peripheral clocks disable

On completion of the Target mode request step, the MC_ME requests each peripheral to enter its stop mode when the peripheral is configured to be disabled via:

- The target mode
- The peripheral configuration registers ME_RUN_PC0…7 and ME_LP_PC0…7
- The peripheral control registers ME_PCTL*n*

CAUTION

The MC_ME does not automatically request peripherals to enter their stop modes if the power domains in which they are residing are to be turned off due to a mode change. Therefore, it is software's responsibility to ensure that those peripherals that are to be powered down are configured in the MC_ME to be frozen.

Each peripheral acknowledges its stop mode request after closing its internal activity. The MC_ME then disables the corresponding clock(s) to this peripheral.

In the case of a SAFE mode transition request, the MC_ME does not wait for the peripherals to acknowledge the stop requests. The SAFE mode clock gating configuration is applied immediately, without regard to any acknowledgement by the peripherals.

Please refer to Section 35.4.6, Peripheral clock gating, for more details.

Each peripheral that may block or disrupt a communication bus to which it is connected ensures that these outputs are forced to a safe or recessive state when the device enters the SAFE mode.

35.4.3.4 Processor low-power mode entry

If, on completion of the Peripheral clocks disable step, the mode transition is to the HALT0 mode, the MC_ME requests the processor to enter its halted state. The processor acknowledges its halt state request after completing all outstanding bus transactions.

If, on completion of the Peripheral clocks disable step, the mode transition is to the STOP0 mode, the MC_ME requests the processor to enter its stopped state. The processor acknowledges its stop state request after completing all outstanding bus transactions.

35.4.3.5 Processor and system memory clock disable

If, on completion of the Processor low-power mode entry step, the mode transition is to the HALT0 or STOP0 mode and the processor is in its appropriate halted or stopped state, the MC_ME disables the processor and system memory clocks to achieve further power saving.

The clocks to the processor and system memory are unaffected while transitioning between software-running modes such as DRUN, RUN0…3, and SAFE.

CAUTION

Clocks to the whole device including the processor and system memory can be disabled in TEST mode.

35.4.3.6 Clock sources switch-on

On completion of the Processor low-power mode entry step, the MC_ME switches on all clock sources based on the <*clock source*>ON bits of the ME_<*current mode*>_MC and ME_<target mode>_MC registers. The following clock sources are switched on at this step:

- 16 MHz internal RC oscillator
- 4-40 MHz crystal oscillator
- System PLL (PLL0)
- Secondary PLL (PLL1)

The clock sources that are required by the target mode are switched on. The duration required for the output clocks to be stable depends on the type of source, and all further steps of mode transition depending on one or more of these clocks waits for the stable status of the respective clocks. The availability status of these clocks is updated in the S_<*clock source*> bits of ME_GS register.

The clock sources that need to be switched off are unaffected during this process in order to not disturb the system clock, which might require one of these clocks before switching to a different target clock.

35.4.3.7 Flash modules switch-on

On completion of the step, if one or more of the flashes needs to be switched to normal mode from its low-power mode (code flash only) or power-down mode (code and data flashes) based on the CFLAON and DFLAON bit fields of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, the MC_ME requests the flash to exit from its low-power/power-down mode. When the flashes are available for access, the S_CFLA and S_DFLA bit fields of the ME_GS register are updated to 11 by hardware.

CAUTION

It is illegal to switch the flashes from low-power mode to power-down mode and from power-down mode to low-power mode. The MC_ME, however, does not prevent this nor does it flag it.

35.4.3.8 Pad outputs-on

On completion of the step, if the PDO bit of the ME_<*target mode*>_MC register is cleared, then

- All pad outputs are enabled to return to their previous state.
- The I/O pads power sequence driver is switched on.

35.4.3.9 Peripheral clocks enable

Based on the current and target device modes, the peripheral configuration registers ME_RUN_PC0…7, ME_LP_PC0…7, and the peripheral control registers ME_PCTL*n*, the MC_ME enables the clocks for selected modules as required. This step is executed only after the process is completed.

35.4.3.10 Processor and memory clock enable

If the mode transition is from any of the low-power modes HALT0 or STOP0 to RUN0…3, the clocks to the processor and system memory are enabled. The process of enabling these clocks is executed only after the Flash modules switch-on process is completed.

35.4.3.11 Processor low-power mode exit

If the mode transition is from any of the low-power modes HALT0 or STOP0 to RUN0…3, the MC_ME requests the processor to exit from its halted or stopped state. This step is executed only after the Processor and memory clock enable process is completed.

CAUTION

When executing from the flash and entering a Low-Power Mode (LPM) where the flash is in low-power or power-down mode, 2-4 clock cycles exist at the beginning of the RUNx to LPM transition during which a wakeup or interrupt will generate a checkstop due to the flash not being available on RUNx mode re-entry. This will cause either a checkstop reset or machine check interrupt.

This scenario can be avoided by configuring the application to handle the machine check interrupt in RAM dealing with the problem only if it occurs. Or by configure the MCU to avoid the machine check interrupt, executing the transition into low power modes in RAM.

There is no absolute requirement to work around the possibility of a checkstop reset if the application can accept the reset, and associated delays, and continue. In this event, the WKPU.WISR will not indicate the channel that triggered the wakeup though the F_CHKSTOP flag will indicate that the reset has occurred. The F_CHKSTOP flag could still be caused by other error conditions so the startup strategy from this condition should be considered alongside any pre-existing strategy for recovering from an F_CHKSTOP condition.

35.4.3.12 System clock switching

Based on the SYSCLK bit field of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, if the target and current system clock configurations differ, the following method is implemented for clock switching.

- The target clock configuration for the 16 MHz internal RC oscillator takes effect only after the S_IRCOSC bit of the ME_GS register is set by hardware (that is, the 16 MHz internal RC oscillator has stabilized).
- The target clock configuration for the System PLL takes effect only after the S_PLL0 bit of the ME_GS register is set by hardware (that is, the System PLL has stabilized).
- If the clock is to be disabled, the SYSCLK bit field should be programmed with 1111. This is possible only in TEST mode.

The current system clock configuration can be observed by reading the S_SYSCLK bit field of the ME_GS register, which is updated after every system clock switching. Until the target clock is available, the system uses the previous clock configuration.

System clock switching starts only after:

- The Clock sources switch-on process has completed if the target system clock source is one of the following:
  - The 16 MHz internal RC oscillator
  - The System PLL. (PLL0)
- The Peripheral clocks disable process has completed in order not to change the system clock frequency before peripherals close their internal activities.

An overview of system clock source selection possibilities for each mode is shown in Table 35-17. A ‘√’ indicates that a given clock source is selectable for a given mode.

Table 35-17. MC_ME system clock selection overview

System clock Source	Mode						
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT0	STOP0
16 MHz internal RC oscillator	√	√ (default)	√	√ (default)	√ (default)	√ (default)	√
System PLL		√		√	√	√	
system clock is disabled		√[1]					

[1] Disabling the system clock during TEST mode will require a reset in order to exit TEST mode.

35.4.3.13 Pad switch-off

If the PDO bit of the ME_<target mode>_MC register is 1 then the outputs of the pads are forced to the high impedance state if the target mode is SAFE or TEST. This step is executed only after the Peripheral clocks disable process has completed.

35.4.3.14 Clock sources (with no dependencies) switch-off

Based on the device mode and the <*clock source*>ON bits of the ME_<*mode*>_MC registers, if a given clock source is to be switched off and no other clock source needs it to be on, the MC_ME requests the clock source to power down and updates its availability status bit S_<*clock source*> of the ME_GS register to 0. The following clock sources are switched off at this step:

- System PLL (PLL0)
- Secondary PLL (PLL1)

This step is executed only after the System clock switching process has completed.

35.4.3.15 Clock sources (with dependencies) switch-off

Based on the device mode and the <*clock source*>ON bits of the ME_<*mode*>_MC registers, if a given clock source is to be switched off and all clock sources that need this clock source to be on have been switched off, the MC_ME requests the clock source to power down and updates its availability status bit S_<*clock source*> of the ME_GS register to 0. The following clock sources are switched off at this step:

- 16 MHz internal RC oscillator
- 4-40 MHz crystal oscillator

This step is executed only after:

- The System clock switching process has completed, in order not to lose the current system clock during mode transition.

- The Clock sources (with no dependencies) switch-off process has completed in order to, for example, prevent unwanted lock transitions.

35.4.3.16 Flash switch-off

Based on the CFLAON and DFLAON bit fields of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, if any of the flashes is to be put in low-power mode (code flash only) or power-down mode (code and data flashes), the MC_ME requests the flash to enter the corresponding power mode and waits for the flash to acknowledge. The exact power mode status of the flashes is updated in the S_CFLA and S_DFLA bit fields of the ME_GS register. This step is executed only when the Processor and system memory clock disable process has completed.

35.4.3.17 Current mode update

The current mode status bit field S_CURRENT_MODE of the ME_GS register is updated with the target mode bit field TARGET_MODE of the ME_MCTL register when:

- All updated status bits in the ME_GS register match the configuration specified in the ME_<*target mode*>_MC register.
- Power sequences are done.
- Clock disable/enable process is finished.
- Processor low-power mode (halt/stop) entry and exit processes are finished.

Software can monitor the mode transition status by reading the S_MTRANS bit of the ME_GS register. The mode transition latency can differ from one mode to another depending on the resources' availability before the new mode request and the target mode's requirements.

If a mode transition is taking longer to complete than expected, the ME_DMTS register can indicate which process is still in progress.

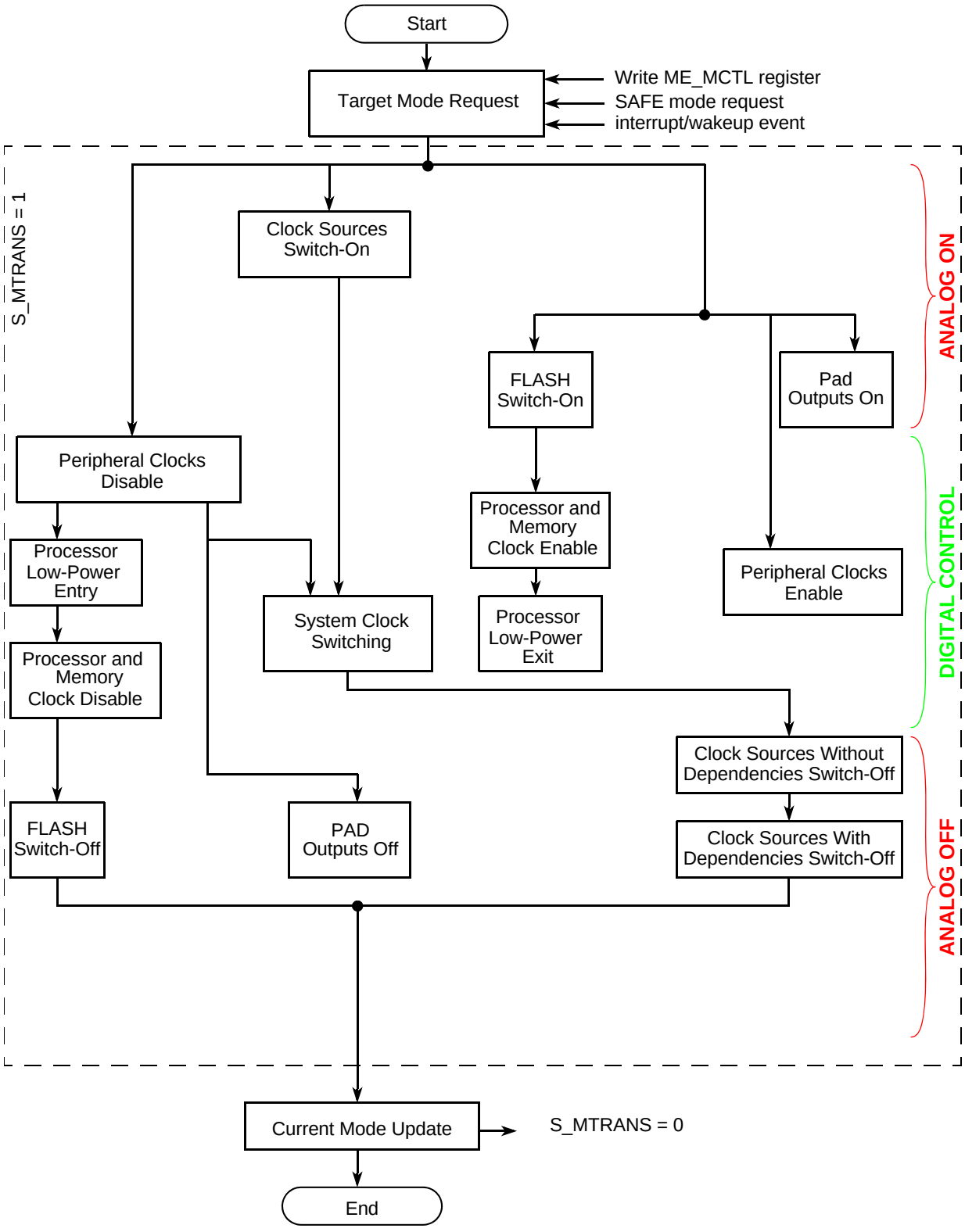


Figure 35-24. MC_ME transition diagram

35.4.4 Protection of mode configuration registers

While programming the mode configuration registers ME_<*mode*>_MC, the following rules must be respected. Otherwise, the write operation is ignored and an invalid mode configuration interrupt may be generated.

- If the 16 MHz internal RC oscillator is selected as the system clock, IRCOSC must be on.
- If the System PLL clock is selected as the system clock, PLL0 must be on.
- If PLL0 is on, XOSC must also be on.
- If PLL1 is on, XOSC must also be on.

NOTE

Software must ensure that clock sources with dependencies other than those mentioned above are switched on as needed. There is no automatic protection mechanism to check this in the MC_ME.

- Configuration 00 for the CFLAON and DFLAON bit fields is reserved.
- Configuration 10 for the DFLAON bit field is reserved.
- If the DFLAON bit field is set to 11, the CFLAON field must also be set to 11.
- System clock configurations marked as ‘reserved’ may not be selected.
- Configuration 1111 for the SYSCLK bit field is allowed only for TEST mode, and only in this case may all system clock sources be turned off.

CAUTION

If the system clock is stopped during TEST mode, the device can exit only via a system reset.

35.4.5 Mode transition interrupts

The MC_ME provides interrupts for:

- Incorrectly configuring a mode.
- Requesting an invalid mode transition.
- Indicating a SAFE mode transition not due to a software request.
- Indicating when a mode transition has completed.

35.4.5.1 Invalid mode configuration interrupt

Whenever a write operation is attempted to the ME_<*mode*>_MC registers that violates the protection rules mentioned in Section 35.4.4, Protection of mode configuration registers, the interrupt pending bit I_ICONF of the ME_IS register is set and an interrupt request is generated if the mask bit M_ICONF of ME_IM register is 1.

35.4.5.2 Invalid mode transition interrupt

The mode transition request is considered invalid under the following conditions:

- If the system is in the SAFE mode and the SAFE mode request from MC_RGM is active, and if the target mode requested is other than RESET or SAFE, then this new mode request is considered to be invalid, and the S_SEA bit of the ME_IMTS register is set.
- If the TARGET_MODE bit field of the ME_MCTL register is written with a value different from the specified mode values (that is, a non-existing mode), an invalid mode transition event is generated. When such a non-existing mode is requested, the S_NMA bit of the ME_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME_MCTL register.
- If some of the device modes are disabled as programmed in the ME_ME register, their respective configurations are considered reserved, and any access to the ME_MCTL register with those values results in an invalid mode transition request. When such a disabled mode is requested, the S_DMA bit of the ME_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME_MCTL register.
- If the target mode is not a valid mode with respect to the current mode, the mode request illegal status bit S_MRI of the ME_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (the S_MTRANS bit of the ME_GS register is 1), the mode transition illegal status bit S_MTI of the ME_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.

NOTE

Because the causes of invalid mode transitions may overlap at the same time, the priority implemented for invalid mode transition status bits of the ME_IMTS register in order from highest to lowest is S_SEA, S_NMA, S_DMA, S_MRI, and S_MTI.

As an exception, the mode transition request is not considered as invalid under the following conditions:

- A new request is allowed to enter the RESET or SAFE mode irrespective of the mode transition status.
- As the exit of HALT0 and STOP0 modes depends on the interrupts of the system, which can occur at any instant, these requests to return to RUN0…3 modes are always valid.
- In order to avoid any unwanted lockup of the device modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined reasonable amount of time for any reason. The parent mode is the device mode before a valid mode request was made.
- Self-transition requests (for example, RUN0 → RUN0) are not considered as invalid even when the mode transition process is active (that is, S_MTRANS is 1). During the low-power mode exit process, if the system is not able to enter the respective RUN0…3 mode properly (that is, all status bits of the ME_GS register match with configuration bits in the ME_<*mode*>_MC register), then software can only request SAFE or RESET mode. It is not possible to request any other mode or to go back to the low-power mode again.

Whenever an invalid mode request is detected, the interrupt pending bit I_IMODE of the ME_IS register is set, and an interrupt request is generated if the mask bit M_IMODE of the ME_IM register is 1.

35.4.5.3 SAFE mode transition interrupt

Whenever the system enters SAFE mode as a result of a SAFE mode request from the MC_RGM due to a hardware failure, the interrupt pending bit I_SAFE of the ME_IS register is set, and an interrupt is generated if the mask bit M_SAFE of ME_IM register is 1.

The SAFE mode interrupt pending bit can be cleared only when the SAFE mode request is deasserted by the MC_RGM (see the MC_RGM chapter for details on how to clear a SAFE mode request). If the system is already in SAFE mode, any new SAFE mode request by the MC_RGM also sets the interrupt pending bit I_SAFE. However, the SAFE mode interrupt pending bit is not set when the SAFE mode is entered by a software request (that is, programming of ME_MCTL register).

35.4.5.4 Mode transition complete interrupt

Whenever the system fully completes a mode transition (that is, the S_MTRANS bit of ME_GS register transits from 1 to 0), the interrupt pending bit I_MTC of the ME_IS register is set, and an interrupt request is generated if the mask bit M_MTC of the ME_IM register is 1. The interrupt bit I_MTC is not set when entering low-power modes HALT0 and STOP0 in order to avoid the same event requesting the immediate exit of these low-power modes.

35.4.6 Peripheral clock gating

During all device modes, each peripheral can be associated with a particular clock gating policy determined by two groups of peripheral configuration registers.

The run peripheral configuration registers ME_RUN_PC0…7 are chosen only during the software running modes DRUN, TEST, SAFE, and RUN0…3. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit that determines whether or not a peripheral clock is to be gated. Run configuration selection for each peripheral is done by the RUN_CFG bit field of the ME_PCTL*n* registers.

The low-power peripheral configuration registers ME_LP_PC0…7 are chosen only during the low-power modes HALT0 and STOP0. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit that determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the LP_CFG bit field of the ME_PCTL*n* registers.

Any modifications to the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL*n* registers do not affect the clock gating behavior until a new mode transition request is generated.

Whenever the processor enters a debug session during any mode, the following occurs for each peripheral:

- The clock is gated if the DBG_F bit of the associated ME_PCTL*n* register is set. Otherwise, the peripheral clock gating status depends on the RUN_CFG and LP_CFG bits. Any further modifications of the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL*n* registers during a debug session will take affect immediately without requiring any new mode request.

35.4.7 Application example

Figure 35-25 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

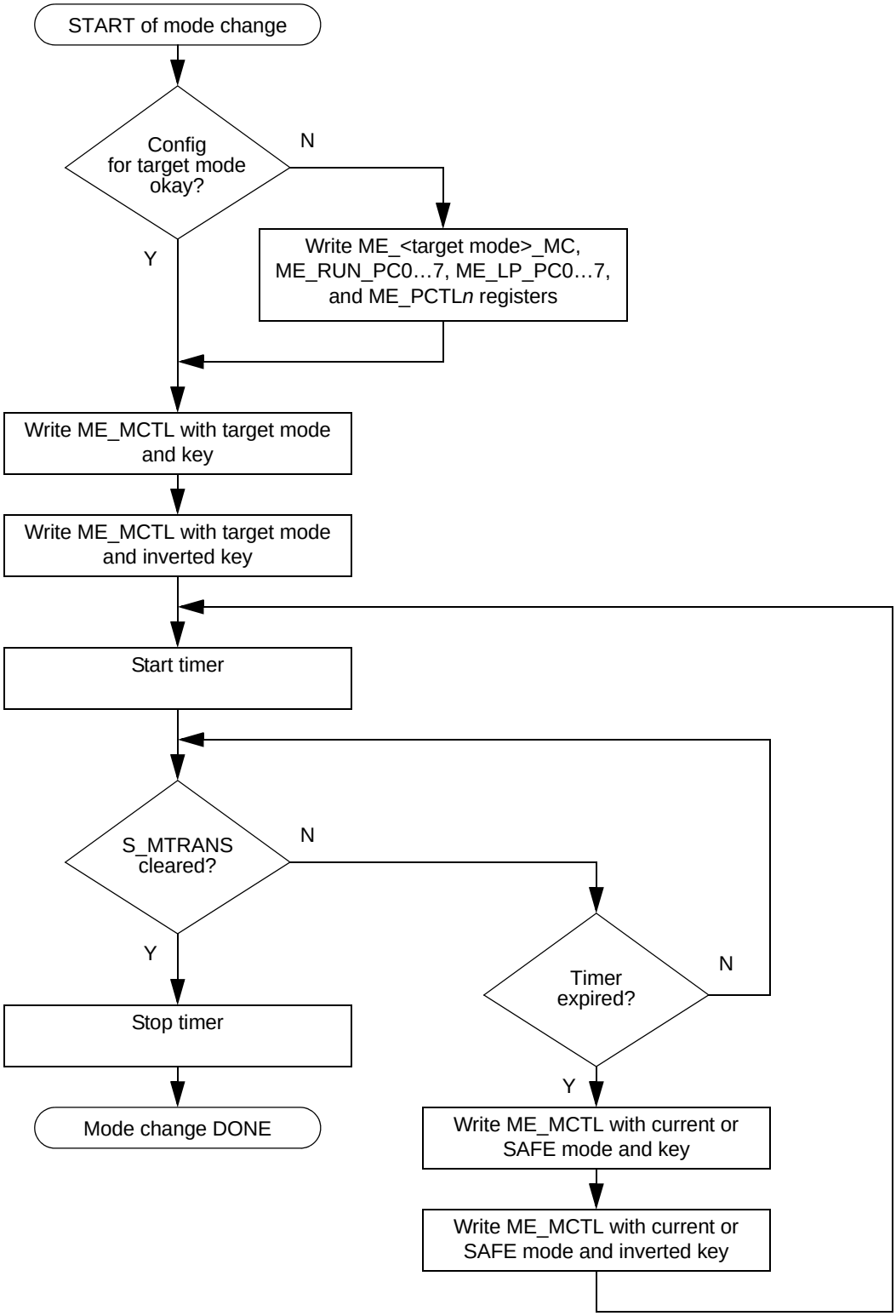


Figure 35-25. MC_ME application example flow diagram

Chapter 36
Multi-Port DDR DRAM Controller (MDDRC)

36.1 Introduction

This chapter describes the multi-port DDR DRAM controller (MDDRC).

The MDDRC is a multi-port DRAM controller (three ports). It supports Mobile-DDR[1], DDR-1, DDR-2, and SDR memories.

A block diagram of the multi-port DRAM controller is shown in Figure 36-1.

36.1.1 Overview

The DRAM controller is a multi-port controller that listens to incoming requests on the three incoming buses and decides on each rising clock edge what command needs to be sent to the DRAM. Each incoming bus is a 64-bit AHB bus.

The block supports connection of one DRAM rank (one chip selects) and supports the four major classes of DRAM:

- Mobile-DDR (LPDDR)
- DDR1
- DDR2[2]
- SDR

It supports these memories in 16-bit wide configurations.

The DRAM controller listens to the incoming requests to all the buses in parallel and then sends commands to the DRAM from the highest priority bus at the current time, while the DRAM is ready to receive the command from this particular bus. If the DRAM is blocked because it needs to meet a timing requirement, the controller sends a command from a bus where there is no blockage.

For example, suppose bus one has an incoming request on priority five, and it hits in bank 1 and the page is not open (the bank needs a precharge + activate command before the request can be serviced). Bus two has an incoming request on priority four, it hits in bank two and the correct page is already open. In this case, the DRAM controller accepts the bus two request first. While it is reading from the appropriate bank, it issues the active + precharge command for the bus one request. Because the DRAM controller sees it cannot issue the read for the bus one request (the bank needs precharge + activate), it takes the bus two request first. Because it can issue the read, the correct page is open. During this, it issues the precharge + activate for the bus 1 request in the background. This request does not suffer from the bus two request being serviced first.

The embedded priority manager determines the relative priority of each bus, and this is used by the DRAM controller to determine which requests are more urgent.

1. The JEDEC standard calls these LPDDR. Most DRAM vendors call them Mobile-DDR.
2. DDR2 can be used only if its minimum frequency is 90 MHz.

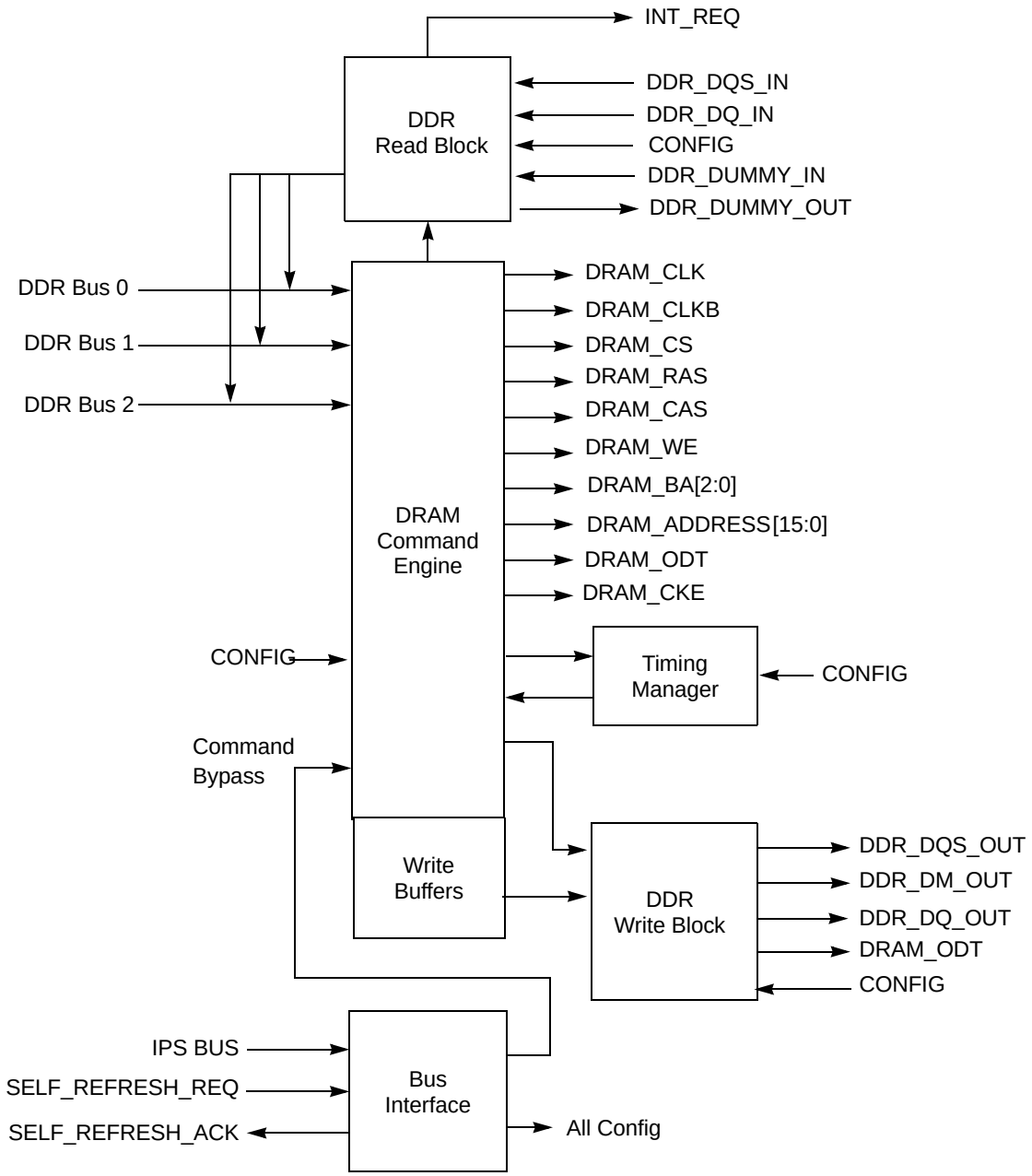


Figure 36-1. Block Diagram of the Multi-port DRAM Controller

36.1.2 Debug mode

When the MCU is in debug mode, the MDDRC behavior is unaffected and remains dictated by the mode of the MDDRC.

36.2 Features

- Supports CAS latency of 2, 3, or 4 clock cycles
- Master buses

  - Three incoming master buses
  - Supports 8-byte and 16-byte bursts
  - Supports byte enables
  - Supports 4-bit priority signal for each bus.
- Arbitration protocol
  - Inside the arbiter block, six different arbiters take out the highest priority request in a certain class. All the arbiters are DRAM state aware, meaning they disregard requests that cannot be sent to the DRAM because of DRAM timing limitations.
    - Arbiter 1: Looks for highest priority read command
    - Arbiter 2: Looks for highest priority write command
    - Arbiter 3: Looks for highest priority activate-for-read command
    - Arbiter 4: Looks for highest priority activate-for-write command
    - Arbiter 5: Looks for highest priority precharge-for-read command
    - Arbiter 6: Looks for highest priority precharge-for-write command
  - After the first prioritization, the next round of arbitrating between the different arbiters is done. A fixed-priority schema is followed:
    - Read and write commands have highest priority
    - Activate has next-highest priority
    - Precharge has lowest priority
    - The DRAM is in read or write mode. In read mode, reads have priority over writes. In write mode, writes have priority over read.
    - DRAM only switches from read to write mode or vice-versa if:

      A high-priority write is found, and the write buffer is full.

      A high-priority read is found.

      The device is in read mode, but no more reads are pending

      The device is in write mode, but no more writes are pending
- Write buffer contains four 16-byte entries
- Supports 16-bit-wide Mobile-DDR/DDR1/DDR2 and SDR DRAM devices
- Controller supports one chip select, with 8 banks
- Supports dynamic on-die termination in the host device and in the DRAM

36.3 Memory map and register description

36.3.1 Memory map

Table 36-1. MDDRC memory map

Offset from MDDRC_BASE (0xC3F9_8000)	Register	Access[1]	Reset value[2]	Section/page
0x0000	DDR System Configuration Register (DDR_SYS_CONFIG)	R/W	0x1000_0000	on page 1351
0x0004	DDR Time Config0 Register (DDR_TIME_CONFIG0)	R/W	0x0000_0000	on page 1355
0x0008	DDR Time Config1 Register (DDR_TIME_CONFIG1)	R/W	0x0000_0000	on page 1356
0x000C	DDR Time Config2 Register (DDR_TIME_CONFIG2)	R/W	0x0000_0000	on page 1357
0x0010	DRAM Command Register (DDR_COMMAND)	R/W	0x0000_0000	on page 1361
0x0014	Compact Command Register (DDR_COMPACT_COMMAND)	R/W	0x0000_0000	on page 1362
0x0018	Self-Refresh Command Register 0 (SELF_REFRESH_CMD_0)	R/W	0x0000_0000	on page 1364
0x001C	Self-Refresh Command Register 1 (SELF_REFRESH_CMD_1)	R/W	0x0000_0000	on page 1364
0x0020	Self-Refresh Command Register 2 (SELF_REFRESH_CMD_2)	R/W	0x0000_0000	on page 1364
0x0024	Self-Refresh Command Register 3 (SELF_REFRESH_CMD_3)	R/W	0x0000_0000	on page 1364
0x0028	Self-Refresh Command Register 4 (SELF_REFRESH_CMD_4)	R/W	0x0000_0000	on page 1364
0x002C	Self-Refresh Command Register 5 (SELF_REFRESH_CMD_5)	R/W	0x0000_0000	on page 1364
0x0030	Self-Refresh Command Register 6 (SELF_REFRESH_CMD_6)	R/W	0x0000_0000	on page 1364
0x0034	Self-Refresh Command Register 7 (SELF_REFRESH_CMD_7)	R/W	0x0000_0000	on page 1364
0x0038	DQS Config Offset Count Register (DQS_CONFIG_OFFSET_COUNT)	R/W	0x0000_0000	on page 1365
0x003C	DQS Config Offset Time Register (DQS_CONFIG_OFFSET_TIME)	R/W	0x0000_0000	on page 1365
0x0040	DQS Delay Status Register (DQS_DELAY_STATUS)	R	0x0UUU_0UUU	on page 1366
0x0044	DDR Auxiliary Configuration Register (DDR_AUX_CONFIG)	R/W	0x0000_0000	on page 1366
0x0048–0x005F	Reserved			
0x0060	DDR Extra Attributes Register	R/W	0x0000_0000	on page 1367
0x0064–0x007F	Reserved			

Table 36-1. MDDRC memory map (continued)

Offset from MDDRC_BASE (0xC3F9_8000)	Register	Access[1]	Reset value[2]	Section/page
0x0080–0x0148	Priority Manager (PRIOMAN) registers—see Chapter 37, Multi-Port DRAM Controller Priority Manager (PRIOMAN)			
0x014C–0x03FF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

36.3.2 Register descriptions

36.3.2.1 DDR System Configuration Register (DDR_SYS_CONFIG)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RST_B	CKE	CLK ON	CMD MODE	DRAM_ROW_SELECT				DRAM_BANK_SELECT			READ_TEST		SELF REF EN	0	RDLY[3]
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RDLY[2:0]			HALF DQS DLY	QUART DQS DLY	WDLY			EARLY ODT	ON DIE TERMINATE	FIFO OV PEND	FIFO UV PEND	0	0	FIFO OV EN	FIFO UV EN
W											w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-2. DDR System Configuration Register (DDR_SYS_CONFIG)

Table 36-2. DDR_SYS_CONFIG field descriptions

Field	Description
RST_B	DRAM controller soft reset. When this bit is 0, the DRAM controller is in the reset state. When this bit is 1, the DRAM controller is out of reset. The bit controls the reset to the internal state machines. The configuration registers are reset by the resets from the hardware reset block, not by this bit.

Table 36-2. DDR_SYS_CONFIG field descriptions (continued)

Field	Description
CKE	Value on the DRAM CKE pin. For functional operation, this needs to be high. During power-down, value can be low.
CLK ON	0 The DRAM clock is stopped. 1 The DRAM clock is running.
CMD MODE	When this bit is 0, the DRAM controller is in normal operation. When this bit is 1, the DRAM controller is in command mode and does not respond to requests on the incoming buses. Command mode is used for DRAM initialization and to switch the DRAM into and out of the different power-down and self-refresh modes.
DRAM_ROW_SELECT and DRAM_BANK_SELECT	These fields control the multiplexing of the bus address to the DRAM bank and row address. Table 36-4 and Table 36-6 give the details. DRAM column address is given in Table 36-5.
READ_TEST	These fields are for production test. Do not use.
SELFREFEN	Self-refresh enable. When this bit is 1, the DRAM controller autonomously enters and exits the self-refresh using the self-refresh command registers when requested by the PMC. When the bit is 0, the transition is blocked.
RDLY	This field controls the expected delay between sending a read command to the DRAM and receiving the read data from the DRAM. RDLY, HALF DQS DLY, and QUART DQS delay together to code for t_{DQSEN}. The t_{DQSEN} is the delay between the read command and when the internal DQS enable goes high. See Figure 36-3. Timing is internally compensated, and is referred to timing at the device pins. t_{DQSEN} should be selected so the L-H transition of DQS enable is always in the preamble of the DQS input of the READ command. Required t_{DQSEN} value depends on the CAS latency (CL), the distance between the DRAM and the device, and the type of DRAM used. Table 36-3 gives the detail on programming t_{DQSEN}.
HALF DQS DLY	This field is an extra field to control the expected read delay between issuing the read command and getting read data from the DRAM. This field offers 1/2 System Bus Clock cycle granularity when programming the delay. See description of field RDLY for details.
QUART DQS DLY	This field is an extra field to control the expected read delay between issuing the read command, and getting read data from the DRAM. This field offers 1/4 System Bus Clock cycle granularity when programming the delay. See the description of the RDLY field.
WDLY	This field controls the write latency (WL) for write commands.

WDLY	Write Latency (System Bus Clock Cycles)
0b001	1
0b010	2
0b011	3
0b100	4
Note: For SDR mode, WDLY must be 001.	

Table 36-2. DDR_SYS_CONFIG field descriptions (continued)

Field	Description
EARLY ODT	This bit controls the assertion of the MODT signal when on-die termination (ODT) is used with DDR2 DRAM. If set to 0 the MODT is asserted two clocks before the write data. If set to 1 the MODT is asserted three clocks before the write data. It may be necessary, depending on the time required to turn on termination resistors in the memory device, to set EARLY_ODT=1.Doing so will insert an extra clock cycle between non-consecutive back-to-back write cycles but has no other performance impact.
ON DIE TERMINATE	This bit controls on-die termination (ODT) in the controller. If this bit is 1, the internal pads generate ODT during read. If the bit is 0, no ODT is provided. The ODT in the DRAM is controlled via the DRAM internal configuration registers. Please consult DRAM data sheet for it.
FIFOOVPENDING FIFOUVPENDING FIFOOVEN FIFOUVEN	These bits control the interrupt generation by the DRAM controller. The DRAM controller has two interrupts: FIFO OV pending and FIFO UV pending. These interrupts are set on overflow or underflow of the FIFO in the read block. When a read command is sent to the DRAM, it is entered into a FIFO. The DRAM is expected to answer by sending back the read data with some up and down edges on the DQS lines (the DQS strobes) used to clock the data. The DRAM controller clocks the read data with the DQS strobes supplied by the DRAM and retrieves the read command from the FIFO after receiving the correct number of read strobes. When the read data strobes returned by the DRAM do not match the expectations of the controller, the FIFO may underflow (if too many DQS strobes are coming back from the DRAM) or overflow (if not enough DQS strobes are coming back). These underflows and overflows are basically the result of problems with the DRAM interface or incorrect parameter settings in the controller or the DRAM. Care has been taken during the design of the DRAM controller not to enter a hang-up state when this occurs. However, read data is corrupt and CPU is informed via the FIFO overflow and FIFO underflow interrupts. The issue is also discussed in Section 36.4.7, Bus interface. • FIFO_OV_PENDING and FIFO_UV_PENDING signal to the CPU if an overflow or underflow interrupt is pending. Writing 1 to these bits clears the corresponding interrupt. • FIFO_OV_EN and FIFO_UV_EN bits are interrupt enable bits. If the pending + enable bit is set at the same time, the interrupt is sent to the CPU.

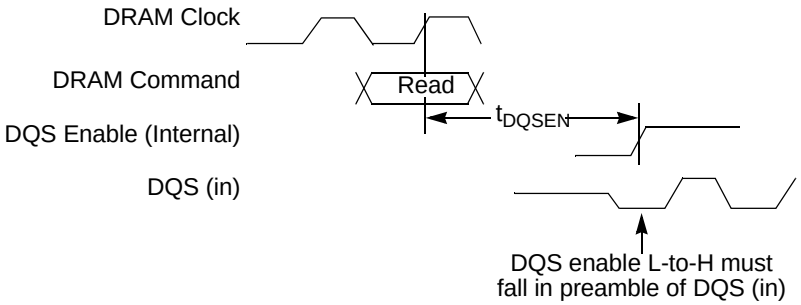


Figure 36-3. t_{DQSEN}

Table 36-3. Programming t_{DQSEN}

{rdly,half_dqs_dly,quart_dqs_dly}	t_{DQSEN} (System bus clock cycles)
1000 0 0	0.5
1000 0 1	0.75

Table 36-3. Programming t_{DQSEN} (continued)

{rdly,half_dqs_dly,quart_dqs_dly}	t_{DQSEN} (System bus clock cycles)
1000 1 0	1.0
1000 1 1	1.25
0100 0 0	1.5
0100 0 1	1.75
0100 1 0	2.0
0100 1 1	2.25
0010 0 0	2.5
0010 0 1	2.75
0010 1 0	3.0
0010 1 1	3.25
0001 0 0	3.5
0001 0 1	3.75
0001 1 0	4.0
0001 1 1	4.25
0000 0 0	4.5
0000 0 1	4.75
0000 1 0	5.0
0000 1 1	5.25

Table 36-4. Number of DRAM banks addressed and mapping of address to DRAM bank address

DRAM_BANK_SELECT	Number of Banks	DRAM Bank Address
0	4	DRAM_BANK[1:0] = address[11:10]
1	4	DRAM_BANK[1:0] = address[12:11]
2	8	DRAM_BANK[2:0] = address[13:11]
3	4	DRAM_BANK[1:0] = address[13:12]
4	8	DRAM_BANK[2:0] = address[14:12]
5	4	DRAM_BANK[1:0] = address[14:13]
6	8	DRAM_BANK[2:0] = address[15:13]
7	4	DRAM_BANK[1:0] = address[15:14]
8	8	DRAM_BANK[2:0] = address[16:14]
9	4	DRAM_BANK[1:0] = address[25:24]
10	8	DRAM_BANK[2:0] = address[26:24]
11	4	DRAM_BANK[1:0] = address[26:25]
12	8	DRAM_BANK[2:0] = address[27:25]
13	8	DRAM_BANK[2:0] = address[28:26]

Table 36-4. Number of DRAM banks addressed and mapping of address to DRAM bank address (continued)

DRAM_BANK_SELECT	Number of Banks	DRAM Bank Address
14	8	DRAM_BANK[2:0] = address[29:27]
15	8	DRAM_BANK[2:0] = address[30:28]

Table 36-5. Mapping of address to DRAM column address

16-Bit Mode	DRAM Column Address
1	DRAM_COLUMN = {address[12:3], 2'b0}

Table 36-6. Mapping of address to DRAM row address

DRAM_ROW_SELECT[2:0]	DRAM row address
0	DRAM_ROW[15:0] = address[25:10]
1	DRAM_ROW[15:0] = address[26:11]
2	DRAM_ROW[15:0] = address[27:12]
3	DRAM_ROW[15:0] = address[28:13]
4	DRAM_ROW[15:0] = address[29:14]
5	DRAM_ROW[15:0] = address[30:15]
6	DRAM_ROW[14:0] = address[30:16]
7	DRAM_ROW[13:0] = address[30:17]

36.3.2.2 Timing Configuration

36.3.2.2.1 DDR Time Configuration Register 0 (DDR_TIME_CONFIG0)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	DRAM_REFRESH_TIME													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DRAM_COMMAND_TIME								DRAM_BANK_PRE_TIME							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-4. DDR Time Configuration Register 0 (DDR_TIME_CONFIG0)

Table 36-7. DDR_TIME_CONFIG0 field descriptions

Field	Description
DRAM_REFRESH_TIME	Refresh interval of the DRAM. Program in this register the number of System Bus Clock cycles between any two refresh requests. This register should contain the maximum number of System Bus Clock cycles between two refresh requests. The average time in System Bus Clock cycles between two refreshes to the DRAM is this number.
DRAM_COMMAND_TIME	Time-out after sending a command to the DRAM in bypass mode. For command sent to the DRAM using the DDR_COMMAND and DDR_COMPACT_COMMAND register, the normal checking of the timing parameters is not done. Instead, any new command to the DRAM is disabled for DRAM_COMMAND_TIME DRAM clock periods. This parameter needs to be programmed for the worst-case time-out.
DRAM_BANK_PRE_TIME	Time-out. Any active bank that has no outstanding requests is automatically precharged by the DRAM controller after this time-out has elapsed since the last access to the bank. This time can be set short, which results in open banks being precharged quite fast to long, which results in open banks left open for a long time. The value is a time count in DRAM clock periods.

36.3.2.2.2 DDR Time Configuration Register 1 (DDR_TIME_CONFIG1)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DRAM_TIME_RFC						DRAM_TIME_WR1					DRAM_TIME_WTR1				DRAM_TIME_RRD[5]
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DRAM_TIME_RRD[4:0]						DRAM_TIME_RC					DRAM_TIME_RAS				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-5. DDR Time Configuration Register 1 (DDR_TIME_CONFIG1)

36.3.2.2.3 DDR Time Configuration Register 2 (DDR_TIME_CONFIG2)

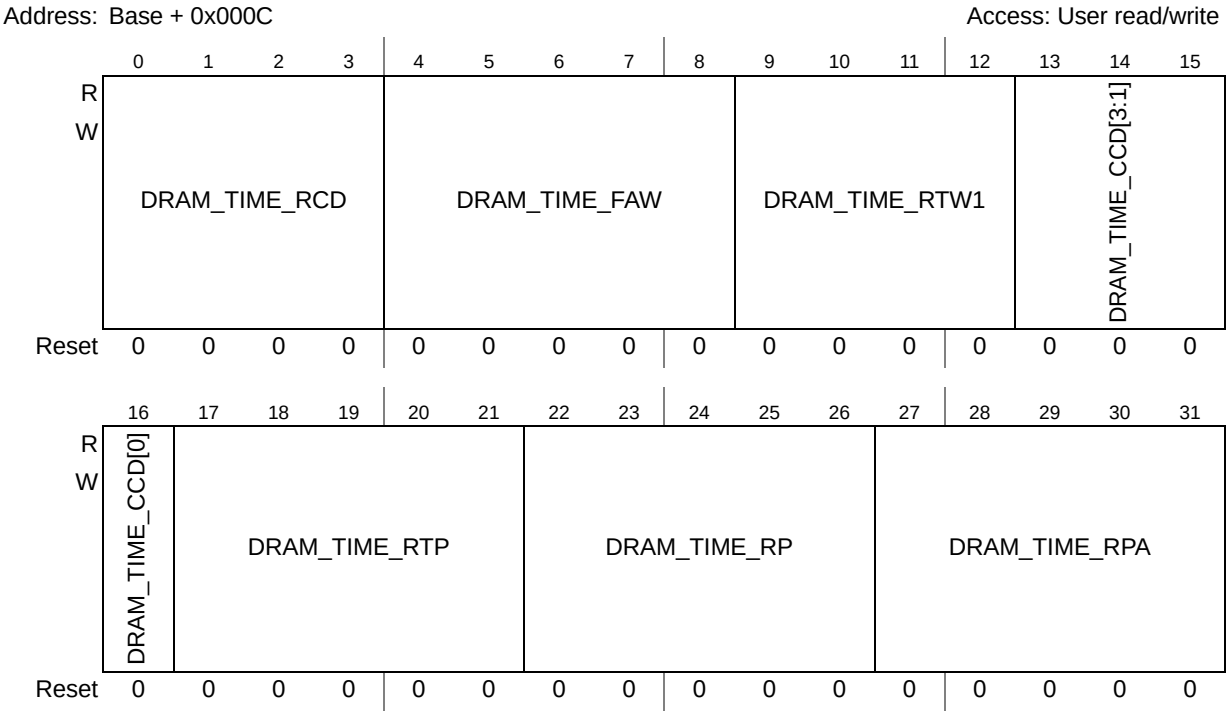


Figure 36-6. DDR Time Configuration Register 2 (DDR_TIME_CONFIG2)

The DDR_TIME_CONFIG1 and DDR_TIME_CONFIG2 registers need to be programmed with the DDR1/DDR2 timing parameters. All times are given in clock cycles.

The timing parameters are selected so the controller System Bus Clock cycles match with the JEDEC DDR2 specification. To interface with DDR1 or Mobile-DDR (LPDDR), some timing parameters need not be enforced, or are calculated differently. See the DRAM datasheet to determine their value. The timing parameters need to be programmed in function of this DRAM requirement. Table 36-8 gives the details.

Table 36-8. Timing parameters

Timing parameter	Controls JEDEC parameter (JEDEC spec)	Formulae (All times in system bus clock cycles)	Description
DRAM_TIME_RFC	t_{RFC}	DRAM_TIME_RFC = t_{RFC}	REFRESH to ACTIVE or REFRESH to REFRESH command interval.
DRAM_TIME_RRD	t_{RRD}	DRAM_TIME_RRD = t_{RRD}	ACTIVE bank A to ACTIVE bank B command.
DRAM_TIME_RC	t_{RC}	DRAM_TIME_RC = t_{RC}	ACTIVE to ACTIVE (same bank) command.
DRAM_TIME_RAS	t_{RAS}	DRAM_TIME_RAS = t_{RAS}	ACTIVE to PRECHARGE command.
DRAM_TIME_RCD	t_{RCD}	DRAM_TIME_RCD = t_{RCD}	ACTIVE to READ or WRITE delay.

Table 36-8. Timing parameters (continued)

Timing parameter	Controls JEDEC parameter (JEDEC spec)	Formulae (All times in system bus clock cycles)	Description
DRAM_TIME_FAW	t_{FAW}	DRAM_TIME_FAW[1] = t_{FAW}	4-bank activate period.
DRAM_TIME_CCD	t_{CCD}	DRAM_TIME_CCD[2] = max(t_{CCD},2)	CAS to CAS delay Because time is needed for data to be sent over, this time is minimum two clocks.
DRAM_TIME_RTP	t_{RTP}	DRAM_TIME_RTP[3] = t_{RTP}	Read to precharge delay. DRAM_TIME_RTP is the read-to-precharge delay and t_{RTP} is the *internal* read-to-precharge delay. Figure 36-7 gives the details.
DRAM_TIME_RP	t_{RP}	DRAM_TIME_RP = t_{RP}	Precharge command period.
DRAM_TIME_RPA	t_{RP}	DRAM_TIME_RPA[4] = t_{RP} + 1 (8-bank device) DRAM_TIME_RPA = t_{RP} (4-bank device)	Precharge all command period.
DRAM_TIME_WR1	t_{WR}	DRAM_TIME_WR1 = WL + t_{WR} + 2	DRAM_TIME_WR1 is the write recovery time, measured in clocks between write command and precharge command. For this reason, WL (the write latency) and the length of the actual write (2) need to be added to t_{WR}. Figure 36-8 gives the details.
DRAM_TIME_WTR1	t_{WTR}	DRAM_TIME_WTR1 = WL + t_{WTR} + 2	DRAM_TIME_WTR1 is the write to read time, measured in clocks between write command and read command. For this reason, WL (the write latency) and the length of the actual write (2) need to be added to t_{WTR}. Figure 36-9 gives the details.

Table 36-8. Timing parameters (continued)

Timing parameter	Controls JEDEC parameter (JEDEC spec)	Formulae (All times in system bus clock cycles)	Description
DRAM_TIME_RTW1	—	DRAM_TIME_RTW1 = CL - WL + 2 + t_{BTA}	DRAM_TIME_RTW1 is the read-to-write time, measured in clocks between the read and write command. There is no limitation on the DRAM on how to set this parameter. The parameter should be set such that there is no contention on the DQ data bus when switching from read to write. Equation given at left tries to come up with a formulae that defines the minimum value of DRAM_TIME_RTW1 to avoid contention. CL is the CAS latency, WL is the write latency, and t_{BTA} is the bus turn-around time. t_{BTA} is the minimum dead time that needs to be put on the bus between the host driving the bus and the DRAM driving the bus to take into account the transit delay on the PCB, the pad delay, the DRAM skew, and the on-chip delay.
CCD	t_{CCD}	CCD[5] = max (t_{CCD},2)	CAS to CAS delay. Because time is needed for data to be sent over, this time is minimum two clocks in 32-bit mode and four clocks in 16-bit mode.
RTP	t_{RTP}	RTP[6] = max(tRTP,2) (DDR2)	Read to precharge delay. RTP is the read-to-precharge delay and t_{RTP} is the internal read-to-precharge delay, hence, the difference for 16-bit mode. Figure 13-7 gives the details.

1 For DRAMs that do not need this check, set equal to 4 × t_{RRD}.

2 For DDR1 and Mobile-DDR t_{CCD} is 2.

3 For DDR1 and Mobile-DDR mode, t_{RTP} is not explicitly given. It is equal to 2.

4 This timing parameter controls precharge all command period duration. The equations shown are the JEDEC definition of the t_{RPA}. Some DRAM vendors do not follow JEDEC on this, and list t_{RPA} directly. In this case, set DRAM_TIME_RPA = t_{RPA}.

5 For DDR1 and Mobile-DDR, t_{CCD} is 2. For SDR, t_{CCD} is 4.

6 t_{RTP} is not explicitly given in SDR, DDR1 and Mobile-DDR mode. For DDR1 and Mobile-DDR, it is equal to 2. For SDR, it is equal to 4.

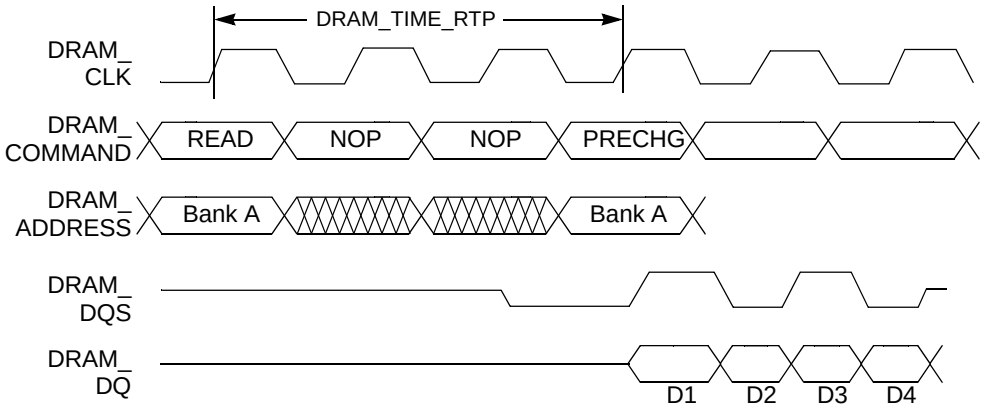


Figure 36-7. Read to precharge timing diagram

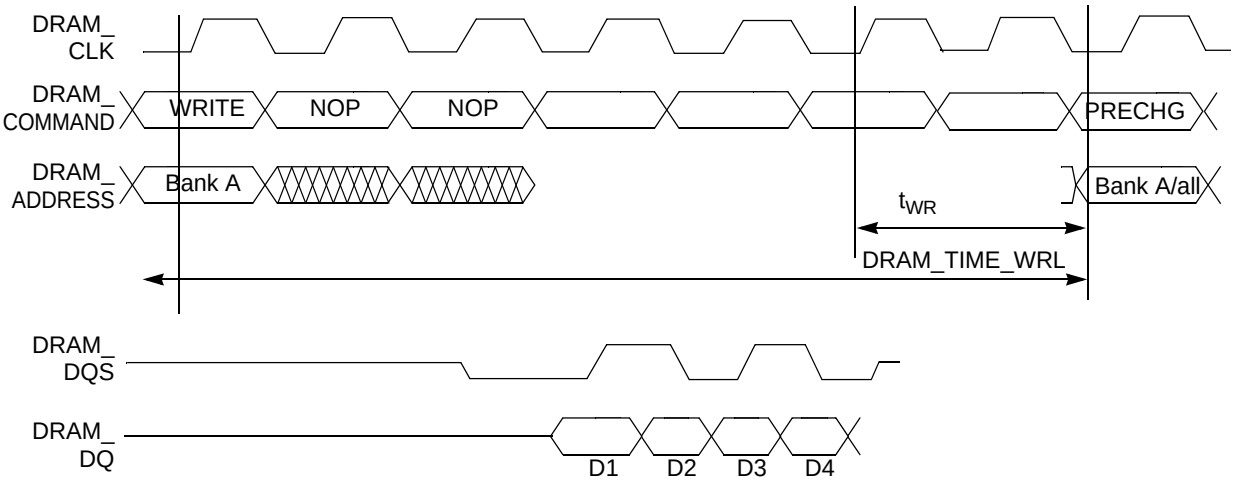


Figure 36-8. Write to precharge timing diagram

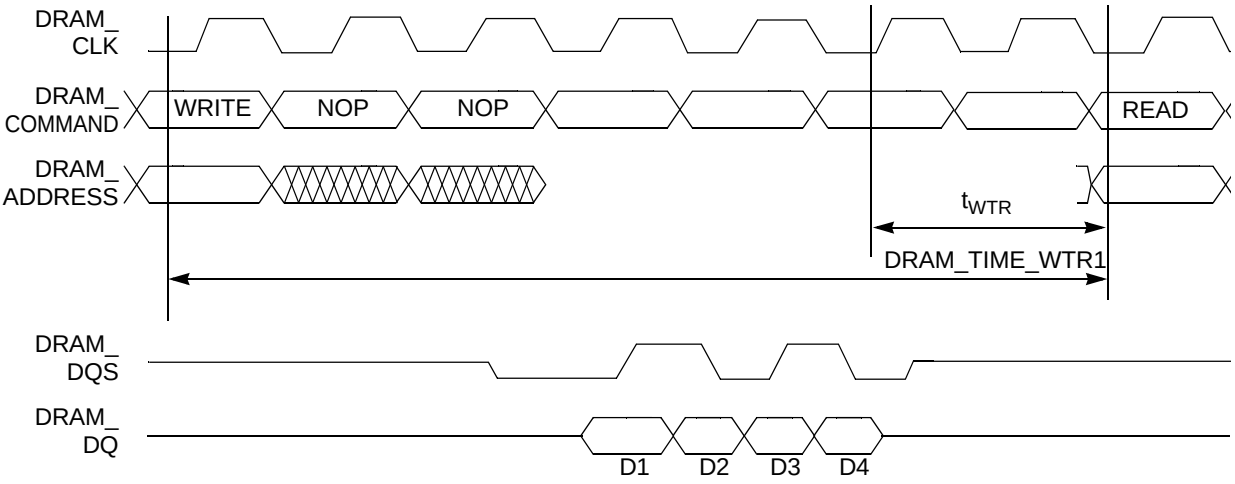


Figure 36-9. Write to read timing diagram

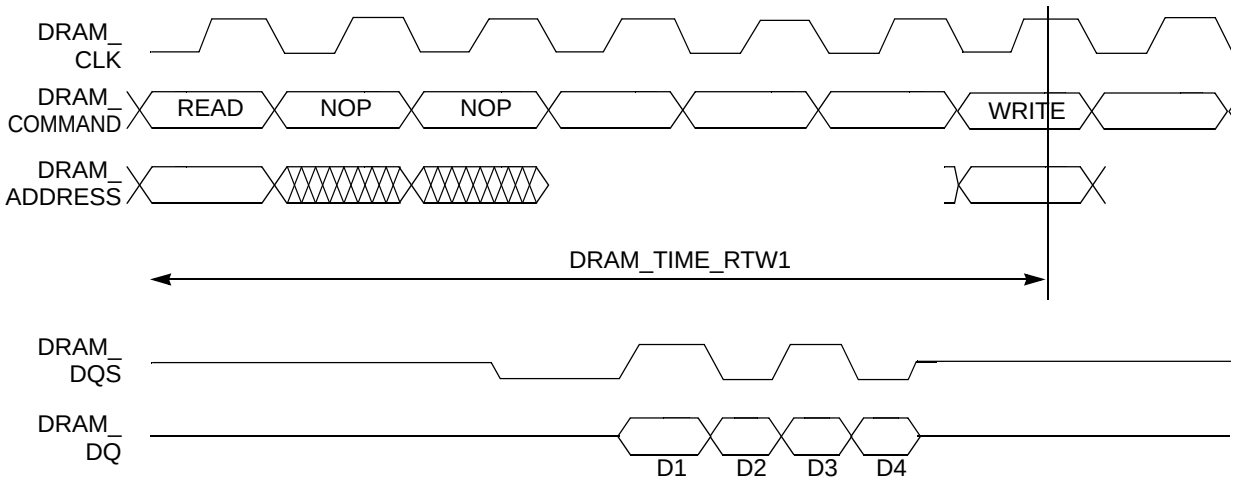


Figure 36-10. Read to write timing diagram

36.3.2.3 DRAM Command Register (DDR_COMMAND)

The DDR_COMMAND register gives the option to send commands directly to the DRAM. This register only operates when the command mode bit (CMD MODE, bit 3) is set in the DDR_SYS_CONFIG register.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W									DRAM_COMMAND[23:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	DRAM_COMMAND[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-11. DRAM Command Register (DDR_COMMAND)

Table 36-9. DDR_COMMAND field descriptions

Field	Description
DRAM_COMMAND	When the DDR_SYS_CONFIG[CMD MODE] bit is set, the value written to bits [23:0] of this register is output on the DRAM address group with following mapping: • DRAM_ADDRESS[14:0] = DRAM_COMMAND[14:0] • DRAM_ADDRESS[15] = 0 • If (DRAM_COMMAND[15] == 1) turn off CKE DRAM attribute bit[1] • DRAM_BA[2:0] = DRAM_COMMAND[18:16] • DRAM_WEB = DRAM_COMMAND[19] • DRAM_CAS = DRAM_COMMAND[20] • DRAM_RAS = DRAM_COMMAND[21] • DRAM_CS0 = DRAM_COMMAND[22] • DRAM_CS1 = DRAM_COMMAND[23] **Note:** The intended use of the command interface is to initialize the DRAM and to put the DRAM into or out of the self-refresh and power-down modes.

1 CKE is turned off on the same clock cycle as when the requested command is being sent to the DRAM.

36.3.2.4 Compact Command Register (DDR_COMPACT_COMMAND)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	DRAM_COMPACT_COMMAND															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-12. Compact Command Register (DDR_COMPACT_COMMAND)

Table 36-10. DDR_COMPACT_COMMAND field descriptions

Field	Description
DRAM_COMPACT_COMMAND	The compact command register gives the option to sent commands to the DRAM using 16-bit writes. See Table 36-11.

Table 36-11. DDR_COMPACT_COMMAND register options

COMPACT_COMMAND[15:14]	Description
00	Write DRAM attributes and wait (wait time = wait time until next command) Wait is executed after writing attributes. • CKE = COMPACT_COMMAND[13] • Self Ref En = COMPACT_COMMAND[12] • CLK ON = COMPACT_COMMAND[11] • CMD MODE = COMPACT_COMMAND[10] • If (COMPACT_COMMAND[7] == 1'b1) — Wait time = (COMPACT_COMMAND[6:0] × 512) DRAM clock periods OR — Wait time = (COMPACT_COMMAND[6:0] × 32) DRAM clock periods
01	DRAM command • DRAM_CS = COMPACT_COMMAND[12] • DRAM_RAS = COMPACT_COMMAND[11] • DRAM_CAS = COMPACT_COMMAND[10] • DRAM_WEB = COMPACT_COMMAND[9] • DRAM_BA[2:0] = COMPACT_COMMAND[8:6] • DRAM_ADDRESS[10] = COMPACT_COMMAND[5] • If (COMPACT_COMMAND[4] == 1'b1) turn off CKE DRAM attribute bit[1]
1x	DRAM set mode registers • DRAM_CS = 0 • DRAM_RAS = 0 • DRAM_CAS = 0 • DRAM_WEB = 0 • DRAM_ADDRESS[13] = 0 • DRAM_BA[2] = 0 • DRAM_ADDRESS[12:0] = COMPACT_COMMAND[12:0] • DRAM_ADDRESS[14:13] = COMPACT_COMMAND[14:13]

1 CKE is turned off the clock cycle when sending the requested command to the DRAM.

The DDR_COMPACT_COMMAND register's main purpose is to be written during enter/exit of self-refresh (the auto-sequencer).

The DDR_COMPACT_COMMAND register allows three types of actions to be executed:

- Write DRAM attributes and wait. Wait is executed after updating the DRAM attributes. It is possible to update the CKE bit, the self-refresh enable, the CLK configuration (on/off), and the CMD mode setting.

  If, during the time the wait is executed, another command is written to DDR_COMPACT_COMMAND, this write is delayed until the wait is over.

  During this time, the peripheral bus and all buses connected to it block and are not able to process any other read or write.
- Write a command to DRAM without controlling the address. In this mode, it is possible to send refresh, activate, and precharge commands to the DRAM.
- Write a DRAM mode register.

36.3.2.5 Enter/Exit Self-Refresh Registers

Address: Base + 0x0018 (SELF_REFRESH_CMD_0) Base + 0x0028 (SELF_REFRESH_CMD_4)
Base + 0x001C (SELF_REFRESH_CMD_1) Base + 0x002C (SELF_REFRESH_CMD_5)
Base + 0x0020 (SELF_REFRESH_CMD_2) Base + 0x0030 (SELF_REFRESH_CMD_6)
Base + 0x0024 (SELF_REFRESH_CMD_3) Base + 0x0034 (SELF_REFRESH_CMD_7) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SELF_REFRESH_CMD*n*[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-13. Self-Refresh Command *n* Register

The self-refresh command registers contain the commands sent to the DRAM when a self-refresh request is given. Figure 36-14 gives the details.

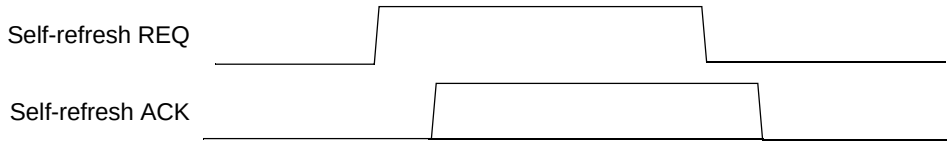


Figure 36-14. Enter/exit self-refresh command protocol

When the DRAM controller sees a low-to-high transition on the incoming self-refresh REQ signal coming from the PMC controller, its reaction depends on the state of the internal self-refresh EN command bit.

- If the self-refresh EN bit is set, the DRAM controller writes self-refresh CMD[0:3] registers, starting with reg 0 and ending with reg 3, to the compact command register. After the last register has been written and its wait time has expired (if any), it pulls high the self-refresh ACK signal to the PMC to acknowledge entry to self-refresh mode.
- If the self-refresh EN bit is clear, the DRAM controller does not react to the request and keeps self-refresh ACK signal low.

When the DRAM controller sees a high-to-low transition on the incoming self-refresh REQ signal coming from the PMC controller, its reaction is similar and depends on the state of the internal self-refresh EN command bit again.

- If the self-refresh EN bit is set, the DRAM controller writes self-refresh CMD[4:7] registers, starting with register CMD4 and ending with register CMD7, to the compact command register. After the last register has been written and its wait time has expired (if any), it pulls low the self-refresh ACK signal to the PMC to acknowledge exit from self-refresh mode.
- If the self-refresh EN bit is clear, the DRAM controller does not react to the request and keeps self-refresh ACK signal high.

36.3.2.6 DQS Config Offset Count (DQS_CONFIG_OFFSET_COUNT) Register

Address: Base + 0x0038 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	DQS SLAVE 3 OFFSET COUNT							0	DQS SLAVE 2 OFFSET COUNT						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	DQS SLAVE 1 OFFSET COUNT							0	DQS SLAVE 0 OFFSET COUNT						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-15. DQS Config Offset Count (DQS_CONFIG_OFFSET_COUNT) Register

Table 36-12. DQS_CONFIG_OFFSET_COUNT field descriptions

Field	Description
DQS_SLAVE_[6:0]_ OFFSET_COUNT	There is a separate field for each DQS input to the controller. These fields code for an offset counted in elemental gate delay increments applied to each DQS slave. The number is a two-complement number that can be positive and negative. This register can be used to compensate systematic delay shift in the DRAM controller due to processing. Leave this register all-zero, unless Freescale issues a report giving a different value.

36.3.2.7 DQS Config Offset Time (DQS_CONFIG_OFFSET_TIME) Register

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	DQS SLAVE 3 OFFSET TIME[5:0]						0	0	DQS SLAVE 2 OFFSET TIME[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	DQS SLAVE 1 OFFSET TIME[5:0]						0	0	DQS SLAVE 0 OFFSET TIME[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-16. DQS Config Offset Time (DQS_CONFIG_OFFSET_TIME) Register

Table 36-13. DQS_CONFIG_OFFSET_TIME field descriptions

Field	Description
DQS_SLAVE_[5:0] _ OFFSET_TIME	There is a separate field for each DQS input to the controller. These fields code for an offset counted in time units. This register can be used to advance or delay the read strobe. Negative values advance the read strobe, positive values retard the read strobe. Time delay coded = [field value (2-complement)] × TDRAM-clock/256. The applied offset range for a 200 MHz clock is approximately ± 290 ps.

36.3.2.8 DQS Delay Status Register (DQS_DELAY_STATUS)

Address: Base + 0x0040 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	DQS MASTER COUNT 2											
W																
Reset[1]	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	DQS MASTER COUNT 1											
W																
Reset	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—

Figure 36-17. DQS Delay Status Register (DQS_DELAY_STATUS)

[1] The reset value of the DQS Delay status register changes over time by internal DQS delay line counting.

Table 36-14. DQS_DELAY_STATUS field descriptions

Field	Description
DQS MASTER COUNT 2	Delay count output by the controller for the first DQS master to code for 1/4 System Bus Clock cycle delay.
DQS MASTER COUNT 1[1]	Delay count output by the controller for the second DQS master to code for 1/4 System Bus Clock cycle delay.

[1] Reset value of these fields are unspecified because after reset they change with the DQS delay line regulates to the quarter cycle delay.

36.3.2.9 DDR Auxiliary Configuration Register (DDR_AUX_CONFIG)

Address: Base + 0x0044 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDATA INJECT	LOOPBACK	RDATA COMPARE	DQS TIMING		DRIVE DQS	INPUTS ALWAYS ON	DQS LOW	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-18. DDR Auxiliary Configuration Register (DDR_AUX_CONFIG)

Table 36-15. DDR_AUX_CONFIG field descriptions

Field	Description
WDATA INJECT	0 Normal operation. 1 Write data from incoming buses is replaced with data from self_test_data_out register.
LOOPBACK	0 Normal operation. 1 loopback mode. Reads are not done from D-RAM, but write data path is activated to insert data.
RDATA COMPARE	0 Normal operation. 1 Incoming read data is compared with self_test_data_expect register, and if one or more bits do not match, the corresponding bits in the read_data_error register are set.
DQS TIMING	00 Normal DQS timing. 01 DQS is 1/4 clock cycle early. 11 DQS is 1/2 clock cycle early. 1*x* Reserved.
DRIVE DQS	0 Normal operation. 1 DQ, DM, DQS driven low-impedance 1/4 clock cycle early.
INPUTS ALWAYS ON	0 Normal operation. 1 DQ, DQS inputs always enabled.
DQS LOW	0 Normal operation. 1 Drive DQS low by default.

36.3.2.10 DDR Extra Attributes Register

Address: Base + 0x0060 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	CONFIG_SDR	CONFIG_CAS3	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-19. DDR Extra Attributes Register

Table 36-16. DDR Extra Attributes field descriptions

Field	Description
CONFIG_SDR	0 DDR mode. 1 SDR mode.
CONFIG_CAS3	0 SDR timing with CAS latency 2. 1 SDR timing with CAS latency 3.

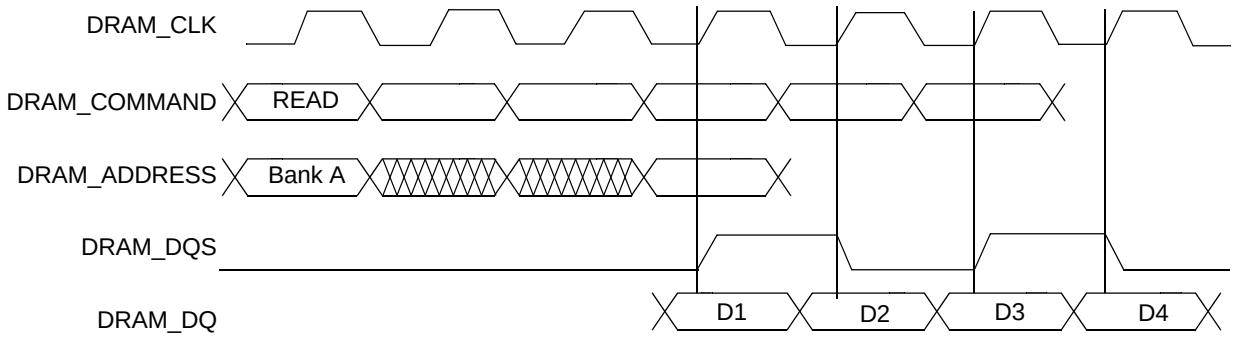


Figure 36-20. SDR read timing diagram

During SDR mode, DQS is driven by the device all the time. DQS is driven out by the controller synchronous with DRAM_CLK the specified CAS delay after the read command was issued. Read data is input relative to the DQS, exactly the same way as in DDR mode. This means that the strobe point is 1/4 clock after the edges of the DQS.

It is possible to shift the sample point to right after the clock edges of DQS by writing 0x2020_2020 to the DQS Config Offset Time register.

NOTE

In SDR mode, t_{DQSEN} is required like in other DDR/DDR2 mode. Hence, RDLY should be configured properly depending on the CAS latency (CONFIG_CAS3 value), the distance between SDRAM and the device, etc. Different from RDLY, WDLY must be set with 0x1 for correct write operation.

36.4 Functional description

The DRAM controller is a multi-port DRAM controller. It listens to incoming requests on multiple buses and decides on each rising clock edge what command needs to be sent to the DRAM.

A block diagram is given in Figure 36-1. The major blocks of the DRAM controller are described in the following sections.

36.4.1 Interfacing with the DRAM

36.4.1.1 Connecting the DRAM

- 16-bit DRAM systems need to be connected to all DQ, DM, DQS lines.
- Row/column address pins need to be connected starting with bit [0] and ending with the highest order DRAM bit. Leave MSB's unconnected if the DRAM has fewer address pins than the controller.
- DRAM bank address pins need to be connected starting with bit [0] and ending with the highest order bank address bit. Leave MSB unconnected if the DRAM has fewer bank address pins than the controller.

36.4.2 Programming DRAM Device Internal Configuration Register

- Set burst type to sequential.
- Burst length is always 8-byte. Means 4-beat bursts in a 16-bit system.
- Set CAS latency to lowest value DRAM can tolerate at intended speed, and then set write latency and read latency accordingly.
- Set posted CAS additive latency to 0.
- Controller never uses auto-precharge on read or write.
- Configure DQS operation for single-ended operation.
- RTT and output drive strength configuration depends on electrical characteristics.

36.4.3 DRAM command engine

This block decides what command to send to the DRAM controller next. Four different commands can be sent to the DRAM to service incoming requests from the three incoming buses. These commands are:

- Precharge
- Activate
- Read
- Write

On every rising clock edge, the DRAM command engine first determines, using parallel logic, what is highest priority pending precharge, activate, read and write command. Next, it decides which of these commands to send to the DRAM.

The arbiters that make the decisions about what command to send next to the DRAM are aware of the current state the DRAM is in. When arbitrating a command on the DRAM bus, the following information is processed:

- For each bank, if it is precharged or not
- For each incoming request, if it hits in an already active bank or not
- For each bank, if the DRAM currently can accept a precharge command to it
- For each bank, if the DRAM currently can accept an activate command to it

- For each bank, if the DRAM currently can accept a read command to it
- For each bank, if the DRAM currently can accept a write command to it

The logic keeping track of what is currently possible on each of the banks is not in the DRAM command engine. It is part of the timing manager, whose task is to signal to the DRAM command engine that commands are currently possible.

36.4.4 Write buffer

All incoming writes are sent first to the write buffer, part of the command engine. Writes are sent to the DRAM in background, whenever possible. The DRAM tries to postpone the writes until there are no further outstanding read requests. However, when the write buffer is full, or when there is a new request for an address already inside the write buffer, the DRAM controller writes the content of the write buffer to the DRAM.

36.4.5 Timing manager

The timing manager consists of a bank of counters. These counters keeps track of all DRAM timing parameters and signals to the DRAM command engine when a precharge, activate, read or write command is possible. This information is supplied to the DRAM command engine for each bank separately.

All timing parameters are programmable in software.

36.4.6 DRAM read block and DRAM write block

Sending a read or write command to the DRAM is a two-step process. First, the command is sent, which is done by the command engine. After some clock cycles, the data must follow.

Manipulating the read data is done by the read block. For every read command sent to the DRAM, the command engine informs the read block. Upon receiving the read command, the read block delays this to account for DRAM pipelining. Then, it receives the correct amount of data from the DRAM DQ inputs and forwards this data to the correct bus.

Manipulating the write data is done by the write block. It works the same way as the read block. The command engine informs the write block of a pending write. Upon receiving the command, the write block delays this to account for DRAM pipelining. Then, it receives the relevant data from the write buffer and transmits this to the DRAM.

36.4.7 Bus interface

The bus interface accepts a slave peripheral bus. The bus interface fulfills several functions:

- It contains all configuration registers.
- It contains logic to send an error interrupt to the processor. The error interrupt is active when the FIFO overflow or FIFO underflow error condition and corresponding interrupt enable in register DDR_SYS_CONFIG is set. The register summary is given in Table 36-1.

The FIFO overflow and underflow flags are tied to a FIFO that keeps track of the number of DQS strobes the DRAM is expected to produce. If a read command is sent to the DRAM, the DRAM is expected to answer after producing the read data on its DQ outputs, with some edges on its DQS output used by the controller to clock the read data. If the DRAM controller produces the read strobes at an incorrect time, or produces not enough or too many read strobes, the DRAM controller may detect some error conditions because they result in an overflow or underflow of the FIFO that keeps track of the number of outstanding DQS pulses. These bits do not detect timing configuration errors. Underflows and overflows signaled by the read FIFO point to following possible error sources:

- Incorrect configuration of the DRAM. Burst length set incorrectly
- Incorrect configuration of the DRAM controller.
  - Incorrect RDLY
  - Incorrect HALF_DQS_DLY
  - Incorrect QUART_DQS_DLY
  - Incorrect DRAM timing parameters or mis-match between various settings.
- Problems with the electrical connections between the DRAM controller and the DRAM

- It contains a bypass path to send commands to the DRAM. This is because the DRAM controller contains no logic to take care of DRAM initialization, programming the mode registers, or putting the DRAM into or out of the sleep and standby modes like self-refresh. Essentially, these functions are made available over the peripheral bus. To program the mode registers, the DRAM controller needs to be put in a bypass mode, where incoming requests are not serviced. In this bypass mode, commands are sent from the peripheral interface directly to the DRAM to program the mode registers or to put the DRAM into or out of sleep mode.
- During bypass mode, all reads and writes are blocked. Refresh keeps running, but can be separately disabled.

Chapter 37
Multi-Port DRAM Controller Priority Manager (PRIOMAN)

37.1 Introduction

The multi-port DRAM controller priority manager (PRIOMAN) is a submodule of the multi-port DDR DRAM controller (MDDRC). The multi-port DRAM controller services the highest priority request from three different buses using a 4-bit priority signal. This 4-bit priority is dynamically set by the DRAM controller priority manager based on register settings and the most recent activity on each bus. In general, the DRAM priority manager increases the priority of a channel if it has not been recently serviced and decreases the priority of channels that have been recently serviced.

A block diagram of the priority manager is given in Figure 37-1. It accepts the request and ACK-signals for all three DRAM buses, and produces the priority signals for the three buses.

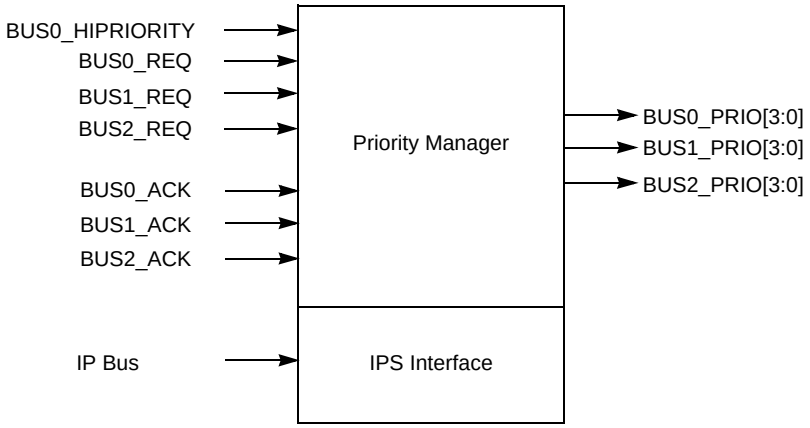


Figure 37-1. Priority Manager Block Diagram

The priority manager uses an ACK-based schema; the priority depends on how many times for the last N requests accepted by the DRAM controller the current owner of the bus won the request. A running average counter keeps track of how many acknowledgements out of the last N acknowledgements have been for a specific bus. This number is then put in a look-up table configurable by writing some configuration registers. The output of the look-up table is the priority for the next request on this bus.

This priority schema is versatile because programming the look-up table allows controlling relative priority to other channels and the average share of the bandwidth the current master gets. The priority schema introduces fairness because the look-up table can be programmed to reduce the priority of a bus that has won a large share of requests and increase the priority of a bus that has lost a large share of requests.

37.1.1 Features

- Dynamic priority calculation based on ACKing history
  - — Fully programmable using look-up table
    - – Can be configured for high or low latency and high or low bandwidth

  - Separate control over average latency and average bandwidth
  - Versatile so it can mimic the CSB arbitration schema
- Fairness guaranteed by reducing priority of channels that receive a lot of grants, and increasing priority of channels that are denied the bus often
- Repeat transfer built into the DRAM controller. Priority manager can set the maximum repeat count by controlling when lowest priority occurs.

- Bus 0 high priority request input to enable real time priority escalation.

37.1.2 Debug mode

When the MCU is in debug mode, the PRIOMAN behavior is unaffected and remains identical to its operation in normal mode.

37.2 Bus connections

The following masters are connected to three buses.

- Bus 0: PDI, FEC, DMA_0, DMA_1
- Bus 1: Power architecture e200 Core_0
- Bus 2: Power architecture e200 Core_1

37.3 Memory map and register description

37.3.1 Memory map

The PRIOMAN module is mapped as a submodule within the MDDRC module.

Table 37-1. PRIOMAN memory map

Offset from MDDRC_BASE (0xC3F9_8000)	Register	Access[1]	Reset Value[2, 3]	Section/Page
0x0000–0x007F	MDDRC registers—see Chapter 36, Multi-Port DDR DRAM Controller (MDDRC)			
0x0080	Priority Manager Configuration 1 Register (PRIOMAN_CONFIG1)	R/W	0x0007_7777	on page 1376
0x0084	Priority Manager Configuration 2 Register (PRIOMAN_CONFIG2)	R/W	0x0000_00U1	on page 1377
0x0088	HIPRIO_CONFIG register	R/W	0x0000_UUUU	on page 1378
0x008C	LUT table 0 main upper register	R/W	0x1111_1222	on page 1379
0x0090	LUT table 1 main upper register	R/W	0x1111_1222	on page 1379
0x0094	LUT table 2 main upper register	R/W	0x1111_1222	on page 1379
0x0098–0x009F	Reserved			
0x00A0	LUT table 0 main lower register	R/W	0x2334_567A	on page 1379

Table 37-1. PRIOMAN memory map (continued)

Offset from MDDRC_BASE (0xC3F9_8000)	Register	Access[1]	Reset Value[2, 3]	Section/Page
0x00A4	LUT table 1 main lower register	R/W	0x2334_567A	on page 1379
0x00A8	LUT table 2 main lower register	R/W	0x2334_567A	on page 1379
0x00AC–0x00B3	Reserved			
0x00B4	LUT table 0 alternate upper register	R/W	0x0000_0000	on page 1380
0x00B8	LUT table 1 alternate upper register	R/W	0x0000_0000	on page 1380
0x00BC	LUT table 2 alternate upper register	R/W	0x0000_0000	on page 1380
0x00C0–0x00C7	Reserved			
0x00C8	LUT table 0 alternate lower register	R/W	0x0000_0000	on page 1381
0x00CC	LUT table 1 alternate lower register	R/W	0x0000_0000	on page 1381
0x00D0	LUT table 2 alternate lower register	R/W	0x0000_0000	on page 1381
0x00D4–0x00DB	Reserved			
0x00DC	Performance monitor configuration register (PERMON_CONFIG)	R/W	0x0800_0000	on page 1383
0x00E0	Event time counter register	R/W	0x0000_0000	on page 1384
0x00E4	Event time preset register	R/W	0x0000_0000	on page 1384
0x00E8	Performance Monitor 1 Address Low (PERF_MNTR1_ADDR_LOW) Register	R/W	0x0000_0000	on page 1385
0x00EC	Performance Monitor 2 Address Low (PERF_MNTR2_ADDR_LOW) Register	R/W	0x0000_0000	on page 1385
0x00F0	Performance Monitor 1 Address High (PERF_MNTR1_ADDR_HI) Register	R/W	0x0000_0000	on page 1385
0x00F4	Performance Monitor 2 Address High (PERF_MNTR2_ADDR_HI) Register	R/W	0x0000_0000	on page 1385
0x0100	Performance Monitor 1 Read Counter (PERF_MNTR1_READ_CNTR)	R	0x0000_0000	on page 1386
0x0104	Performance Monitor 2 Read Counter (PERF_MNTR2_READ_CNTR)	R	0x0000_0000	on page 1386
0x0108	Performance Monitor 1 Write Counter (PERF_MNTR1_WRITE_CNTR)	R	0x0000_0000	on page 1386
0x010C	Performance Monitor 2 Write Counter (PERF_MNTR2_WRITE_CNTR)	R	0x0000_0000	on page 1386
0x0110	Granted ack counter 0 register	R	0x0000_0000	on page 1387
0x0114	Granted ack counter 1 register	R	0x0000_0000	on page 1387
0x0118	Granted ack counter 2 register	R	0x0000_0000	on page 1387
0x011C–0x0123	Reserved			

Table 37-1. PRIOMAN memory map (continued)

Offset from MDDRC_BASE (0xC3F9_8000)	Register	Access[1]	Reset Value[2, 3]	Section/Page
0x0124	Cumulative wait counter 0 register	R	0x0000_0000	on page 1387
0x0128	Cumulative wait counter 1 register	R	0x0000_0000	on page 1387
0x012C	Cumulative wait counter 2 register	R	0x0000_0000	on page 1387
0x0130–0x0137	Reserved			
0x0138	Summed priority counter 0 register	R	0x0000_0000	on page 1388
0x013C	Summed priority counter 1 register	R	0x0000_0000	on page 1388
0x0140	Summed priority counter 2 register	R	0x0000_0000	on page 1388
0x0144–0x03FF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

37.3.2 Register descriptions

37.3.2.1 Priority Manager Configuration 1 Register (PRIOMAN_CONFIG1)

Address: Base + 0x0080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	LUT SEL2[1:0]		LUT SEL1[1:0]		LUT SEL0[1:0]		0	0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0			ACK_COUNT2[3:0]				ACK_COUNT1[3:0]				ACK_COUNT0[3:0]			
W																
Reset	0	0	0	0	0	1	1	1	0	1	1	1	0	1	1	1

Figure 37-2. Priority Manager Configuration 1 Register (PRIOMAN_CONFIG1)

Table 37-2. PRIOMAN_CONFIG1 field descriptions

Field	Description
LUT SEL*n*	Lookup Table Select. Selects between primary and secondary look-up table configuration registers. 00 Select main look-up table configuration register. 01 Select alternate look-up table configuration register. 10 Select alternate look-up table configuration register if congested flag is set. See Section 37.3.2.3, High Priority Configuration Register (HIPRIO_CONFIG). 11 Select alternate look-up table configuration register if bus 0 incoming BUS0_HIPRIORITY signal is high.
ACK_COUNT*n* [3:0]	Configuration fields. One for every channel. Determines how many requests the number of ACKs for the self-channel is counted.[1] 0000 1 ACK is counted. 0001 2 ACKs are counted. 0010 3 ACKs are counted. 0011 4 ACKs are counted. 0100 6 ACKs are counted. 0101 8 ACKs are counted. 0110 12 ACKs are counted. 0111 16 ACKs are counted. 1000 24 ACKs are counted. 1001 32 ACKs are counted. 1010 48 ACKs are counted. 1011 63 ACKs are counted.

[1] Look-up table input is the running average of the number of ACKs for the self channel counted over the grant total of the last N ACKs. ACK_COUNT[2:0] controls the parameter N.

37.3.2.2 Priority Manager Configuration 2 Register (PRIOMAN_CONFIG2)

Address: Base + 0x0084 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	CONGESTED	0	1	0	ACKSEL2	ACKSEL1	ACKSEL0
W																
Reset	0	0	0	0	0	0	0	0	0	—	0	1	0	0	0	1

Figure 37-3. Priority Manager Configuration 2 Register (PRIOMAN_CONFIG2)

Table 37-3. PRIOMAN_CONFIG2 field descriptions

Field	Description
CONGESTED	Read only 0 Congested flag is cleared. 1 Congested flag is set.

Table 37-3. PRIOMAN_CONFIG2 field descriptions (continued)

Field	Description
ACK SEL*n*	One bit is provided for each priority manager channel. They determine what happens if the current channel is not requesting. 0 No special overrule. Regulates default priority to high value. 1 If current channel is not requesting, every ACK for other channel is treated like an ACK for the current channel. Regulates default priority to low value.

37.3.2.3 High Priority Configuration Register (HIPRIO_CONFIG)

The High Priority Configuration Register (HIPRIO_CONFIG) controls the hiprio detection logic. The hiprio detection logic detects what percentage of the requests ACKed by the DRAM controller are ACKed with a priority larger than eight.

Address: Base + 0x0088 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SET_CONGEST_LEVEL[11:4]								CLEAR_CONGEST_LEVEL[11:4]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FILTER BANDWIDTH			AVERAGE_HIPRIORITY[12:0]												
W																
Reset	0	0	0	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 37-4. High Priority Configuration Register (HIPRIO_CONFIG)

Table 37-4. HIPRIO_CONFIG field descriptions

Field	Description
SET_CONGEST_LEVEL[11:4]	If(average_hipriority[12:4] > set_congest_level[12:4]) → set the congested flag.
CLEAR_CONGEST_LEVEL[11:4]	If(average_hipriority[12:4] < clear_congest_level[12:4]) → clear the congested flag.
AVERAGE_HIPRIORITY[12:0]	Average number of high priority requests to DRAM, coded between values 0x1000 and 0x0000 0x1000: 100% high-priority requests. 0x0000: 0% high-priority requests.
FILTER BANDWIDTH[2:0]	This setting controls the averaging time of the filter used for average_hipriority[12:0][1] 000 Time constant W0 = 8 ACKS, K = 0.125 001 Time constant W0 = 16 ACKS, K = 0.0625 010 Time constant W0 = 32 ACKS, K = 0.0312 011 Time constant W0 = 64 ACKS, K = 0.0156 100 Time constant W0 = 128 ACKS, K = 0.0078 101 Time constant W0 = 256 ACKS, K = 0.0039 110 Time constant W0 = 512 ACKS, K = 0.0020 111 Time constant W0 = 1024 ACKS, K = 0.0010

[1] Refer to Equation 37-1 and Equation 37-2 for the relationship between filter bandwidth and filter behavior.

37.3.2.4 Look-up Table *n* Main Upper Registers (LUT0–LUT2)

The Look-up Table *n* Main Upper (LUT0–LUT2) registers contain the upper eight entries of the look-up tables for channels 0–2, main table. All registers contain identical fields.

Address: Base + 0x008C (LUT0 main upper)
Base + 0x0090 (LUT1 main upper)
Base + 0x0094 (LUT2 main upper)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRIO15[3:0]				PRIO14[3:0]				PRIO13[3:0]				PRIO12[3:0]			
Reset	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRIO11[3:0]				PRIO10[3:0]				PRIO9[3:0]				PRIO8[3:0]			
Reset	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0

Figure 37-5. Look-up Table Main Upper Registers (LUT0–LUT2)

Table 37-5. LUT0–LUT2 Main Upper field descriptions

Field	Description
PRIO15[3:0]	Priority setting if 15 or more ACKs for own channel counted
PRIO14[3:0]	Priority setting if 14 ACKs for own channel counted
PRIO13[3:0]	Priority setting if 13 ACKs for own channel counted
PRIO12[3:0]	Priority setting if 12 ACKs for own channel counted
PRIO11[3:0]	Priority setting if 11 ACKs for own channel counted
PRIO10[3:0]	Priority setting if 10 ACKs for own channel counted
PRIO9[3:0]	Priority setting if 9 ACKs for own channel counted
PRIO8[3:0]	Priority setting if 8 ACKs for own channel counted

37.3.2.5 Look-up Table *n* Main Lower Registers (LUT0–LUT2)

The Look-Up Table *n* Main Lower (LUT0–LUT2) registers contain the lower eight entries of the look-up tables for channels 0–2, main table. All registers contain identical fields.

Address: Base + 0x00A0 (LUT0 main lower)
Base + 0x00A4 (LUT1 main lower)
Base + 0x00A8 (LUT2 main lower)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRIO7[3:0]				PRIO6[3:0]				PRIO5[3:0]				PRIO4[3:0]			
Reset	0	0	1	0	0	0	1	1	0	0	1	1	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRIO3[3:0]				PRIO2[3:0]				PRIO1[3:0]				PRIO0[3:0]			
Reset	0	1	0	1	0	1	1	0	0	1	1	1	1	0	1	0

Figure 37-6. Look-up Table Main Lower Registers (LUT0–LUT2)

Table 37-6. LUT0–LUT2 Main Lower field descriptions

Field	Description
PRIO7[3:0]	Priority setting if 7 ACKs for own channel counted
PRIO6[3:0]	Priority setting if 6 ACKs for own channel counted
PRIO5[3:0]	Priority setting if 5 ACKs for own channel counted
PRIO4[3:0]	Priority setting if 4 ACKs for own channel counted
PRIO3[3:0]	Priority setting if 3 ACKs for own channel counted
PRIO2[3:0]	Priority setting if 2 ACKs for own channel counted
PRIO1[3:0]	Priority setting if 1 ACK for own channel counted
PRIO0[3:0]	Priority setting if 0 ACKs for own channel counted

37.3.2.6 LUT0–LUT2 Alternate Upper Registers

These registers contain the upper eight entries of the look-up tables for channels 0–2, alternate table. All registers contain identical fields.

Address: Base + 0x00B4 (LUT0 alternate upper)
Base + 0x00B8 (LUT1 alternate upper)
Base + 0x00BC (LUT2 alternate upper)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRIO15[3:0]				PRIO14[3:0]				PRIO13[3:0]				PRIO12[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRIO11[3:0]				PRIO10[3:0]				PRIO9[3:0]				PRIO8[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-7. LUT0–LUT2 Alternate Upper Register

Table 37-7. LUT0–LUT2 Alternate Upper field descriptions

Field	Description
PRIO15[3:0]	Priority setting if 15 or more ACKs for own channel counted
PRIO14[3:0]	Priority setting if 14 ACKs for own channel counted
PRIO13[3:0]	Priority setting if 13 ACKs for own channel counted
PRIO12[3:0]	Priority setting if 12 ACKs for own channel counted
PRIO11[3:0]	Priority setting if 11 ACKs for own channel counted
PRIO10[3:0]	Priority setting if 10 ACKs for own channel counted
PRIO9[3:0]	Priority setting if 9 ACKs for own channel counted
PRIO8[3:0]	Priority setting if 8 ACKs for own channel counted

37.3.2.7 LUT0–LUT2 Alternate Lower

These registers contain the lower eight entries of the look-up tables for channels 0–2, alternate table. All registers contain identical fields.

Address: Base + 0xC8 (LUT0 alternate lower)
Base + 0xCC (LUT1 alternate lower)
Base + 0xD0 (LUT2 alternate lower)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRIO7[3:0]				PRIO6[3:0]				PRIO5[3:0]				PRIO4[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PRIO3[3:0]				PRIO2[3:0]				PRIO1[3:0]				PRIO0[3:0]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-8. LUT0–LUT2 Alternate Lower Register

Table 37-8. LUT0–LUT2 Alternate Lower field descriptions

Field	Description
PRIO7[3:0]	Priority setting if 7 ACKs for own channel counted
PRIO6[3:0]	Priority setting if 6 ACKs for own channel counted
PRIO5[3:0]	Priority setting if 5 ACKs for own channel counted
PRIO4[3:0]	Priority setting if 4 ACKs for own channel counted
PRIO3[3:0]	Priority setting if 3 ACKs for own channel counted
PRIO2[3:0]	Priority setting if 2 ACKs for own channel counted
PRIO1[3:0]	Priority setting if 1 ACK for own channel counted
PRIO0[3:0]	Priority setting if 0 ACKs for own channel counted

37.3.2.8 Performance Monitor Configuration Register (PERMON_CONFIG)

Address: Base + 0x00DC Access: User read/write

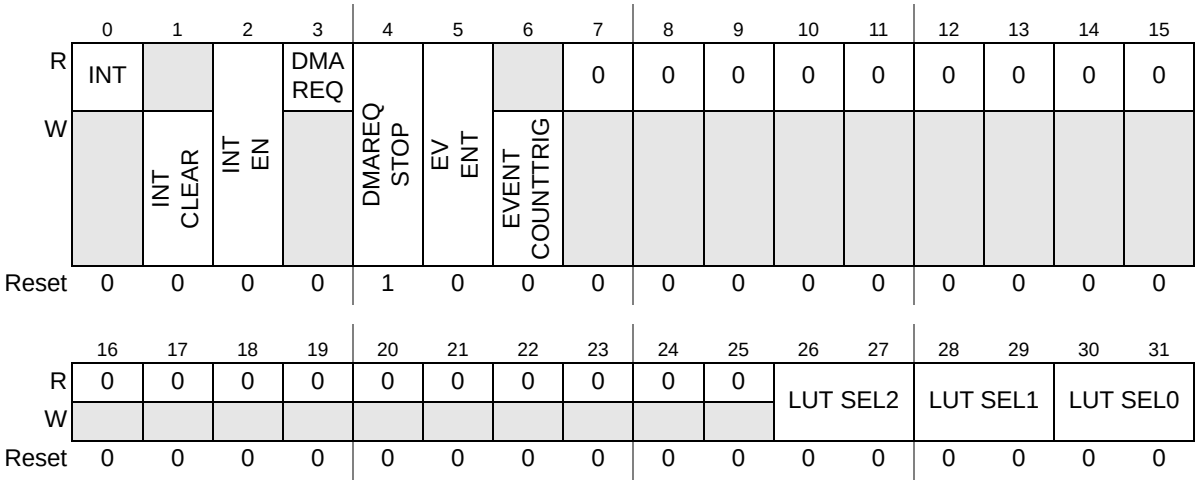


Figure 37-9. Performance Monitor Configuration Register (PERMON_CONFIG)

Table 37-9. PERMON_CONFIG field descriptions field descriptions

Field	Description
INT	Read-Only Sticky Bit. Interrupt pending register. Set when interrupt is made pending.
INTCLEAR	Write-Only. Writing a 1 to this bit clears the INT bit.
INTEN	Interrupt Enable. When this bit is 1 and the INT bit is 1, the processor gets an interrupt request.
DMAREQ	Read-Only. DMA request. Set when event counter time reaches 0. Cleared when first counter register is read.
DMAREQSTOP[1]	Read/Write. 0 DMA request functions as expected. 1 DMA request is cleared and cannot get set.
EVENTCOUNT FREERUN	Event Counter. 0 Event Counter Single-Shot. After reaching 0, the event time counter stays at 0 and is not reloaded. 1 Event Counter Free Run. After reaching zero, the event time counter is reloaded from event time preset and a new cycle starts.
EVENTCOUNT TRIGGER	Write-Only Bit. Writing to this bit causes all count registers to be transferred to the buffer registers, and subsequent be cleared. It causes the event counter to be reloaded from the event time preset register. No interrupt or DMA request is generated on writing this register, but both are generated when the event time counter register reaches 0.
LUT SEL*n*	Selectors between primary and secondary look-up table configuration register. These selectors determine which LUT table is used for the summed priority counters. The priorities entered into these counters may depend on a different LUT table than the priorities sent to the DRAM controller. 00 Select main look-up table configuration register. 01 Select alternate look-up table configuration register. 10 Select alternate look-up table configuration register if congested flag is set[2]. 11 Select alternate look-up table configuration register if bus 0 incoming. BUS0_HIPRIORITY flag is high.

1 This bit should be set as long as the DMA channel is not configured to manage the request. After configuring the DMA, clear the bit, and data starts to be transferred on every time tick.

2 Congested flag is explained in Section 37.4.2, Congestion detector.

37.3.2.9 Event Time Counter Register

The Event Time Counter register contains the 24-bit EVENT_TIME_COUNTER field. The counter decrements to 0. The interrupt and the DMA request are made pending when it reachers 0. On reaching 0, the counter reloads from the event count preset register if the EVENTCOUNTFREERUN bit is set in the PERMON_CONFIG register (see Section 37.3.2.8, Performance Monitor Configuration Register (PERMON_CONFIG)).

On reaching 0, all performance monitor count registers are loaded in the performance monitor buffer registers, and cleared.

Address: Base + 0x00E0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	EVENT_TIME_COUNTER[23:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EVENT_TIME_COUNTER[23:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-10. Event Time Counter Register

37.3.2.10 Event Time Preset Register

The Event Time Preset register contains the 24-bit preset value to be loaded into the event time counter register in case this preloads.

Address: Base + 0x00E4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	EVENT_TIME_PRESET[23:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EVENT_TIME_PRESET[23:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-11. Event Time Preset Register

37.3.2.11 Performance Monitor 1 and 2 Address Registers

These registers determine whether a processor access hits in the performance monitor 1 or performance monitor 2 address space.

- If ((e200 CPU address ≥ performance monitor 1 address low) &&
  (e200 CPU address < performance monitor 1 address hi))
  Increment *performance monitor 1 read counter* on reads
  Increment *performance monitor 1 write counter* on writes.
- If ((e200 CPU address ≥ performance monitor 2 address low) &&
  (e200 CPU address < performance monitor 2 address hi))
  Increment *performance monitor 2 read counter* on reads
  Increment *performance monitor 2 write counter* on writes.

Address: Base + 0x0E8 (PERF_MNTR1_ADDR_LOW)
Base + 0x0EC (PERF_MNTR2_ADDR_LOW)
Base + 0x0F0 (PERF_MNTR1_ADDR_HI)
Base + 0x0F4 (PERF_MNTR2_ADDR_HI)
Access: User read/write

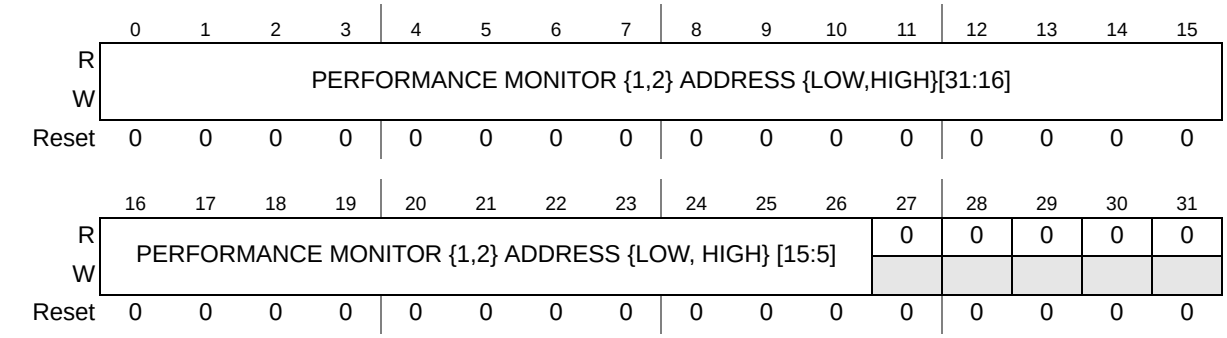


Figure 37-12. Performance Monitor Address Registers

Table 37-10. Performance Monitor Address Registers field descriptions

Field	Description
PERFORMANCE MONITOR ADDRESS	

37.3.2.12 Performance Monitor Read Counter Register

Address: Base + 0x100 (Performance Monitor 1 Read Counter)
Base + 0x104 (Performance Monitor 2 Read Counter)
Access: User read-only

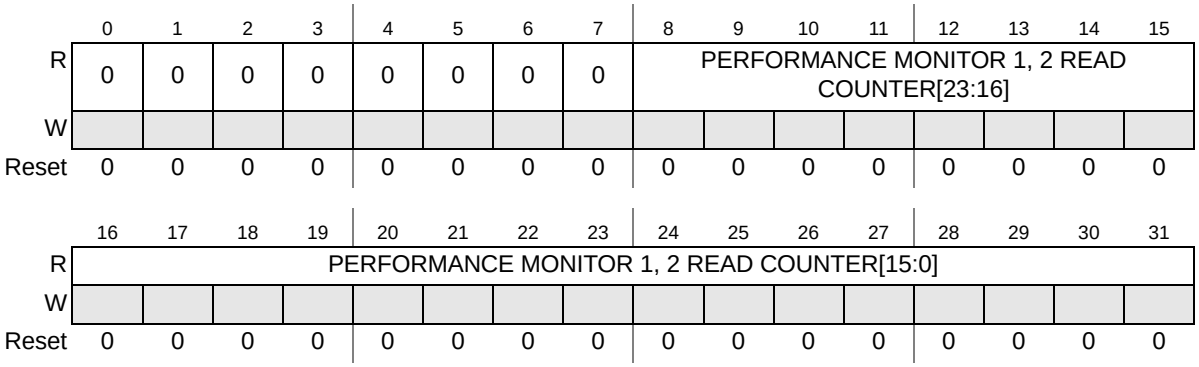


Figure 37-13. Performance Monitor Read Counter Register

Table 37-11. Performance Monitor Read Counter Register field descriptions

Field	Description
PERFORMANCE MONITOR 1–2 READ COUNTER	Every time the Processor performs a read access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor read counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.

37.3.2.13 Performance Monitor Write Counter Register

Address: Base + 0x108 (Performance Monitor 1 Write Counter)
Base + 0x10C (Performance Monitor 2 Write Counter)
Access: User read-only

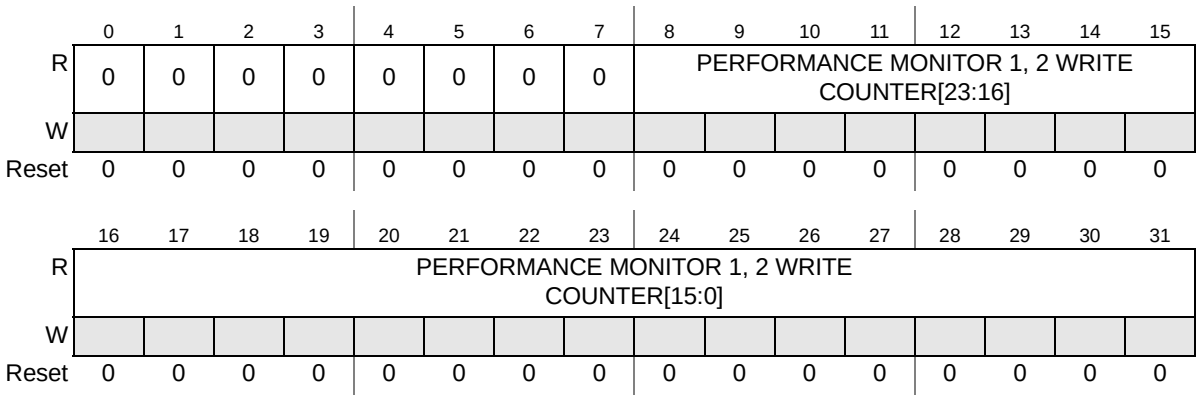


Figure 37-14. Performance Monitor Write Counter Registers

Table 37-12. Performance Monitor Counter Write Register field descriptions

Field	Description
PERFORMANCE MONITOR 1–2 WRITE COUNTER	Every time the Processor performs a write access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor write counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.

37.3.2.14 Granted ACK Counter Registers

Address: Base + 0x110 (Granted ACK counter 0)
Base + 0x114 (Granted ACK counter 1)
Base + 0x118 (Granted ACK counter 2)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	GRANTED ACK COUNTER 0–2[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GRANTED ACK COUNTER 0–2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-15. Granted ACK Counter 0–2 Registers

37.3.2.15 Cumulative Wait Counter Registers

Address: Base + 0x124 (Cumulative wait counter 0)
Base + 0x128 (Cumulative wait counter 1)
Base + 0x12C (Cumulative wait counter 2)
Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	CUMULATIVE WAIT COUNTER 0–2[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CUMULATIVE WAIT COUNTER 0–2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-16. Cumulative Wait Counter 0–2 Registers

37.3.2.16 Summed Priority Counter Registers

Address: Base + 0x138 (Summed priority counter 0)
Base + 0x13C (Summed priority counter 1)
Base + 0x140 (Summed priority counter 2)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SUMMED PRIORITY COUNTER 0–2[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUMMED PRIORITY COUNTER 0–2[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-17. Summed Priority Counter 0–2 Registers

37.3.2.17 Counter register descriptions and values

The counter registers contain 11 different 24-bit counter values. All these counter values count certain events. Table 37-13 gives the details on the nature of the event. Counter values Summed Priority Counter 2, Summed Priority Counter 3, and Summed Priority Counter 4 are available in two sets of registers. They are available in registers with the same name, but they are also available in a set of three other registers (granted ACK counter or cumulative wait counter registers). The multiple-mapping of the three upper Summed Priority Counter registers allows easy and compact DMA transfer to memory. Because of the multiple mapping, all 11 count values can be transferred to memory with a 64-byte DMA transfer starting at address 0x100. The multiple mapping allows the DMA to get all information with a 64-byte transfer, but some decompression is needed on decoding the data, while the CPU can read the 11 registers and mask out the upper 8 bits to get relevant information.

Table 37-13. Monitor counter descriptions

Field	Description
PERFORMANCE MONITOR 1–2 READ COUNTER	Every time the processor performs a read access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor read counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.
PERFORMANCE MONITOR 1–2 WRITE COUNTER	Every time the processor performs a write access with an address that hits in the address window for counter 1 or 2, the respective counter is incremented. An address hits in the address window for performance monitor write counter 1 if the address is higher or equal than the performance monitor 1 address low, and lower than the performance monitor 1 address high. Similar for the second counter.
GRANTED ACK COUNTER 0–2	Every time the multi-port DRAM controller grants a request for channel 0–2, the respective counter is incremented.

Table 37-13. Monitor counter descriptions (continued)

Field	Description
CUMULATIVE WAIT COUNTER 0–2	Every time a request is pending to the multi-port DRAM controller for channel 0–2 and it is not granted in the current cycle, the respective counter is incremented.
SUMMED PRIORITY COUNTER 0–2	Every time a request is granted by the multi-port DRAM controller for channel 0– 2, a priority code is added to the respective counter. See text for details.

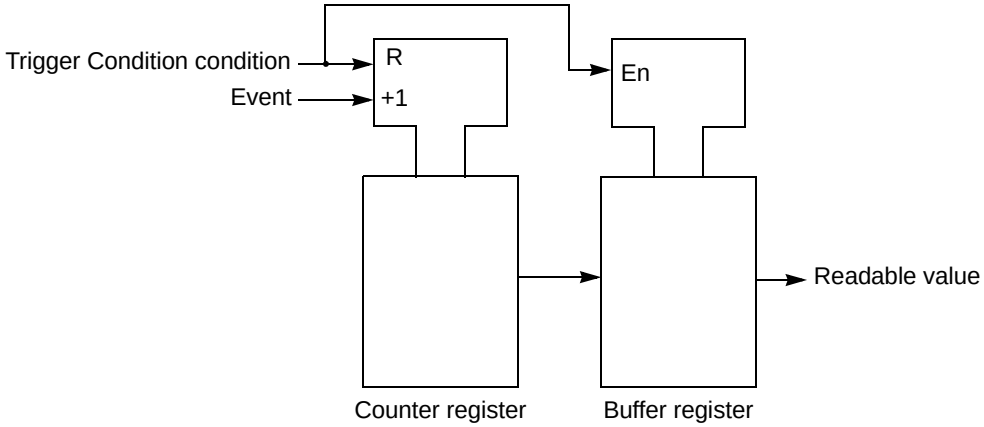


Figure 37-18. Monitor counters

All counters in Table 37-13 are double-buffered; Figure 37-18 gives details. There are always two registers associated with every counter. The first register is the counter. It counts the events mentioned in the table. When the trigger condition occurs, the time event counter reaches zero, and the counter register is transferred to the buffer register; the counter register is then cleared. When accessing the register, the buffer register value is always returned.

The priority code added may or may not be the same as the priority code the request used on the DRAM controller. The codes are equal if the LUT SEL field for the channel is the same in the PRIOMAN_CONFIG and PERMON_CONFIG registers. If the LUT SEL fields differ, the field in PRIOMAN_CONFIG is used to calculate the channel priority code on the DRAM, and the field in PERMON_CONFIG is used to calculate the priority code added to this register.

The possibility to use unequal LUT SEL fields makes it possible to use the main look-up tables for DRAM priority programming and the alternate look-up tables for performance monitoring. Making the look-up tables independent increases the possibility of what can be monitored.

37.4 Functional description

The priority manager calculates the outgoing priority for the channels of the multi-port DRAM controller. The priority of any channel at a given time is a function of the request-granting history of the DRAM controller. A granted request is called an ACK, so this schema is called an ACK-based schema, because the priority is determined by the history of which channels have been ACK-ed in the past and when.

The priority manager calculates the priorities in a dynamic way. This means that a priority is never constant, but changes over time even when the request is not serviced. As a request ages while it is not being serviced, its priority escalates to a higher level, and as the level increases, it is eventually serviced.

The DRAM controller has a built-in preference to offer repeat for any incoming read request. The repeat goes on as long as the requesting channel keeps requesting and its priority is greater than 0. When the outgoing priority for any channel is 0, the DRAM controller no longer services or repeats the request. This feature allows the priority manager to control the maximum repeat count for any incoming channel.

37.4.1 Description of operation—overview

Priority calculation for all channels is independent. There is no direct cross-dependency of the priority of one channel on the priority of another channel. The algorithm looks at the last N arbitration cycles on the bus. N is a programmable number, set by the ACK_COUNT*n* field in the PRIOMAN_CONFIG1 register, described in Figure 37-2. For the last N arbitration cycles, the number of times the own channel won the bus is summed up and saturated to a maximum of 15. This number of 0 to 15 is input into the applicable look-up table. LUT table 0 is for channel 0, LUT table 1 is for channel 1, and so on. The value for the particular number is the priority code going to the multi-port DRAM controller. If N is set to 16 and the own channel was granted the bus four times in the last 16 bus grants, the index into the look-up table is four. The field prio4[3:0] of the relevant look-up table is the priority going to the multi-port DRAM controller.

Each channel has two look-up tables: the main and the alternate. The algorithm may switch between both, depending on some settings. The default look-up table is the main. However, the alternate is used if:

- The particular channel has been configured to look at the bus0 incoming BUS0_HIPRIORITY flag.
- The particular channel has been configured to look at the congestion monitor, and this block indicates the multi-port DRAM is congested.

Figure 37-19 contains a block diagram of the priority channel.

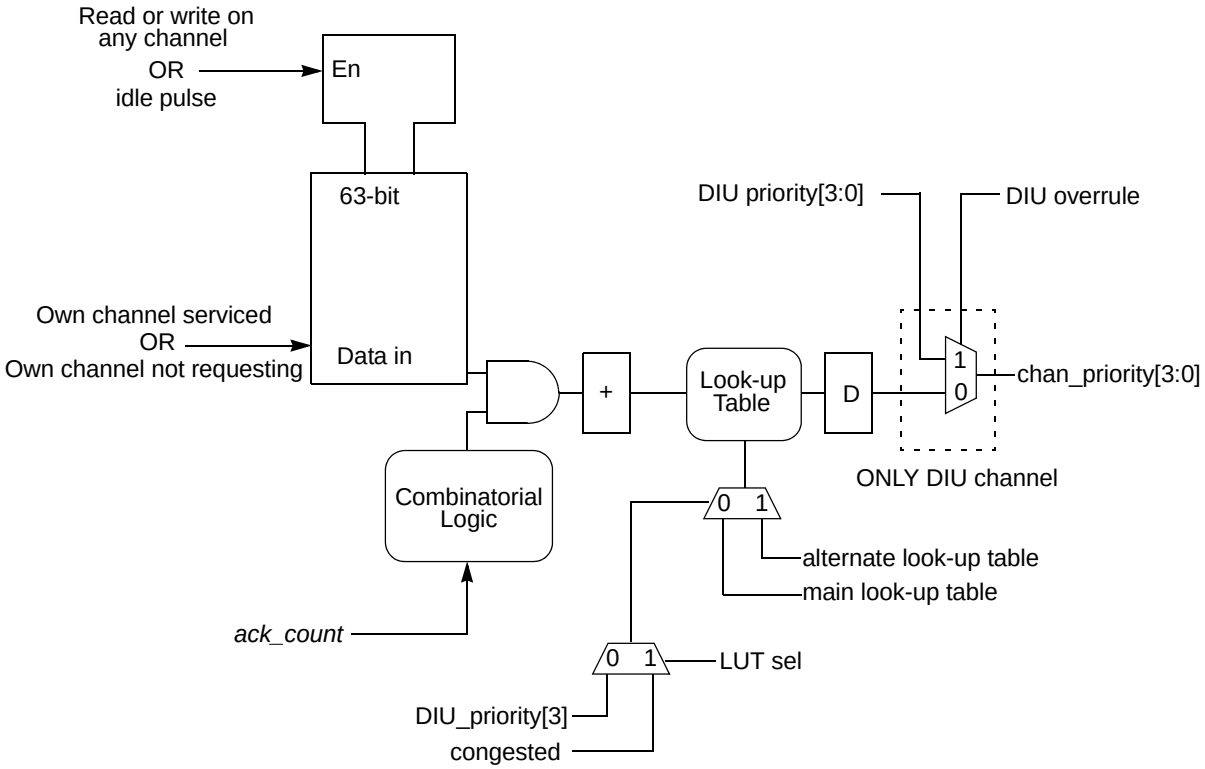


Figure 37-19. Priority channel block diagram

The shift register shifts in information of the recent ACKs. This 63-stage shift register contains information on the last 63 bus cycles of the DRAM controller.

The shift register is shifted any time a read or write request has been granted to the DRAM (an ACK to the requesting bus) or when there is an IDLE_PULSE. An idle pulse is generated every time the DRAM is idle for four consecutive clock cycles. Idle means none of the incoming buses is making a request.

The shift data in is the corrected ACK for the self channel. If the shift register shifts because the current cycle is granted to the self channel, a 1 is shifted in; if not, a 0 is shifted in. It always occurs like this when the own channel is requesting access. However, if the own channel is not requesting access, depending on control bit ACK_SEL, a 1 or a 0 is shifted in. If ACK_SEL is 1, a 1 is shifted in all the time when the self channel is not requesting and there is an ACK on any other channel or an idle pulse. If ACK_SEL is 0, zeros are shifted in.

The correction for the non-requesting channel allows steering the default priority, the priority that the channel gets when it has not been requesting for some time. If ACK_SEL is set to 1, the default priority is low. This setting is appropriate for peripherals with (large) FIFOs. When they are not requesting, the FIFO is quite full. When they do get on the bus, they can start with low priority and escalate to higher priority after some time.

Setting ACK_SEL to 0 is appropriate for peripherals that desire high priority. In this case, the Power Architecture processor should receive high priority. When it is not on the bus, it is because it finds the instruction or data that it needs in the processor caches, so it does not request. When the cache misses, the request comes on the bus, and needs to be serviced fast. Therefore, ACK_SEL is set to 0, the default

priority is high and servicing fast. If the Power Architecture Processor gets on the bus frequently (due to a lot of cache swapping), the priority manager detects this and degrades its priorities over time. The other masters continue to receive their fair share of bus bandwidth.

The output of the shift register is ANDed in to look at only the last N ACKs. Combinatorial logic decodes the ANDing code from register field ACK_COUNT. The number of ones after the ANDing is added up in ADDER 4 and saturated. The result out of ADDER 4 is a number from 0 to 15. This number is input in the look-up table. Table look-up content is taken for channel 1 from register LUT table 1 main[63:0] or LUT table 1 alternate[63:0]. Because of the 64-bit nature of the registers, four 32-bit registers are involved. The description is given in Figure 37-5, Figure 37-6, Figure 37-7, and Figure 37-8.

The MUX selects whether to use the main or the alternate register. The MUX condition has two possible sources again, selected by MUX **7**, by means of control bit LUT SEL described in register PRIOMAN_CONFIG, with details in Figure 37-2.

If LUT SEL is 1, the alternate table is selected when the multi-port controller is congested. If LUT SEL is 0, the alternate table is selected when bus0 incoming BUS0_HIPRIORITY is high.

The pipeline register is present purely for implementation reasons. It has no algorithmic function.

37.4.2 Congestion detector

The congestion detector's purpose is to detect when the multi-port DRAM controller is congested. Congestion is assumed if the share of the requests with priorities equal to or greater than eight is more than a certain percentage. If congestion occurs, the priority manager may react by exchanging the look-up tables with the alternate look-up tables. This reduces the average priority of the incoming requests. The reduced priorities mean that, on average, every incoming channel gets a lower priority and the DRAM controller tries harder to optimize on bandwidth and less to optimize to service the high-priority requests first. The switch-over is driven by the congestion state. If many requests come in on high priority, they all need to be serviced first, the congested flag goes high, and the controller reacts to this by reducing the request priorities (by switching in the alternate tables). Therefore, it can concentrate on the ones that are important and have room again for optimized bandwidth.

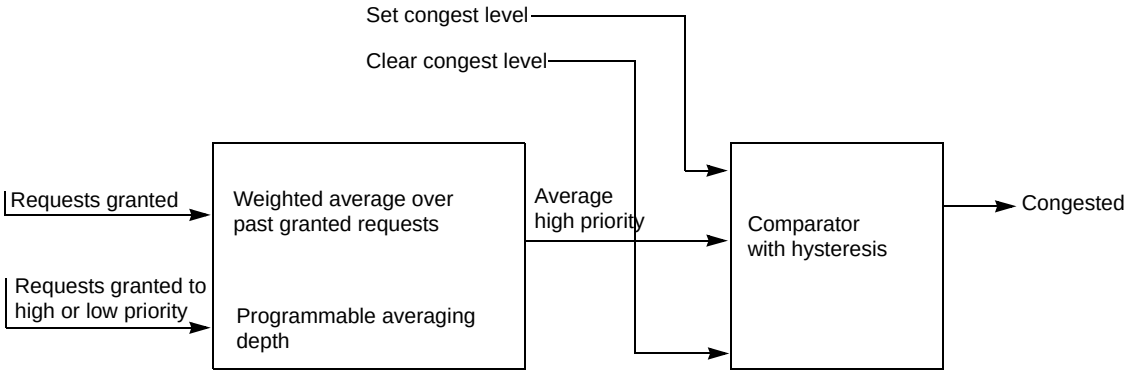


Figure 37-20. Congestion detector—simplified block diagram

A block diagram of the congestion detector is given in Figure 37-20. The congestion detector consists of an averaging block, followed by a comparator with hysteresis. The averaging block calculates the weighted average of the percentage of high-priority requests, as in the equation below.

$$\text{average priority} = \sum \text{weight(k)} \cdot \text{val(k)} \qquad \textit{Eqn. 37-1}$$

The weighted average uses an exponential weighting when looking at the past granted requests. Requests granted in a more distant past have a lower weighting coefficient. The weighting coefficient uses an exponential back-off, following the formula below.

$$\text{weight(k)} = \frac{1}{W0} \cdot \exp\left(-\frac{k}{W0}\right) \qquad \textit{Eqn. 37-2}$$

In this formula, *weight(k)* is the weighting coefficient used for the request granted k acknowledges ago, meaning k other requests have been granted after this one. The coefficient W0 is programmable, dependent on the control field filter bandwidth in register HIPRIO_CONFIG, detailed in Figure 37-4.

The value input in the weighting block, *val(k)*, depends on the priority of the request granted. It is 0 if the priority was 7 or lower; it is 0x1000 if the priority was 8 or higher.

The result of the weighted average is a number between 0 and 0x1000 input in the comparator with hysteresis. This result, AVERAGE_HIPRIORITY, can be monitored in the HIPRIO_CONFIG register (Figure 37-4).

The weighted averaging block is followed by a comparator with hysteresis, with a programmable low threshold.

- If AVERAGE_HIPRIORITY is greater than SET_CONGEST_LEVEL, the congested flag is set.
- If AVERAGE_HIPRIORITY is lower than CLEAR_CONGEST_LEVEL, the congested flag is cleared.

This page is intentionally left blank.

Chapter 38
Memory Protection Unit (MPU)

38.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in a device. Using pre-programmed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or that have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

38.2 Block diagram

A simplified block diagram of the MPU module is shown in Figure 38-1. The hardware's two-dimensional connection matrix is clearly visible with the basic access evaluation "macro" shown as the replicated submodule block. The XBAR bus slave ports (s{0,1,2,3}_h*) are shown on the left side of the diagram, the region descriptor registers in the middle, and the IPS bus interface (ips_*) on the right side. The evaluation macro contains the two magnitude comparators connected to the start and end address registers from each region descriptor (rgdn) as well as the combinational logic blocks to determine the region hit and the access protection error. For information on the details of the access evaluation macro, see Section 38.7.1, Access evaluation macro.

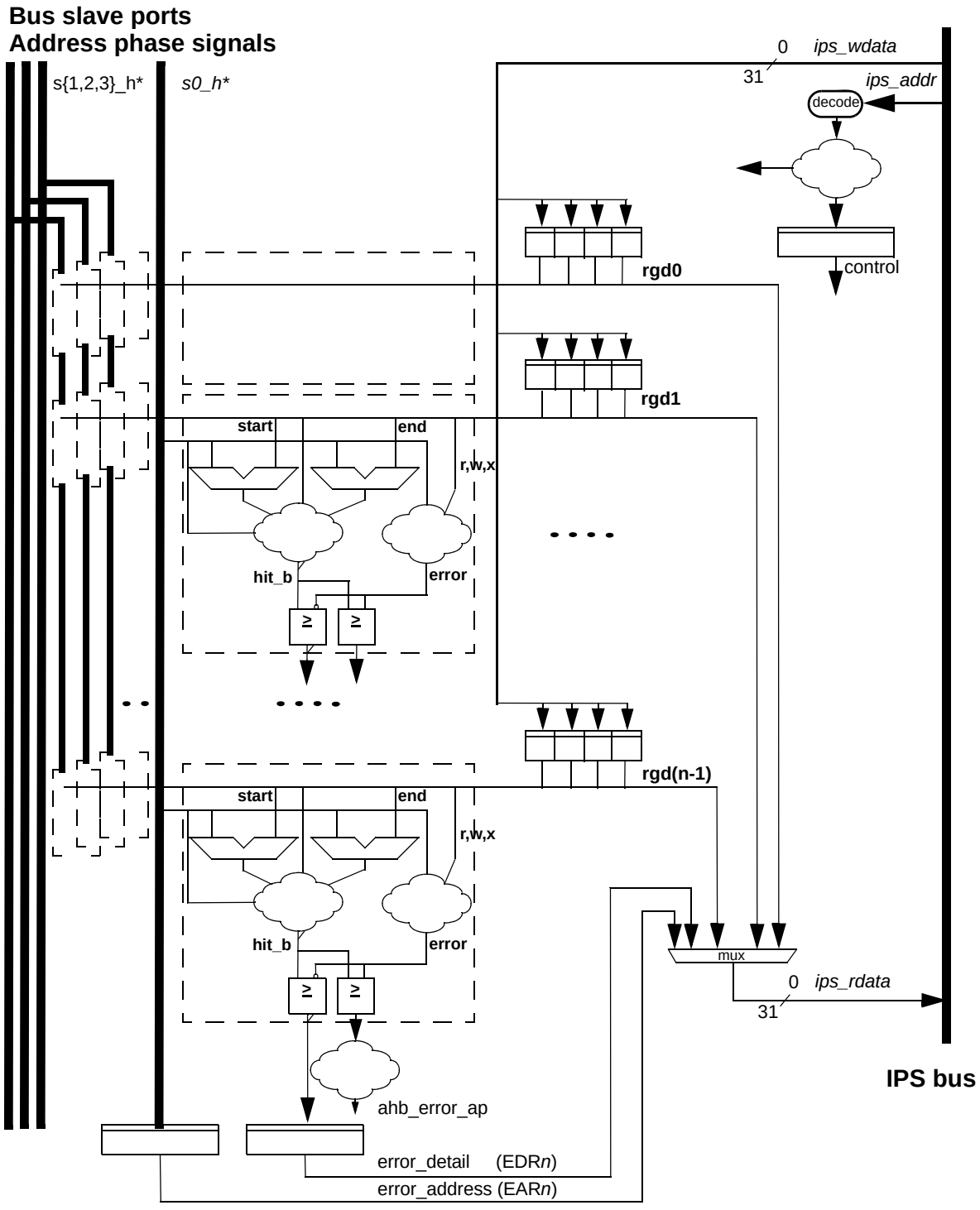


Figure 38-1. MPU block diagram

38.3 Features

The MPU implements a two-dimensional hardware array of memory region descriptors and the XBAR slave ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for 16 memory region descriptors (128 bits per descriptor)
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB.
  - 4 bus masters that support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses. The non-core masters are not used on this device.
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor.
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter only the access rights of a descriptor.
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software. See Section 38.7.2, Functional integration details, for details and Section 38.9, Application information, for an example.
- Support for 3 XBAR slave port connections
  - MPU hardware continuously monitors every XBAR slave port access using the pre-programmed memory region descriptors.
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in *all* memory regions where it does hit. In the event of an access error, the XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device.
  - 64-bit error registers, one for each XBAR slave port, capture the last faulting address, attributes and “detail” information.
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled.

On this device, the two instantiations of the MPU are referred to as MPU_0 (attached to the slave side of XBAR_0) and MPU_1 (attached to the slave side of XBAR_1). Both instantiations are identical and do not differ in Lock Step Mode (LSM) or Decoupled Parallel Mode (DPM). However, memory addressing differs between these two modes. See Table 38-2.

The MPU port allocation is shown in Table 38-1.

Table 38-1. MPU port allocation

XBAR slave port	MPU_0	MPU_1
Flash memory	Port 0	Port 0
SRAM	Port 1	Port 1
Peripheral bridge	Port 2	Port 2

38.4 Modes of operation

The MPU module does not support any special modes of operation. As a memory-mapped device located on the platform's high-speed system bus, it responds based strictly on the memory addresses of the connected system buses. The IPS bus is used to access the MPU's programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the XBAR system bus port(s).

Power dissipation is minimized when the MPU's global enable/disable bit is cleared (MPU_CESR[VLD] = 0).

38.5 External signal description

The MPU module does not include any external interface.

38.6 Memory map and register definition

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 KB space. The programming model is partitioned into three groups:

- Control/status registers
- The data structure containing the region descriptors
- The alternate view of the region descriptor access control values

The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an IPS error termination.

The MPU programming model map is shown in Table 38-3.

Table 38-2. MPU module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	MPU_0	0xFFF1_0000
	MPU_1	
Decoupled Parallel Mode (DPM)	MPU_0	0xFFF1_0000 (same as LSM)
	MPU_1	0x8FF1_0000

Table 38-3. MPU memory map

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	MPU Control/Error Status Register (MPU_CESR)	R/W	0x0080_3200	on page 1404
0x0004–0x000F	Reserved			
0x0010	MPU Error Address Register, Slave Port 0 (MPU_EAR0)	R	0x0000_0000	on page 1405

Table 38-3. MPU memory map (continued)

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0014	MPU Error Detail Register, Slave Port 0 (MPU_EDR0)	R	0x0000_0000	on page 1405
0x0018	MPU Error Address Register, Slave Port 1 (MPU_EAR1)	R	0x0000_0000	on page 1405
0x001C	MPU Error Detail Register, Slave Port 1 (MPU_EDR1)	R	0x0000_0000	on page 1405
0x0020	MPU Error Address Register, Slave Port 2 (MPU_EAR2)	R	0x0000_0000	on page 1405
0x0024	MPU Error Detail Register, Slave Port 2 (MPU_EDR2)	R	0x0000_0000	on page 1405
0x0028–0x03FF	Reserved			
0x0400	MPU Region Descriptor 0 (MPU_RGD0) (128-bit area)			on page 1407
0x0400	MPU Region Descriptor 0, Word 0 (MPU_RGD0.Word0)	R/W	0x0000_0000	on page 1407
0x0404	MPU Region Descriptor 0, Word 1 (MPU_RGD0.Word1)	R/W	0xFFFF_FFFF	on page 1408
0x0408	MPU Region Descriptor 0, Word 2 (MPU_RGD0.Word2)	R/W	0x0061_8600	on page 1408
0x040C	MPU Region Descriptor 0, Word 3 (MPU_RGD0.Word3)	R/W	0x0000_0000	on page 1411
0x0410	MPU Region Descriptor 1 (MPU_RGD1) (128-bit area)			on page 1407
0x0410	MPU Region Descriptor 1, Word 0 (MPU_RGD1.Word0)	R/W	0x0000_0000	on page 1407
0x0414	MPU Region Descriptor 1, Word 1 (MPU_RGD1.Word1)	R/W	0x0000_001F	on page 1408
0x0418	MPU Region Descriptor 1, Word 2 (MPU_RGD1.Word2)	R/W	0x0000_0000	on page 1408
0x041C	MPU Region Descriptor 1, Word 3 (MPU_RGD1.Word3)	R/W	0x0000_0000	on page 1411
0x0420	MPU Region Descriptor 2 (MPU_RGD2) (128-bit area)			on page 1407
0x0420	MPU Region Descriptor 2, Word 0 (MPU_RGD2.Word0)	R/W	0x0000_0000	on page 1407
0x0424	MPU Region Descriptor 2, Word 1 (MPU_RGD2.Word1)	R/W	0x0000_001F	on page 1408
0x0428	MPU Region Descriptor 2, Word 2 (MPU_RGD2.Word2)	R/W	0x0000_0000	on page 1408
0x042C	MPU Region Descriptor 2, Word 3 (MPU_RGD2.Word3)	R/W	0x0000_0000	on page 1411
0x0430	MPU Region Descriptor 3 (MPU_RGD3) (128-bit area)			on page 1407

Table 38-3. MPU memory map (continued)

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0430	MPU Region Descriptor 3, Word 0 (MPU_RGD3.Word0)	R/W	0x0000_0000	on page 1407
0x0434	MPU Region Descriptor 3, Word 1 (MPU_RGD3.Word1)	R/W	0x0000_001F	on page 1408
0x0438	MPU Region Descriptor 3, Word 2 (MPU_RGD3.Word2)	R/W	0x0000_0000	on page 1408
0x043C	MPU Region Descriptor 3, Word 3 (MPU_RGD3.Word3)	R/W	0x0000_0000	on page 1411
0x0440	MPU Region Descriptor 4 (MPU_RGD4) (128-bit area)			on page 1407
0x0440	MPU Region Descriptor 4, Word 0 (MPU_RGD4.Word0)	R/W	0x0000_0000	on page 1407
0x0444	MPU Region Descriptor 4, Word 1 (MPU_RGD4.Word1)	R/W	0x0000_001F	on page 1408
0x0448	MPU Region Descriptor 4, Word 2 (MPU_RGD4.Word2)	R/W	0x0000_0000	on page 1408
0x044C	MPU Region Descriptor 4, Word 3 (MPU_RGD4.Word3)	R/W	0x0000_0000	on page 1411
0x0450	MPU Region Descriptor 5 (MPU_RGD5) (128-bit area)			on page 1407
0x0450	MPU Region Descriptor 5, Word 0 (MPU_RGD5.Word0)	R/W	0x0000_0000	on page 1407
0x0454	MPU Region Descriptor 5, Word 1 (MPU_RGD5.Word1)	R/W	0x0000_001F	on page 1408
0x0458	MPU Region Descriptor 5, Word 2 (MPU_RGD5.Word2)	R/W	0x0000_0000	on page 1408
0x045C	MPU Region Descriptor 5, Word 3 (MPU_RGD5.Word3)	R/W	0x0000_0000	on page 1411
0x0460	MPU Region Descriptor 6 (MPU_RGD6) (128-bit area)			on page 1407
0x0460	MPU Region Descriptor 6, Word 0 (MPU_RGD6.Word0)	R/W	0x0000_0000	on page 1407
0x0464	MPU Region Descriptor 6, Word 1 (MPU_RGD6.Word1)	R/W	0x0000_001F	on page 1408
0x0468	MPU Region Descriptor 6, Word 2 (MPU_RGD6.Word2)	R/W	0x0000_0000	on page 1408
0x046C	MPU Region Descriptor 6, Word 3 (MPU_RGD6.Word3)	R/W	0x0000_0000	on page 1411
0x0470	MPU Region Descriptor 7 (MPU_RGD7) (128-bit area)			on page 1407
0x0470	MPU Region Descriptor 7, Word 0 (MPU_RGD7.Word0)	R/W	0x0000_0000	on page 1407

Table 38-3. MPU memory map (continued)

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x0474	MPU Region Descriptor 7, Word 1 (MPU_RGD7.Word1)	R/W	0x0000_001F	on page 1408
0x0478	MPU Region Descriptor 7, Word 2 (MPU_RGD7.Word2)	R/W	0x0000_0000	on page 1408
0x047C	MPU Region Descriptor 7, Word 3 (MPU_RGD7.Word3)	R/W	0x0000_0000	on page 1411
0x0480	MPU Region Descriptor 8 (MPU_RGD8) (128-bit area)			on page 1407
0x0480	MPU Region Descriptor 8, Word 0 (MPU_RGD8.Word0)	R/W	0x0000_0000	on page 1407
0x0484	MPU Region Descriptor 8, Word 1 (MPU_RGD8.Word1)	R/W	0x0000_001F	on page 1408
0x0488	MPU Region Descriptor 8, Word 2 (MPU_RGD8.Word2)	R/W	0x0000_0000	on page 1408
0x048C	MPU Region Descriptor 8, Word 3 (MPU_RGD8.Word3)	R/W	0x0000_0000	on page 1411
0x0490	MPU Region Descriptor 9 (MPU_RGD9) (128-bit area)			on page 1407
0x0490	MPU Region Descriptor 9, Word 0 (MPU_RGD9.Word0)	R/W	0x0000_0000	on page 1407
0x0494	MPU Region Descriptor 9, Word 1 (MPU_RGD9.Word1)	R/W	0x0000_001F	on page 1408
0x0498	MPU Region Descriptor 9, Word 2 (MPU_RGD9.Word2)	R/W	0x0000_0000	on page 1408
C0x049	MPU Region Descriptor 9, Word 3 (MPU_RGD9.Word3)	R/W	0x0000_0000	on page 1411
0x04A0	MPU Region Descriptor 10 (MPU_RGD10) (128-bit area)			on page 1407
0x04A0	MPU Region Descriptor 10, Word 0 (MPU_RGD10.Word0)	R/W	0x0000_0000	on page 1407
0x04A4	MPU Region Descriptor 10, Word 1 (MPU_RGD10.Word1)	R/W	0x0000_001F	on page 1408
0x04A8	MPU Region Descriptor 10, Word 2 (MPU_RGD10.Word2)	R/W	0x0000_0000	on page 1408
0x04AC	MPU Region Descriptor 10, Word 3 (MPU_RGD10.Word3)	R/W	0x0000_0000	on page 1411
0x04B0	MPU Region Descriptor 11 (MPU_RGD11) (128-bit area)			on page 1407
0x04B0	MPU Region Descriptor 11, Word 0 (MPU_RGD11.Word0)	R/W	0x0000_0000	on page 1407
0x04B4	MPU Region Descriptor 11, Word 1 (MPU_RGD11.Word1)	R/W	0x0000_001F	on page 1408

Table 38-3. MPU memory map (continued)

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x04B8	MPU Region Descriptor 11, Word 2 (MPU_RGD11.Word2)	R/W	0x0000_0000	on page 1408
0x04BC	MPU Region Descriptor 11, Word 3 (MPU_RGD11.Word3)	R/W	0x0000_0000	on page 1411
0x04C0	MPU Region Descriptor 12 (MPU_RGD12) (128-bit area)			on page 1407
0x04C0	MPU Region Descriptor 12, Word 0 (MPU_RGD12.Word0)	R/W	0x0000_0000	on page 1407
0x04C4	MPU Region Descriptor 12, Word 1 (MPU_RGD12.Word1)	R/W	0x0000_001F	on page 1408
0x04C8	MPU Region Descriptor 12, Word 2 (MPU_RGD12.Word2)	R/W	0x0000_0000	on page 1408
0x04CC	MPU Region Descriptor 12, Word 3 (MPU_RGD12.Word3)	R/W	0x0000_0000	on page 1411
0x04D0	MPU Region Descriptor 13 (MPU_RGD13) (128-bit area)			on page 1407
0x04D0	MPU Region Descriptor 13, Word 0 (MPU_RGD13.Word0)	R/W	0x0000_0000	on page 1407
0x04D4	MPU Region Descriptor 13, Word 1 (MPU_RGD13.Word1)	R/W	0x0000_001F	on page 1408
0x04D8	MPU Region Descriptor 13, Word 2 (MPU_RGD13.Word2)	R/W	0x0000_0000	on page 1408
0x04DC	MPU Region Descriptor 13, Word 3 (MPU_RGD13.Word3)	R/W	0x0000_0000	on page 1411
0x04E0	MPU Region Descriptor 14 (MPU_RGD14) (128-bit area)			on page 1407
0x04E0	MPU Region Descriptor 14, Word 0 (MPU_RGD14.Word0)	R/W	0x0000_0000	on page 1407
0x04E4	MPU Region Descriptor 14, Word 1 (MPU_RGD14.Word1)	R/W	0x0000_001F	on page 1408
0x04E8	MPU Region Descriptor 14, Word 2 (MPU_RGD14.Word2)	R/W	0x0000_0000	on page 1408
0x04EC	MPU Region Descriptor 14, Word 3 (MPU_RGD14.Word3)	R/W	0x0000_0000	on page 1411
0x04F0	MPU Region Descriptor 15 (MPU_RGD15) (128-bit area)			on page 1407
0x04F0	MPU Region Descriptor 15, Word 0 (MPU_RGD15.Word0)	R/W	0x0000_0000	on page 1407
0x04F4	MPU Region Descriptor 15, Word 1 (MPU_RGD15.Word1)	R/W	0x0000_001F	on page 1408
0x04F8	MPU Region Descriptor 15, Word 2 (MPU_RGD15.Word2)	R/W	0x0000_0000	on page 1408

Table 38-3. MPU memory map (continued)

Offset from MPU_BASE	Register	Access[1]	Reset Value[2]	Location
0x04FC	MPU Region Descriptor 15, Word 3 (MPU_RGD15.Word3)	R/W	0x0000_0000	on page 1411
0x0500–0x07FF	Reserved			
0x0800	MPU RGD Alternate Access Control 0 (MPU_RGDAAC0)	R/W	0x0061_8600	on page 1412
0x0804	MPU RGD Alternate Access Control 1 (MPU_RGDAAC1)	R/W	0x0000_0000	on page 1412
0x0808	MPU RGD Alternate Access Control 2 (MPU_RGDAAC2)	R/W	0x0000_0000	on page 1412
0x080C	MPU RGD Alternate Access Control 3 (MPU_RGDAAC3)	R/W	0x0000_0000	on page 1412
0x0810	MPU RGD Alternate Access Control 4 (MPU_RGDAAC4)	R/W	0x0000_0000	on page 1412
0x0814	MPU RGD Alternate Access Control 5 (MPU_RGDAAC5)	R/W	0x0000_0000	on page 1412
0x0818	MPU RGD Alternate Access Control 6 (MPU_RGDAAC6)	R/W	0x0000_0000	on page 1412
0x081C	MPU RGD Alternate Access Control 7 (MPU_RGDAAC7)	R/W	0x0000_0000	on page 1412
0x0820	MPU RGD Alternate Access Control 8 (MPU_RGDAAC8)	R/W	0x0000_0000	on page 1412
0x0824	MPU RGD Alternate Access Control 9 (MPU_RGDAAC9)	R/W	0x0000_0000	on page 1412
0x0828	MPU RGD Alternate Access Control 10 (MPU_RGDAAC10)	R/W	0x0000_0000	on page 1412
0x082C	MPU RGD Alternate Access Control 11 (MPU_RGDAAC11)	R/W	0x0000_0000	on page 1412
0x0830	MPU RGD Alternate Access Control 12 (MPU_RGDAAC12)	R/W	0x0000_0000	on page 1412
0x0834	MPU RGD Alternate Access Control 13 (MPU_RGDAAC13)	R/W	0x0000_0000	on page 1412
0x0838	MPU RGD Alternate Access Control 14 (MPU_RGDAAC14)	R/W	0x0000_0000	on page 1412
0x083C	MPU RGD Alternate Access Control 15 (MPU_RGDAAC15)	R/W	0x0000_0000	on page 1412
0x0840–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

38.6.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPERR								1	0	0	0	HRL			
W	w1c															
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NSP				NRGD				0	0	0	0	0	0	0	VLD
W																
Reset	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 38-2. MPU Control/Error Status Register (MPU_CESR)

Table 38-4. MPU_CESR field descriptions

Field	Description
SPERR	Slave Port n Error, where the slave port number matches the bit number. Each bit in this field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR*n* and MPU_EDR*n* registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written as a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A “find first one” instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do not contain a captured error. 1 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do contain a captured error. The bits are mapped as follows: Bit 0 Error on Slave Port 0 (flash memory). Bit 1 Error on Slave Port 1 (SRAM). Bit 2 Error on Slave Port 2 (PBRIDGE) Bits 3–7 are not implemented. **Note:** Port assignments are mode-specific for LSM and DPM. See Chapter 9, Multi-Layer AHB Crossbar Switch (XBAR).
HRL	Hardware Revision Level. This 4-bit read-only field specifies the MPU’s hardware and definition revision level. It can be read by software to determine the functional definition of the module.
NSP	Number of Slave Ports. This 4-bit read-only field specifies the number of slave ports connected to the MPU.
NRGD	Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 00 8 region descriptors. 01 12 region descriptors. 10 16 region descriptors. 11 Reserved
VLD	Valid. This bit provides a global enable/disable for the MPU. When the MPU is disabled, all accesses from all bus masters are allowed. 0 The MPU is disabled. 1 The MPU is enabled.

38.6.2 MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

When the MPU detects an access error on slave port n, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* register at the same time. This register and the corresponding MPU_EDR*n* register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

Address: Base + 0x010 (MPU_EAR0)
Base + 0x018 (MPU_EAR1)
Base + 0x020 (MPU_EAR2)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EADDR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EADDR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38-3. MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

Table 38-5. MPU_EAR*n* field descriptions

Field	Description
EADDR	Error Address. This read-only field is the reference address from slave port *n* that generated the access error.

38.6.3 MPU Error Detail Register, Slave Port *n* (MPU_EDR*n*)

When the MPU detects an access error on slave port n, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU_EAR*n* register at the same time. Note this register and the corresponding MPU_EAR*n* register contain the most recent access error; there are no hardware interlocks with the MPU_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

Address: Base + 0x014 (MPU_EDR0)
Base + 0x01C (MPU_EDR1)
Base + 0x024 (MPU_EDR2)

Access: User read-only

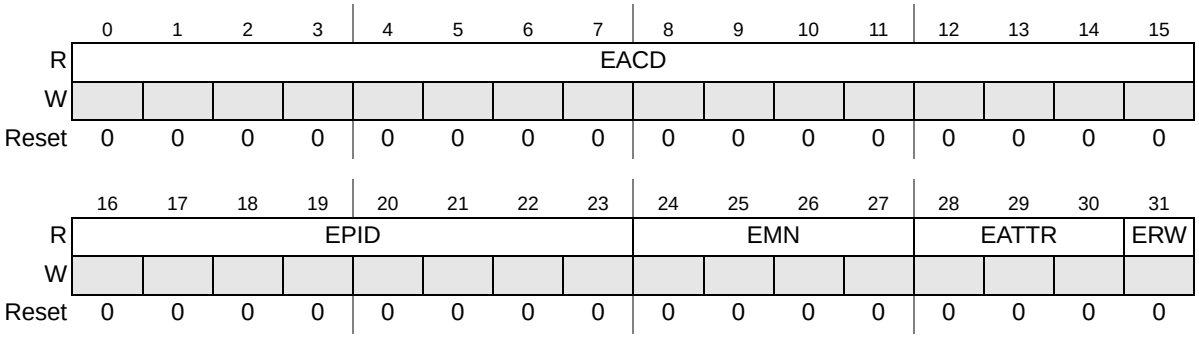


Figure 38-4. MPU Error Detail Register, Slave Port *n* (MPU_EDR*n*)

Table 38-6. MPU_EDR*n* field descriptions

Field	Description
EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit where the error occurred. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDR*n* register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits. If only a single EACD bit is set, then the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, then the protection error was caused in an overlapping set of region descriptors.
EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven only by processor cores; for other bus masters, this field is cleared.
EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field determines the bus master that generated the access error. 0000 Master 0. 0001 Master 1. 0010 Master 2. 0011 Master 3. 0100 Master 4 (Reserved, Master not implemented on MPC5675K). 0101 Master 5 (Reserved, Master not implemented on MPC5675K). 0110 Master 6 (not implemented on XBAR_2). 0111 Master 7 (Reserved, Master not implemented on MPC5675K). All other values are reserved. **Note:** Port assignments are mode-specific for LSM and DPM. See Chapter 9, Multi-Layer AHB Crossbar Switch (XBAR).

Table 38-6. MPU_EDR*n* field descriptions (continued)

Field	Description
EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 00 User mode, instruction access. 01 User mode, data access. 10 Supervisor mode, instruction access. 11 Supervisor mode, data access. All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read. 1 Write.

38.6.4 MPU Region Descriptor *n* (MPU_RGD*n*)

Each 128-bit (four 32-bit word) region descriptor specifies a given memory space and the access attributes associated with that space.

The region descriptors are organized sequentially in the MPU's programming model and each of the four 32-bit words are detailed in the subsequent sections.

38.6.4.1 MPU Region Descriptor *n*, Word 0 (MPU_RGD*n*.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor's valid bit (see Section 38.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3), for more information).

Address: Base + 0x0x400 + (16 × *n*) + 0x0
See Table 38-3 for details.

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SRTADDR[26:11]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SRTADDR[10:0]											0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38-5. MPU Region Descriptor *n*, Word 0 Register (MPU_RGD*n*.Word0)

Table 38-7. MPU_RGD*n*.Word0 field descriptions

Field	Description
SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

38.6.4.2 MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor's valid bit (see Section 38.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3), for more information).

Address: Base + 0x0x400 + (16 × *n*) + 0x4
See Table 38-3 for details.

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ENDADDR[26:11]															
Reset (*n* = 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset (*n* > 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENDADDR[10:0]											1	1	1	1	1
W																
Reset (*n* = 0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset (*n* > 0)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 38-6. MPU Region Descriptor *n*, Word 1 Register (MPU_RGD*n*.Word1)

Table 38-8. MPU_RGD*n*.Word1 field descriptions

Field	Description
ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; it is software's responsibility to properly load these region descriptor fields.

38.6.4.3 MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. Bus masters 0–3 have a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. For these fields, the bus master number refers to the logical master number as specified in the "Logical Master IDs" table in the XBAR chapter. See Table 9-1. (Nexus controllers have IDs 8 and 9, but only the last 3 bits of the ID are used for this purpose, so they share privileges with cores 0 and 1.)

NOTE

The logical master number depends on whether the MPC5675K is operating in LSM or DPM, as shown in Table 9-1.

For the processor privilege rights, three flags are associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

For bus masters 4–7, only read/write access can be controlled.

NOTE

Bus master 7 is not implemented on the MPC5675K. See Table 9-1.

Writes to this word clear the region descriptor's valid bit (see Section 38.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3), for more information). Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (Alternate Access Control *n*) as stores to these locations do not affect the descriptor's valid bit.

Address: Base + 0x0x400 + (16 × *n*) + 0x8
See Table 38-3 for details.

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	M3PE	M3SM			M3UM		M2PE	M2SM1
W																
Reset (n = 0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n > 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M2SM0	M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W																
Reset (n = 0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n > 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38-7. MPU Region Descriptor *n*, Word 2 Register (MPU_RGD*n*.Word2)

Table 38-9. MPU_RGD*n*.Word2 field descriptions

Field	Description
M6RE	Bus master 6 read enable. If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.
M6WE	Bus master 6 write enable. If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.
M5RE	Bus master 5 read enable. If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.

Table 38-9. MPU_RGD*n*.Word2 field descriptions (continued)

Field	Description
M5WE	Bus master 5 write enable. If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.
M4RE	Bus master 4 read enable. If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.
M4WE	Bus master 4 write enable. If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 00 `r, w, x` = read, write, and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M3UM for user mode.
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 00 `r, w, x` = read, write, and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M2UM for user mode.
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 00 `r, w, x` = read, write, and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M1UM for user mode.

Table 38-9. MPU_RGD*n*.Word2 field descriptions (continued)

Field	Description
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 00 r, w, x = read, write, and execute allowed. 01 r, –, x = read and execute allowed, but no write. 10 r, w, – = read and write allowed, but no execute. 11 Same access controls as that defined by M0UM for user mode.
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

38.6.4.4 MPU Region Descriptor *n*, Word 3 (MPU_RGD*n*.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Since the region descriptor is a 4-word entity, there are potential coherency issues as this structure is being updated since multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGD*n*.Word0, then MPU_RGD*n*.Word1,... and finally MPU_RGD*n*.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of MPU_RGD*n*.Word2 is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (Alternate Access Control *n*) as stores to these locations do *not* affect the descriptor's valid bit.

Address: Base + 0x0x400 + (16 × *n*) + 0xC
See Table 38-3 for details.

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PID								PIDMASK							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38-8. MPU Region Descriptor *n*, Word 3 Register (MPU_RGD*n*.Word3)

Table 38-10. MPU_RGDn.Word3 field descriptions

Field	Description
PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.
PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, then the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see Section 38.7.1.1, Access evaluation—hit determination.
VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, while a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid. 1 Region descriptor is valid.

38.6.5 MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAAC*n*)

As noted in Section 38.6.4.3, MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2), it is expected that since system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, *an alternate programming view of this 32-bit entity is desired*. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (Alternate Access Control *n*) as stores to these locations do *not* affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU_RGD*n*.Word2.

Address: Base + 0x800 + (4 × *n*)
See Table 38-3 for details.

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	M3PE	M3SM			M3UM		M2PE	M2SM1
W																
Reset (n = 0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n > 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M2SM0	M2UM			M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W																
Reset (n = 0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n > 0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38-9. MPU RGD Alternate Access Control *n* (MPU_RGDAAC*n*)

Since the MPU_RGDAAC*n* register is another memory mapping for MPU_RGD*n*.Word2. The field definitions shown in Table 38-11 are identical to those presented in Table 38-9.

Table 38-11. MPU_RGDAAC*n* field descriptions

Field	Description
M6RE	Bus master 6 read enable. If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.
M6WE	Bus master 6 write enable. If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.
M5RE	Bus master 5 read enable. If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.
M5WE	Bus master 5 write enable. If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.
M4RE	Bus master 4 read enable. If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.
M4WE	Bus master 4 write enable. If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.
M3PE	Bus master 3 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGD*n*.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M3SM	Bus master 3 supervisor mode access control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 00 `r, w, x` = read, write and execute allowed. 01 `r, –, x` = read and execute allowed, but no write. 10 `r, w, –` = read and write allowed, but no execute. 11 Same access controls as that defined by M3UM for user mode.

Table 38-11. MPU_RGDAAC*n* field descriptions (continued)

Field	Description
M3UM	Bus master 3 user mode access control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus master 2 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M2SM	Bus master 2 supervisor mode access control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 00 r, w, x = read, write and execute allowed. 01 r, –, x = read and execute allowed, but no write. 10 r, w, – = read and write allowed, but no execute. 11 Same access controls as that defined by M2UM for user mode.
M2UM	Bus master 2 user mode access control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus master 1 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus master 1 supervisor mode access control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 00 r, w, x = read, write and execute allowed. 01 r, –, x = read and execute allowed, but no write. 10 r, w, – = read and write allowed, but no execute. 11 Same access controls as that defined by M1UM for user mode.
M1UM	Bus master 1 user mode access control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus master 0 process identifier enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M0SM	Bus master 0 supervisor mode access control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 00 r, w, x = read, write and execute allowed. 01 r, –, x = read and execute allowed, but no write. 10 r, w, – = read and write allowed, but no execute. 11 Same access controls as that defined by M0UM for user mode.
M0UM	Bus master 0 user mode access control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

38.7 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated XBAR bus cycles.

38.7.1 Access evaluation macro

As previously discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in Figure 38-10, the access evaluation macro inputs the XBAR system bus address phase signals (AHB_ap) and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (*hit_b*) and detection of an access protection violation (*error*).

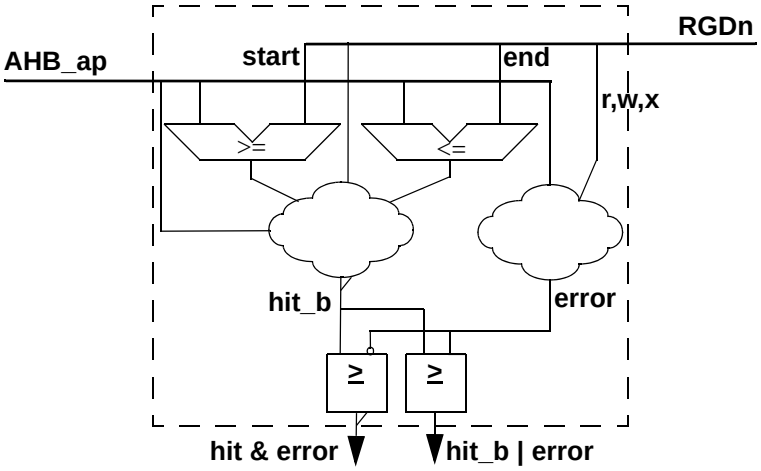


Figure 38-10. MPU access evaluation macro

Figure 38-10 is not intended to be a schematic of the actual access evaluation macro, but rather a generalized block diagram showing the major functions included in this logic block.

38.7.1.1 Access evaluation—hit determination

To determine if the current XBAR reference hits in the given region, two magnitude comparators are used in conjunction with the region's start and end addresses. The boolean equation for this portion of the hit determination is defined as:

region_hit

=((haddr[0:26]>= rgdn.srtaddr[0:26]) & (haddr[0:26] <= rgdn.endaddr[0:26]))

& rgdn.vld ***Eqn. 38-1***

where `haddr[*]` is the current XBAR reference address, `rgdn.srtaddr[*]` and `rgdn.endaddr[*]` are the start and end addresses, and `rgdn.vld` is the valid bit, all from region descriptor n. Recall there are *no hardware checks* to verify that `rgdn.endaddr` $\geq$ `rgdn.srtaddr`,

and it is software's responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to the algebraic comparison of the XBAR reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. Using the `hmaster[*]` number to select the appropriate M*x*PE field from the region descriptor, a process identifier hit term is formed as:

pid_hit

=~rgdn.mxpe

| ((current_pid[0:7] | rgdn.pidmask[0:7])== (rgdn.pid[0:7] | rgdn.pidmask[0:7])) ***Eqn. 38-2***

where the `current_pid[*]` is the selected process identifier from the current bus master, and `rgdn.pid[*]` and `rgdn.pidmask[*]` are the appropriate process identifier fields from the region descriptor n. For XBAR bus masters that do *not* output a process identifier, the MPU forces the `pid_hit` term to be asserted.

As shown in Figure 38-10, the access evaluation macro actually forms the logical complement (*hit_b*) of the combined region_hit and pid_hit boolean equations.

38.7.1.2 Access evaluation – privilege violation determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the XBAR `hmaster[*]` and `hprot[1]` (supervisor/user mode) signals, a set of effective permissions (`eff_rgd[r,w,x]`) is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in Table 38-12.

Table 38-12. Protection violation definition

Description	Inputs					Output
	hwrite	hprot[0]	eff_rgd[r]	eff_rgd[w]	eff_rgd[x]	Protection Violation?
inst fetch read	0	0	—	—	0	yes, no x permission
inst fetch read	0	0	—	—	1	no, access is allowed
data read	0	1	0	—	—	yes, no r permission
data read	0	1	1	—	—	no, access is allowed
data write	1	—	—	0	—	yes, no w permission
data write	1	—	—	1	—	no, access is allowed

The resulting boolean equation for the processor protection violations is:

cpu_protection_violation

= ~hwrite & ~hprot[0] & ~eff_rgdn[x]// ifetch & no x

| ~hwrite & hprot[0] & ~eff_rgdn[r]// data_read & no r

| hwrite& ~eff_rgdn[w] // data_write& no w ***Eqn. 38-3***

The resulting boolean equation for the non-processor protection violations is:

protection_violation

= ~hwrite & ~eff_rgdn[r]// data_read & no r

| hwrite& ~eff_rgdn[w] // data_write& no w ***Eqn. 38-4***

As shown in Figure 38-10, the output of the protection violation logic is the *error* signal, that is, *error* = protection_violation.

The access evaluation macro then uses the *hit_b* and *error* signals to form two outputs. The combined *(hit_b | error)* signal is used to signal the current access is not allowed and *(~hit_b & error)* is used as the input to MPU_EDR*n* (error detail register) in the event of an error.

The critical timing arc through the access evaluation macro involves the delay from the arrival of haddr[*] through the two magnitude comparators and through the *hit_b* generation as shown in Figure 38-11.

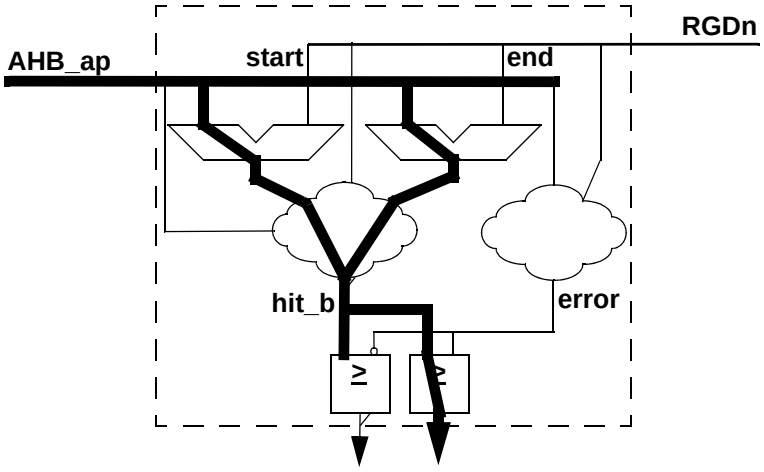


Figure 38-11. Access evaluation macro critical timing path

38.7.2 Functional integration details

For each XBAR slave port being monitored, the MPU performs a **reduction-AND** of all the individual *(hit_b | error)* terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in *any* region descriptor, a protection error is reported.

2. If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
3. If the access hits in multiple (overlapping) regions and *all* regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to *permission granting over access denying* for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see Section 38.9, Application information.

The handling of the XBAR bus cycle with a protection error requires two distinct actions. First, recall the protection error logic resides within the XBAR address phase pipeline stage. In the event of a protection error, the reference must be inhibited from entering the XBAR data phase for the targeted slave device. Stated differently, the reference's address phase must be aborted as viewed by the targeted slave device. This is accomplished by dynamically revising the `htrans[*]` signal sent to the slave device. The `htrans[*]` attribute signals an idle or valid reference (as a non-sequential or sequential access), and this signal is "intercepted" by the MPU. If the access is allowed, then the MPU passes the original `htrans[*]` value to the slave device. However, if a protection error is detected, then the MPU forces `htrans[*]` sent to the slave to the IDLE encoding. This effectively cancels the transaction before it is seen by the slave device.

While forcing `htrans[*] = IDLE` effectively prevents the bus cycle from being transmitted to the slave device, the XBAR transaction has already been "committed" in other portions of the platform, namely in the initiating bus master and the crossbar switch. To properly terminate the bus cycle for these modules, it is necessary to post the standard 2-cycle XBAR error response. For this response, the `hresp[*]` signal is forced to the ERROR encoding for 2 cycles, the first with `hready` negated and the second with `hready` asserted. To perform these termination functions, the MPU "intercepts" the `hready` and `hresp[*]` signals from the slave devices, performs the required logic functions to force the response on the bus cycle with a protection error, and then outputs the adjusted values to the crossbar switch.

The timing diagram of MPU operations is shown in Figure 38-12.

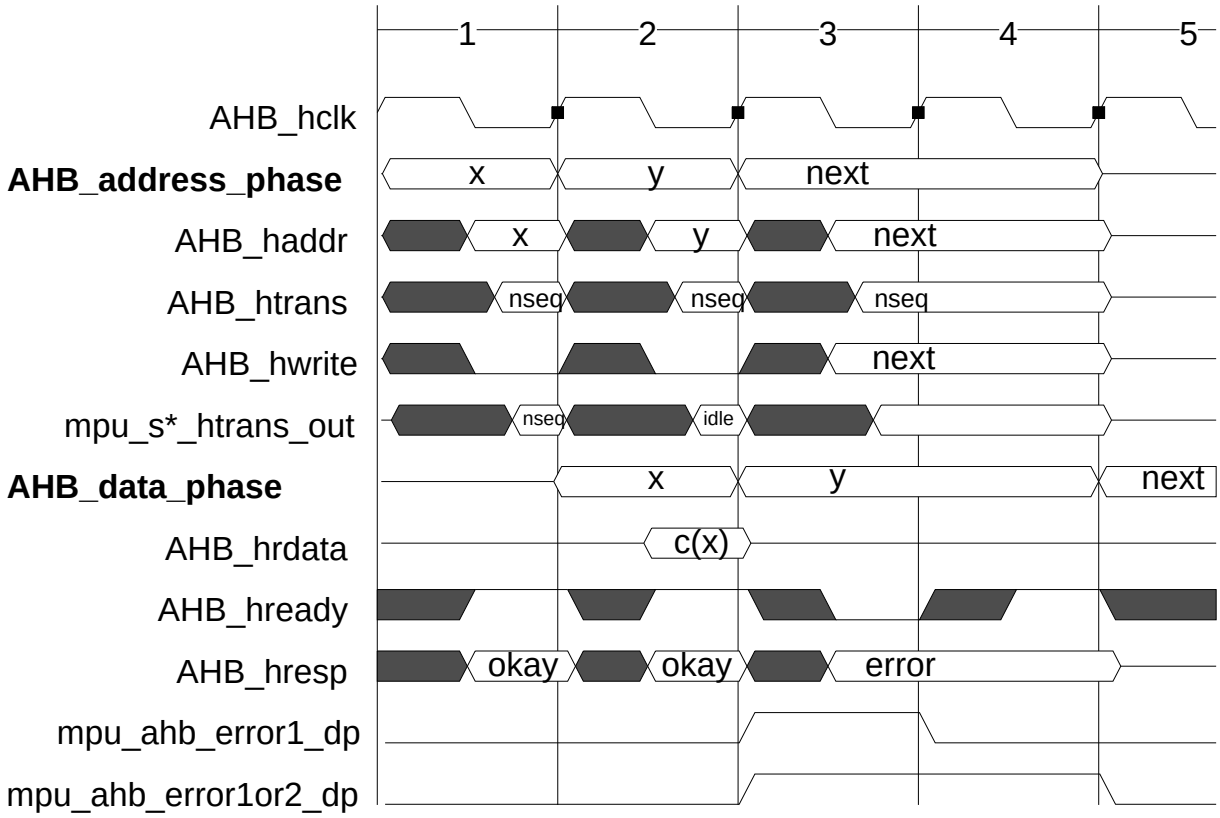


Figure 38-12. MPU and error-terminated XBAR bus cycles

This timing diagram depicts two XBAR non-sequential (nseq) read transactions. The first read is of address x and the second is of address y, where the read of address x is allowed but the read of address y generates an MPU-detected protection error.

For the read @ x, its address phase occurs in cycle 1. This transaction is allowed and the MPU passes the `AHB_htrans[*]` input to the targeted slave device as `mpu_s*_htrans_out[*]`. The data phase for this read occurs in cycle 2 with the accessed operand driven on `AHB_hrdata[*]` with a normal termination (`AHB_hready = 1, AHB_hresp[*] = OKAY`).

The read @ y begins its address phase in cycle 2. This access detects a protection violation, so the transfer is aborted before being sent to the targeted slave device. This can be seen by the MPU forcing the `mpu_s*_htans_out[*] = IDLE` during this cycle. The MPU's error registers (MPU_EAR*n* and MPU_EDR*n*) are loaded at the end of cycle 2. Although aborted to the slave device, the read @ y must still transition into its data phase, which lasts for cycles 3 and 4. During these two cycles, the MPU asserts two internal control states to signal the data phase of an error-terminated transfer. The `mpu_ahb_error1_dp` control state forces `AHB_hready` to be negated during cycle 3, while `mpu_ahb_error1or2_dp state` forces the `AHB_hresp[*] = ERROR` in cycles 3 and 4.

The "next" XBAR transaction is stalled (`AHB_hready = 0`) in its address phase during cycle 3, which finally occurs during cycle 4.

38.8 Initialization information

The reset state of MPU_CESR[VLD] disables the entire module. Recall that while the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGDn) are loaded at system startup, including the setting of the MPU_RGDn.Word3[VLD] bits, *before* MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Recall that if a memory reference does not hit in *any* region descriptor, the attempted access is terminated with an error.

38.9 Application information

In an operational system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD*n*, it would typically be performed using four 32-bit word writes. As discussed in Section 38.6.4.4, MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3), the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed by clearing MPU_RGDn.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAACn) would typically be performed. Recall that writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGDn.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses, respectively, and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.
5. When the MPU detects an access error, the current XBAR bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EAR*n* and MPU_EDR*n* registers. The error-terminated XBAR bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information by reading the MPU_E{A,D}R*n* registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[SPERR].

6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can often reduce the number of descriptors required for a given set of access controls. It is important to note that, in the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (DMA1, a traditional data movement engine transferring data between RAM and peripherals and DMA2, a second engine transferring data to/from the RAM only). Consider the region descriptor assignments in Table 38-13. In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private, and executable memory spaces.

   Of particular interest are the two overlapping spaces: region descriptors 2 & 3 and 3 & 4.

   The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (`rw- | r--`) = (`rw-`) permissions, while CP1 has (`--- | r--`) = (`r--`) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 define another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (`r-- | ---`) = (`r--`) permission, while CP1 has (`rw- | r--`) = (`rw-`) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

   The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the IPS bus is partitioned into two regions: one containing the MPU's programming model accessible only to the two processor cores and the remaining peripheral region accessible to both processors and the traditional DMA1 master.

   This simple example is intended to show one possible application of the capabilities of the Memory Protection Unit in a typical system.

Table 38-13. Overlapping region descriptor example

Region description	RGDn	CP0	CP1	DMA1	DMA2	System space
CP0 code	0	rwx	r--	--	--	Flash memory
CP1 code	1	r--	rwx	--	--	
CP0 data and stack	2	rw-	---	--	--	SRAM
CP0 → CP1 shared data	3	r--	r--	--	--	
CP1 → CP0 shared data						
CP1 data and stack	4	---	rw-	--	--	
Shared DMA data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	--	--	IPS
Peripherals	7	rw-	rw-	rw	--	

This page is intentionally left blank.

Chapter 39
Nexus Development Interface (NDI)

39.1 Parameter values

The parameter values for the MPC5675K are shown in Table 39-1.

Table 39-1. Device parameter values

Parameter	Variable Name	Value
FPM_MDO—Number of MDO pins available in full-port mode	F	16
RPM_MDO—Number of MDO pins available in reduced-port mode	R	12
NUM_AUX—Number of modules on the device sharing the Nexus auxiliary port (not including the NPC)	X	2 in LSM 4 in DPM
NUM_BKPT—Number of breakpoint requests coming into the module from Nexus and non-Nexus sources on the device	B	—
NUM_EVTO—Number of modules that input an $\overline{\text{EVTO}}$	E	4
NUM_JCOMP—Number of JCOMP bits used	J	1
NUM_MSEO—Number of $\overline{\text{MSEO}}$ bits used	M	2
PIN—Part identification number	PIN	0x2C0
MIC—Manufacturer identity code	MIC	0x00E
DC—Design center code	DC	0x2B

39.2 Introduction

Figure 39-1 is a block diagram of the Nexus Port Controller (NPC).

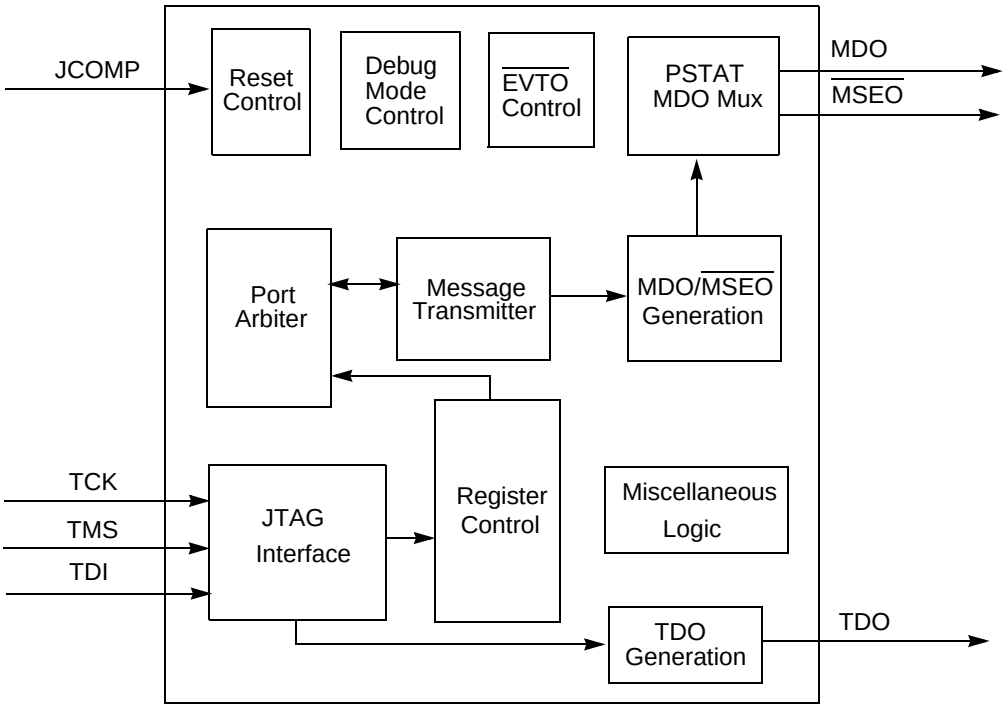


Figure 39-1. Nexus Port Controller (NPC) Block Diagram

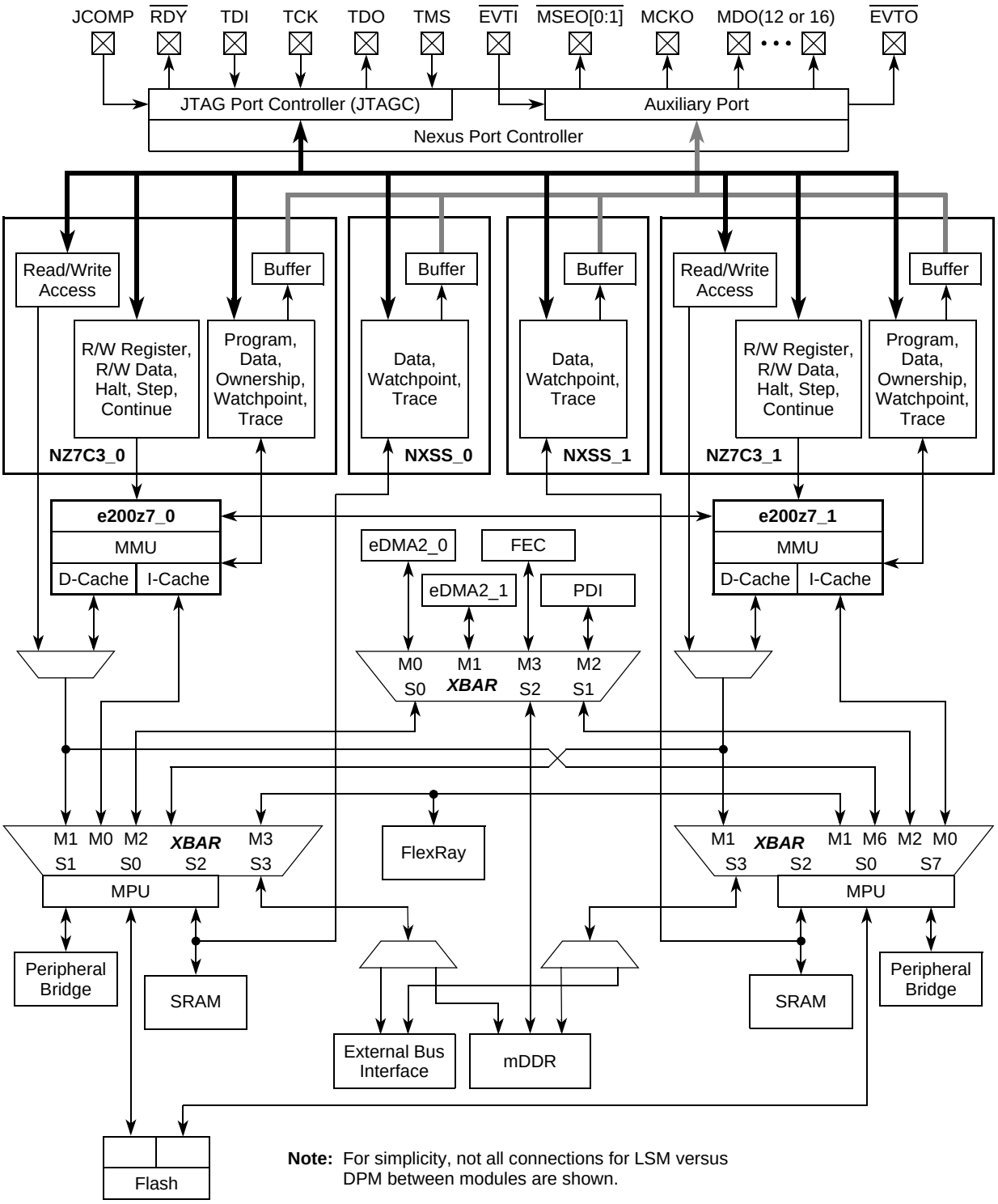


Figure 39-2. Nexus Development Interface

Figure 39-3 shows how the NPC interfaces with the various Nexus modules in the MPC5675K.

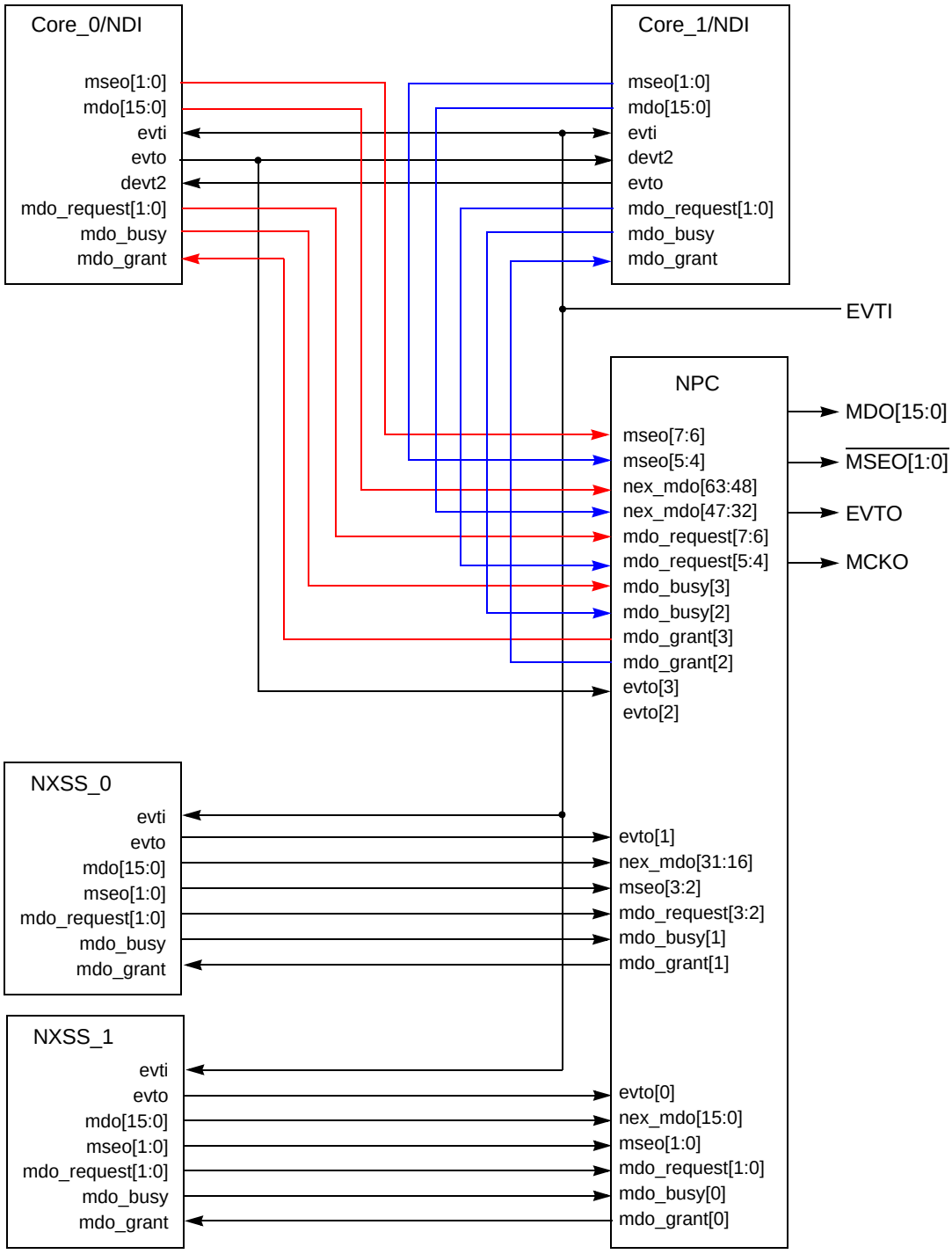


Figure 39-3. NPC/Nexus block interfaces

39.2.1 Overview

On MPC5675K, four modules share their input/output with the development tool based on the IEEE-ISTO 5001-2003 standard. The NPC controls the usage of the input and output port in a manner that allows all

the individual development interface modules to share the port, and appear to the development tool to be a single module. The four modules are two e200z7 core Nexus Interfaces (NZ7C3_0 and NZ7C3_1) and two Nexus Cross-Bar (XBAR) Slave SRAM Data Trace modules (NXSS_0 and NXSS_1). For information on the NZ7C3 Nexus interfaces, see document E200Z760RM, *e200z760n3 Power Architecture® Core Reference Manual*. Information on the NPC and the NXSS modules is included in this section.

39.2.2 Features

The NPC module performs the following functions:

- Controls arbitration for ownership of the Nexus Auxiliary Output Port.
- Nexus Device Identification Register and Messaging.
- Generates MCKO enable and frequency division control signals.
- Controls sharing of $\overline{\text{EVTO}}$.
- Control of the device-wide debug mode.
- Generates asynchronous reset signal for Nexus modules based on JCOMP input, censorship status, and power-on reset status.
- MDO0 indicates when the internal BIST mode has been completed and development tools can access the MCU.
- Provides Nexus support for censorship mode.

39.2.3 Modes of operation

The NPC module uses the JCOMP input and an internal power-on reset indication as primary reset signals. Upon exit of reset, the mode of operation is determined by the Port Configuration Register (PCR) settings.

39.2.3.1 Reset

The NPC module is asynchronously placed in reset when either:

- Power-on reset is asserted.
- JCOMP is not set for Nexus access.
- The device enters censored mode (censorship status signal, nex_disable, is asserted).
- The TAP controller state machine is in the Test-Logic-Reset state.

Holding TMS high for five consecutive rising edges of TCK evokes the Test-Logic-Reset state regardless of the current TAP controller state. Following negation of power-on reset, the NPC remains in reset until the internal self-test concludes. The NPC is unaffected by other sources of reset. While in reset, the following actions occur:

- The TAP controller is forced into the Test-Logic-Reset state.
- The auxiliary output port pins are negated.
- The TDI, TMS, and TCK TAP inputs are ignored (when in power-on reset, censored mode, or when JCOMP is not set for NPC operation only).

- Debug registers default back to their reset values.

39.2.3.2 Disabled-port mode

In disabled-port mode, auxiliary output pin port enable signals are negated, thereby disabling message transmission. Any debug feature that generates messages cannot be used. The primary features available are class 1 features and read/write access to the registers. Class 1 features include the ability to trigger a breakpoint event indication through $\overline{\text{EVTO}}$.

39.2.3.3 Full-port mode

Full-port mode (FPM) is entered by asserting the MCKO_EN and FPM bits in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. In FPM, 16 MDO pins are available.

39.2.3.4 Reduced-port mode

Reduced-port mode (RPM) is entered by asserting the MCKO_EN bit and negating the FPM bit in the PCR. All trace features are enabled or can be enabled by writing the configuration registers via the TAP. In RPM, 12 MDO pins are available.

39.2.3.5 Censored mode

In censored mode, it is not possible to read the contents of internal flash memory externally. The NPC and all other Nexus modules are held in reset when in censored mode, preventing Nexus read/write to memory-mapped resources and the transmission of Nexus trace messages.

If the device is censored, the Nexus blocks are kept disabled to protect flash memory contents from being read.

39.2.3.6 Nexus double data rate (DDR) mode

Nexus double data rate (DDR) mode is enabled by asserting the DDR_EN bit in the PCR. DDR mode is only supported for MCKO_DIV settings of CORE_CLK/2 and CORE_CLK/4. In double data rate mode, message data is updated between each edge (both rising and falling) of MCKO, effectively doubling the message throughput of single data rate mode. The MDO data should be captured by the tool on both edges of the MCKO. Note DDR mode is not supported by the IEE-ISTO 5001-2003 standard.

NOTE

DDR mode is not compliant with the IEEE-ISTO 5001-2003 standard. The standard does not allow this format.

39.3 External signal description

39.3.1 Overview

The NPC pin interface provides for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The NPC pin definition is outlined in Table 39-2.

Table 39-2. NPC signal properties

Name	Port	Function	Reset state
$\overline{\text{EVTO}}$	Auxiliary	Event Out pin	0b1
JCOMP	JTAG	JTAG Compliancy and TAP Sharing Control	—
MDO	Auxiliary	Message Data Out pins	0[1]
$\overline{\text{MSEO}}$	Auxiliary	Message Start/End Out pins	0b11
TCK	JTAG	Test Clock Input	—
TDI	JTAG	Test Data Input	—
TDO	JTAG	Test Data Output	High Z[2]
TMS	JTAG	Test Mode Select Input	—

1 MDO0 indicates when the internal BIST mode has been completed and development tools can access the MCU. MDO0 is driven high until the BIST completes.

2 TDO output buffer enable is negated when the NPC is not in the Shift-IR or Shift-DR states. A weak pullup resistor may be required on TDO to prevent the signal from floating when tools are attached.

39.3.2 Detailed signal descriptions

This section describes each of the signals listed in Table 39-2 in more detail.Note that the JTAG test clock (TCK) input from the pin is not a direct input to the NPC.

39.3.2.1 $\overline{\text{EVTO}}$—Event Out

The Event Out ($\overline{\text{EVTO}}$) pin is an output pin that is asserted upon a breakpoint occurrence to provide breakpoint status indication. The $\overline{\text{EVTO}}$ output of the NPC is generated based on the values of the individual $\overline{\text{EVTO}}$ signals from all Nexus modules that implement the signal.

39.3.2.2 JCOMP—JTAG Compliance

The JCOMP signal provides the ability to share the TAP. The NPC TAP controller is enabled when JCOMP is set to the NPC enable encoding; otherwise, the NPC TAP controller remains in reset.

39.3.2.3 MDO—Message Data Out

The Message Data Out (MDO) pins are output pins used for transmitting trace information (OTM, BTM, DTM, and other messages) to the development tool . The development tool should sample MDO on the rising edge of MCKO (in normal Nexus mode and on both edges of the MCKO if DDR mode is enabled). The width of the MDO bus is in the NPC Port Configuration Register.

39.3.2.4 $\overline{\text{MSEO}}$—Message Start/End Out

The Message Start/End Out ($\overline{\text{MSEO}}$) pin is an output pin that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool should sample $\overline{\text{MSEO}}$ on the rising edge of MCKO (in normal Nexus mode, on both edges of MCKO when DDR mode is configured).

39.3.2.5 TCK—Test Clock Input

The Test Clock Input (TCK) pin synchronizes the test logic and control register access through the JTAG port.

39.3.2.6 TDI—Test Data Input

The Test Data Input (TDI) pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

39.3.2.7 TDO—Nexus Test Data Output

The Test Data Output (TDO) pin transmits serial output for instructions and data. TDO is three-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

39.3.2.8 TMS—Test Mode Select

Test Mode Select Input (TMS) pin sequences the IEEE 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

39.4 Nexus Master IDs

MPC5675K has the following four Nexus modules:

- Nexus Crossbar Slave SRAM Data Trace Monitor 0 (NXSS_0)
- Nexus Crossbar Slave SRAM Data Trace Monitor 1 (NXSS_1)
- Nexus Core_0 (NZ7C3_0)
- Nexus Core_1 (NZ7C3_1)

These modules are described in Table 39-3 through Table 39-5. Each Nexus module can be individually enabled/disabled.

The Nexus Core_0 and 1 modules are attached to the Master XBAR Bus of Core_0 and Core_1, respectively. The Nexus SRC IDs are hard-coded by design.

AHB Master ID Select and Nexus SRC ID are configured via the Nexus Crossbar Slave SRAM Data Trace Monitor register: Data Trace Control (DTC).

DTC[19:16], the AHB Master ID SEL, is used to select which AHB Master should be traced as illustrated in Table 9-1 (Logical master IDs).

Nexus SRC ID = DTC[15:12] — used to place an SRC ID in the Nexus trace messages to identify which Master is associated with the message. The Nexus SRC ID should be set to 0b1101 for NXSS_0 and 0b1110 for NXSS_1.

Table 39-3. Nexus Crossbar Slave SRAM Data Trace Monitor 0 and 1

Master	AHB Master ID	
	LS Mode	DP Mode
Core_0	0	0
Core_1	—	1
DMA_0	2	2
DMA_1	—	6
Nexus Core_0	8	8
Nexus Core_1	—	9
FlexRay	3	—
Ethernet	4	—
PDI	5	—

Table 39-4. Nexus Core_0

Master	Nexus SRC ID	
	LS Mode	DP Mode
Core_0	0	0

Table 39-5. Nexus Core_1

Master	Nexus SRC ID	
	LS Mode	DP Mode
Core_1	0	1

39.5 Register description

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

Table 39-6 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC module does not implement the client select control register because the value does not matter when accessing the registers. Note that the bypass and instruction registers have no index values. These registers are not accessed in the same manner as Nexus client registers. Refer to the individual register descriptions for more detail.

Table 39-6. NPC registers

Index	Register
0	Device ID Register (DID)
127	Port Configuration Register (PCR)

39.5.1 Register descriptions

This section consists of NPC register descriptions.

39.5.1.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

39.5.1.2 Instruction Register

The NPC module uses a 4-bit instruction register as shown in Figure 39-4 (4-Bit Instruction Register). The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and Test-Logic-Reset TAP controller states. Synchronous entry into the Test-Logic-Reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the Test-Logic-Reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

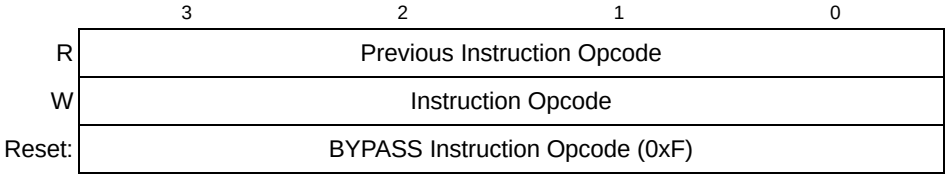


Figure 39-4. 4-Bit Instruction Register

39.5.1.3 Nexus Device ID Register (DID)

The Nexus Device ID Register (DID), shown in Figure 39-5, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Register index: 0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Part Revision Number				Design Center						Part Identification Number					
W																
RESET:	PRN				DC						PIN					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Part Identification Number				Manufacturer Identity Code											I
W																
RESET:	PIN (cont'd.)				0	0	0	0	0	0	0	1	1	1	0	1

Figure 39-5. Nexus Device ID Register

Table 39-7. DID field descriptions

Field	Description
PRN	Part Revision Number. These bits contain the revision number of the part.
DC	Design Center. These bits indicate the device design center.
PIN	Part Identification Number. These bits contain the part number of the device.
MIC	Manufacturer Identity Code. These bits contain the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale Semiconductor, 0xE.
I	IDCODE Register ID. The I bit identifies this register as the device identification register and not the bypass register.

39.5.1.4 Port Configuration Register (PCR)

Register index: 127

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FPM	0	MCKO_EN	MCKO_DIV			EVT_EN	DDR_EN	NEXCFG	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LP_DBG	0	0	0	0	0	LP2_SYN	LP1_SYN	0	0	0	0	0	0	0	PSTAT_EN
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39-6. Port Configuration Register (PCR)

The Port Configuration Register (PCR), shown in Figure 39-6, selects the NPC mode of operation, enables MCKO, and selects the MCKO frequency, and enables or disables MCKO gating. This register should be configured as soon as the NPC is enabled.

The PCR may be rewritten by the debug tool subsequent to the enabling of the NPC for low-power debug support. In this case, the debug tool may set and clear the LP_DBG and LP*n*_SYN bits, but must preserve the original state of the remaining bits in the register.

NOTE

The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Table 39-8. PCR field descriptions

Field	Description
FPM	Full Port Mode. The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 A subset of MDO pins are used to transmit messages. 1 All MDO pins are used to transmit messages.
MCKO_EN	MCKO Enable. This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.
MCKO_DIV	MCKO Division Factor. The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. Table 39-9 shows the meaning of MCKO_DIV Values. In this table, CORE_CLK represents the system clock frequency.
EVT_EN	$\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ Enable. This bit enables the $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port functions. 0 $\overline{\text{EVTO}}$/$\overline{\text{EVTI}}$ port disabled. 1 EVTO/EVTI port enabled.
DDR_EN	Double Data Rate Mode Enable. This bit enables Nexus double data rate (DDR) mode. In DDR mode, message data is updated on both rising and falling edges of MCKO, effectively doubling message throughput. 0 DDR mode disabled. Single Data Rate (SDR) mode, as defined in the IEEE-ISTO 5001-2003 standard. 1 DDR mode enabled. Double Data Rate mode; this mode does not comply with the IEEE-ISTO 5001-2003 standard and may not be supported by all tool vendors.
NEXCFG	Nexus Configuration Select. This bit is not used in MPC5675K. 0 NEXCFG cleared. 1 NEXCFG set.
LP_DBG_EN	Low-Power Debug Enable. This bit enables debug functionality on exit from low-power modes on supported devices. 0 Low-power debug disabled. 1 Low-power debug enabled.
LP*n*_SYN	Low-Power Mode *n* Synchronization. These bits are used to synchronize the entry into low-power modes between the device and debug tool. Supported devices set these bits before a pending entry into low-power mode. After reading the bit as set, the debug tool then clears the bit to acknowledge to the device that it may enter the low-power mode. 0 Low-power mode entry acknowledged. 1 Low-power mode entry pending.
PSTAT_EN	Processor Status Mode Enable[1]. This bit enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable. 0 PSTAT mode disabled. 1 PSTAT mode enabled.

1 PSTAT mode is intended for factory processor debug only. The PSTAT_EN bit should be written to disable PSTAT mode if Nexus messaging is desired. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled.

Table 39-9. MCKO_DIV values

MCKO_DIV[2:0]	MCKO Frequency
0	CORE_CLK[1]
1	CORE_CLK/2[1]
2	CORE_CLK/3
3	CORE_CLK/4
4	Reserved
5	Reserved
6	Reserved

1 The CORE_CLK setting for MCKO frequency should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins. See the maximum MCKO frequency specification in the MPC5675K Datasheet (MPC5675K).

39.6 Functional description

39.6.1 NPC reset configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 39-10 describes the NPC reset configuration options.

Table 39-10. NPC reset configuration options

JCOMP equal to npc_jcomp_plug?	PCR[MCKO_EN]	PCR[FPM] bit	Configuration
No	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-Port Mode
Yes	1	0	Reduced-Port Mode

CAUTION

If the low-power mode debug handshake has been enabled and an external reset or a 'functional' reset occurs while the device is in a low-power mode, the device will not exit reset. Therefore the NPC_PCR[LP_DBG_EN] bit must be cleared to ensure the correct reset sequence.

39.6.2 Auxiliary output port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the Nexus modules and arbitrates for access to the port.

39.6.2.1 Output message protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 39-7 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

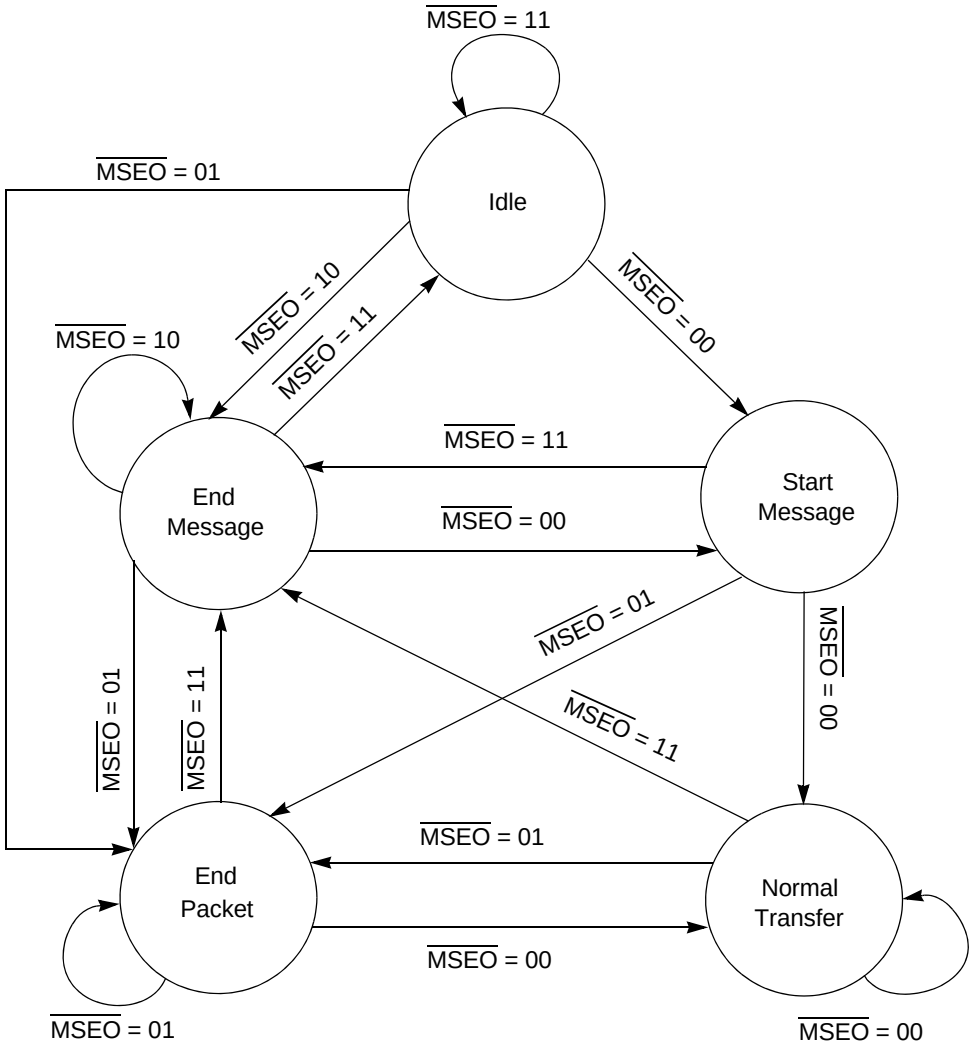


Figure 39-7. $\overline{\text{MSEO}}$ transfers (for 2-bit $\overline{\text{MSEO}}$)

39.6.2.2 Output messages

In addition to sending out messages generated in other Nexus modules, the NPC can also output the device ID message contained in the device ID register and the port replacement output message on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 39-11 describes the device ID and port replacement output messages that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 39-11. NPC output messages

Message name	Min. packet size (bits)	Max. packet size (bits)	Packet type	Packet name	Packet description
Device ID Message	6	6	Fixed	TCODE	Value = 1
	32	32	Fixed	ID	DID register contents

Figure 39-8 shows the various message formats that the pin interface formatter has to encounter. Note that for variable-length fields, the transmitted size of the field is determined from the range of the least significant bit to the most significant non-zero-valued bit (that is, most significant zero-valued bits are not transmitted).

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. size[1] (bits)	Max. size[2] (bits)
Device ID Message	1	Fixed = 32	NA	NA	NA	NA	38	38

Figure 39-8. Message field sizes

1 Minimum information size. The actual number of bits transmitted depends on the number of MDO pins.

2 Maximum information size. The actual number of bits transmitted depends on the number of MDO pins.

The double edges in Figure 39-8 indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

39.6.2.3 Message rules

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field may start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets may start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.

- Within a field, the lowest significant bits are shifted out first. Figure 39-9 shows the transmission sequence of a message that is made up of a TCODE followed by two fields.

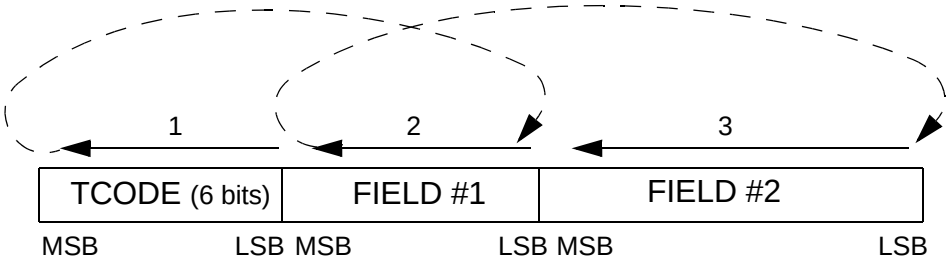


Figure 39-9. Transmission sequence of messages

NOTE

Tools should decode the Data Tag (DQTAG) field of the Nexus Data Acquisition Message (DQM) as a variable length packet instead of a fixed length packet.

39.6.3 IEEE 1149.1-2001 (JTAG) TAP

The NPC module uses the IEEE 1149.1-2001 TAP for accessing registers. Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. There may also be other modules on the MCU that use the TAP and implement a TAP controller. The value of the JCOMP input controls ownership of the port between Nexus and non-Nexus modules sharing the TAP.

Refer to the IEEE 1149.1-2001 specification for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE 1149.1-2001 state machine shown in Figure 39-11. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2001 standard. It is shown in Figure 39-12.

The instructions implemented by the NPC TAP controller are listed in Table 39-12. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is four bits.

Table 39-12. Implemented instructions

Instruction name	Private/ Public	Opcode	Description
NEXUS-ENABLE	public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in Figure 39-10. This applies for the instruction register and all Nexus tool-mapped registers.

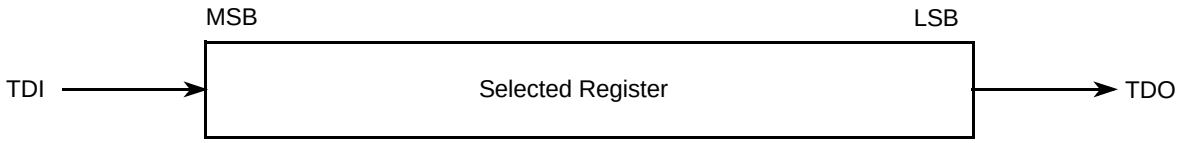
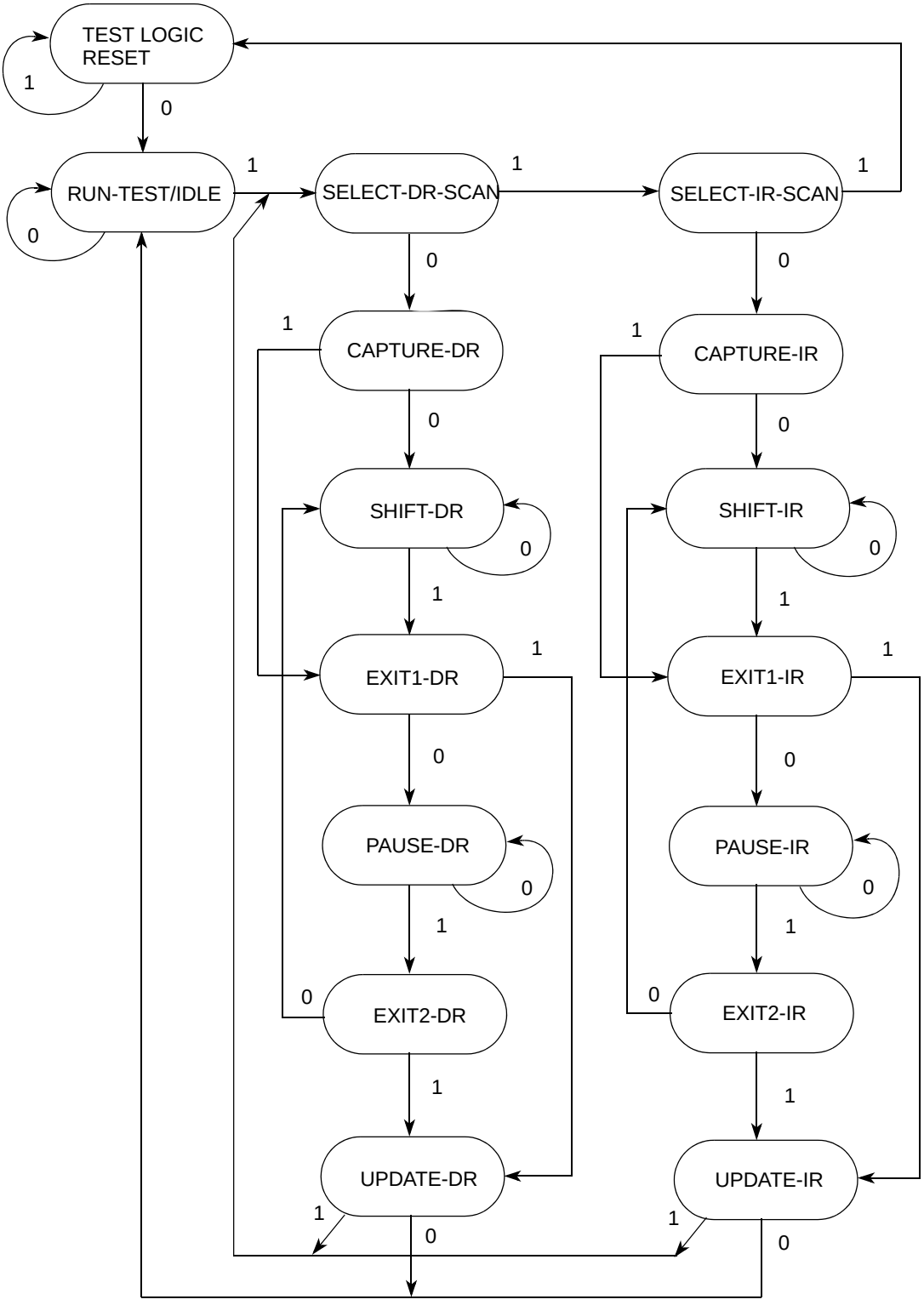


Figure 39-10. Shifting data into register

39.6.3.1 Enabling the NPC TAP controller

Assertion of the power-on reset signal or setting JCOMP to a value other than the NPC enable encoding resets the NPC TAP controller. When not in power-on reset, the NPC TAP controller is enabled by driving JCOMP with the NPC enable value and exiting the Test-Logic-Reset state. Loading the NEXUS-ENABLE instruction then grants access to Nexus debug.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 39-11. IEEE 1149.1-2001 TAP controller state machine

39.6.3.2 Retrieving device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the Nexus Device ID register through the TAP. Transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR, if the NPC is enabled. Transmission of the device identification message serially via TDO is achieved by performing a read of the register contents as described in Section 39.6.3.4, Selecting a Nexus client register.

39.6.3.3 Loading NEXUS-ENABLE instruction

Access to the NPC registers is enabled when the TAP controller instruction register is loaded with the NEXUS-ENABLE instruction. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in Figure 39-12, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. Table 39-13 illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

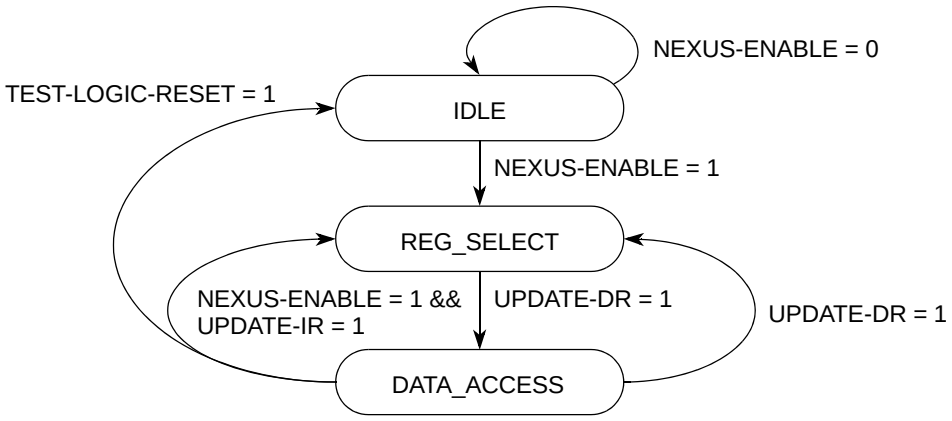


Figure 39-12. NEXUS controller state machine

Table 39-13. Loading NEXUS-ENABLE instruction

Clock	TMS	IEEE 1149.1 state	Nexus State	Description
0	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	IDLE	Transitional state
2	1	SELECT-IR-SCAN	IDLE	Transitional state
3	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
3 TCKS				
12	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
13	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
14	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

39.6.3.4 Selecting a Nexus client register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path. The Nexus controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path enters an 8-bit Nexus command consisting of a read/write control bit in the LSB followed by a 7-bit register address index, as illustrated in Figure 39-13. The read/write control bit is set to 1 for writes and 0 for reads.

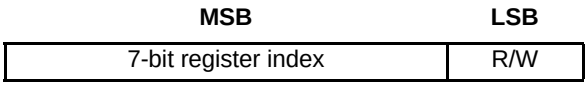


Figure 39-13. IEEE 1149.1 controller command input

The second pass through the SELECT-DR-SCAN path reads or writes the register data by shifting in the data (LSB first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated once the required number of bits have been acquired.

Table 39-14 illustrates a sequence that writes a 32-bit value to a register.

Table 39-14. Write to a 32-Bit Nexus Client register

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
12	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
13	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
14	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
15	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
16	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
48	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.

Table 39-14. Write to a 32-Bit Nexus Client register (continued)

Clock	TMS	IEEE 1149.1 state	Nexus state	Description
49	1	UPDATE-DR	DATA_ACCESS	Value written to register
50	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

39.6.4 Nexus JTAG port sharing

Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers. When Nexus has ownership of the TAP, only the module whose NEXUS-ENABLE instruction is loaded has control of the TAP. This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. If no register is selected as the shift path for a Nexus module, that module acts like a single-bit shift register, or bypass register.

39.6.5 Nexus system integration

The MPC5675K device has four Nexus modules:

- CORE_0
- CORE_1
- NXSS_0
- NXSS_1

The muxing of these Nexus modules to NPC is based on whether the device is run in LSM or DPM mode.

39.6.5.1 Decoupled Parallel Mode (DPM)

In DPM, the NPC port-sharing features are fully utilized, which allows a dynamic arbitration for the Nexus Auxiliary Port from each Nexus implementation on the device.

Figure 39-14 shows the Nexus Auxiliary Port Sharing in DPM.

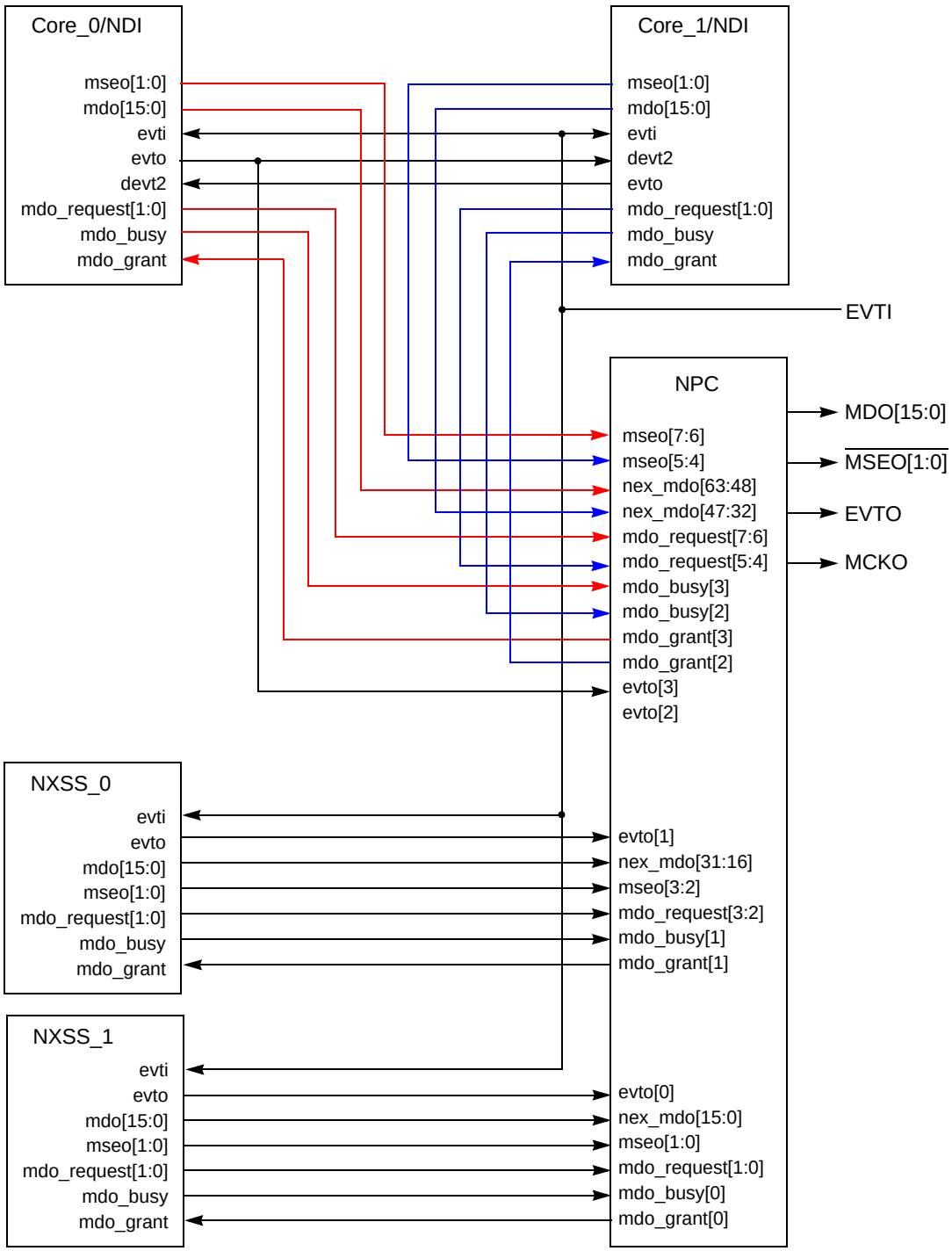


Figure 39-14. Nexus auxiliary port sharing in DPM

39.6.6 Lock Step Mode (LSM)

When operating in LSM, only the outputs of one of the shared modules is actually connected to the NPC since both cores are operating in parallel together and both of the SRAM modules are operating in parallel.

Figure 39-15 shows the Nexus Auxiliary Port Sharing in LSM. The TEST_CTRL[TDO_SEL] bit in the JTAGC selects which outputs are monitored by the NPC.

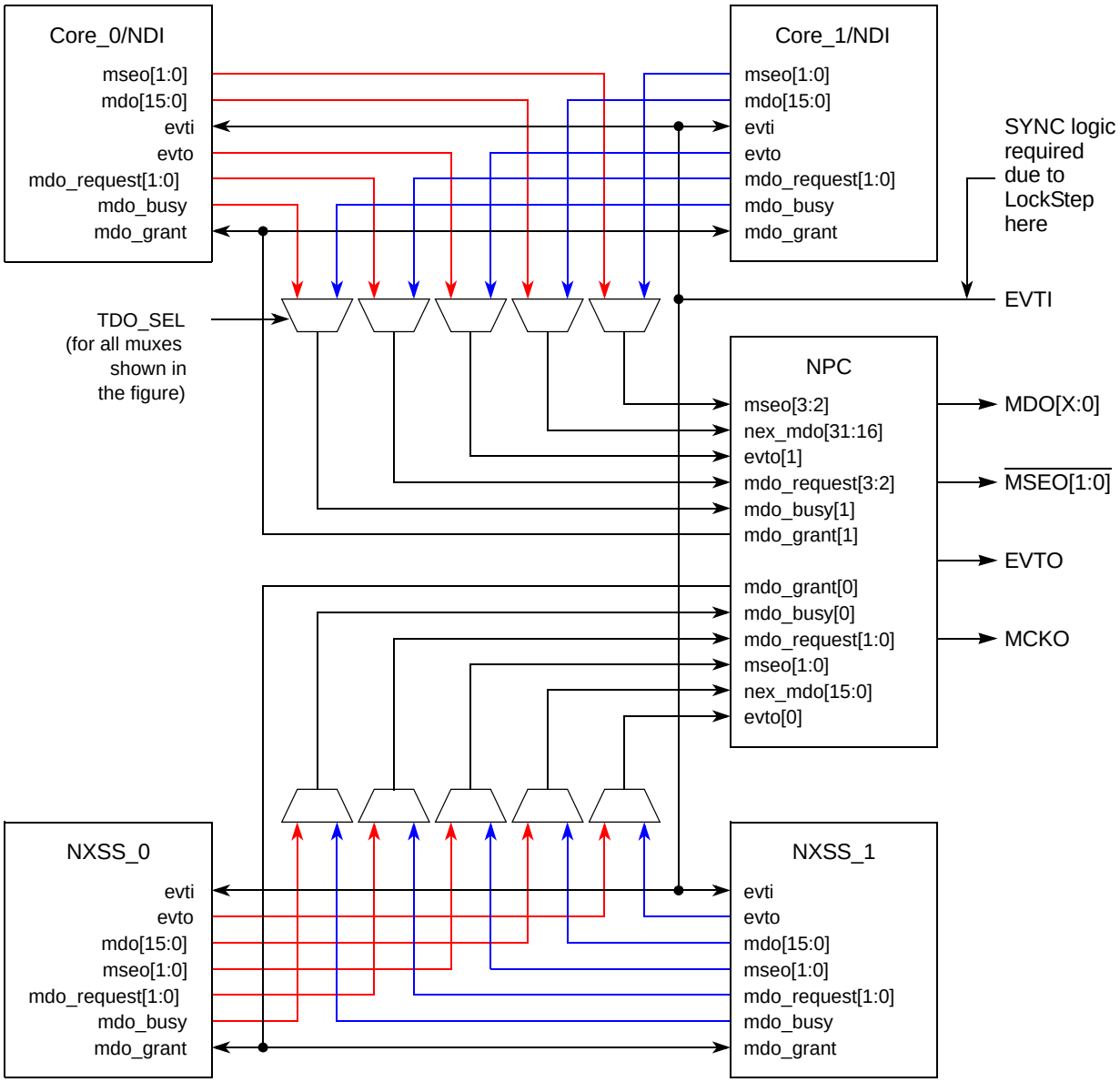


Figure 39-15. Nexus auxiliary port sharing in LSM

39.6.7 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV field in the PCR. Possible operating frequencies include system clock, one-half system clock, and one-quarter system clock speed.

Possible operating frequencies and division factors for various Nexus in MPC5675K are summarized in Section 39.6.7.1, Core Nexus standalone.

39.6.7.1 Core Nexus standalone

When running Nexus in SDR mode (that is, using a single edge of MCKO to launch the data), the following is supported. The table is platform-to-core clock ratio versus MCKO division factor.

Table 39-15. SDR mode

Platform to core clock ratio	DIV1	DIV2	DIV3	DIV4
1 to 1	YES (@40 MHz)	YES (@60 MHz, Pad)	YES (@30 MHz, 33:66 duty)	YES (@22.5 MHz)
1 to 2	YES (@40 MHz)	YES (@60 MHz, Pad)	YES (@60 MHz, 33:66 duty)	YES (@45 MHz)
1 to 3	YES(@40 MHz)	YES (@60 MHz, Pad)	YES (@60 MHz, 33:66 duty)	YES (@45 MHz)

When using DDR mode (that is, using both the edges of MCKO to capture the data), the following is supported:

Table 39-16. DDR mode

Platform to core clock ratio	DIV1	DIV2	DIV3	DIV4
1 to 1	No	YES (@45 MHz)	No	No
1 to 2	No	YES (@45 MHz)	No	Yes (@45 MHz)
1 to 3	No	YES (@45 MHz)	No	No

39.6.7.2 AHB Nexus (Nexus XBAR slave SRAM trace monitor) standalone

When running Nexus in SDR mode:

Table 39-17. SDR mode

Platform to core clock ratio	DIV1	DIV2	DIV3	DIV4
1 to 1	No	YES (@60 MHz, Pad)	YES (@30 MHz, 33:66 duty)	YES (@22.5 MHz)
1 to 2	No	YES (@60 MHz, Pad)	No	YES (@45 MHz)
1 to 3	No	No	No	No

When running Nexus in DDR mode:

Table 39-18. DDR mode

Platform to core clock ratio	DIV1	DIV2	DIV3	DIV4
1 to 1	No	YES (@45 MHz)	No	No
1 to 2	No	No	No	Yes (@45 MHz)
1 to 3	No	No	No	No

NOTE

1) Wherever the limiting factor of frequency is the PAD, it is noted in the table; otherwise, the limiting factor is architectural (for example, the clock ratios themselves) or the design implementation.

2) DDR mode DIV4 provides higher trace throughput than SDR mode DIV2 when the platform to core clock ratio is 2.

If all the Nexus modules are to be used simultaneously, it is necessary to choose a subset from the above tables that is common to all. For example, if using 1:1 platform to core frequency, DDR mode DIV2 is supported by all the Nexus modules.

39.6.8 $\overline{\text{EVTO}}$ sharing

The NPC module controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. After receiving a single clock period of asserted $\overline{\text{EVTO}}$ from any Nexus client, the NPC latches the result and drives $\overline{\text{EVTO}}$ for one MCKO period on the following clock. When there is no active MCKO, such as in disabled mode, the NPC drives $\overline{\text{EVTO}}$ for two system clock periods. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

39.6.9 Nexus reset control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus modules. If JCOMP is negated (low), an internal reset is asserted, indicating that all Nexus modules should be held in reset. Internal Nexus reset is also asserted when in power-on reset, or if the device is in a censored mode.

39.6.10 Nexus ready status

Following a power-on reset, MDO[0] can be used to indicate to the debugger whether the NPC is ready for usage by the development tool. It is driven low once the chip is ready for development or tool commands. While power-on reset or LBIST is in progress, MDO0 is driven high after the initial POR until the LBIST completes.

39.7 Initialization/application information

39.7.1 Accessing NPC tool-mapped registers

To initialize the TAP for Nexus register accesses, the following sequence is required:

1. Enable the Nexus TAP controller.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. Note that the prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2003 standard for more detail.

39.8 Crossbar slave port data trace module (NXSS)

The MPC5675K provides two Crossbar Slave Port Data Trace Modules (NXSS). The Crossbar Slave Port Data Trace Module 0 (NXSS_0) and the Crossbar Slave Port Data Trace Module 1 (NXSS_1) are compliant with the Class 3 defined data trace feature of the IEEE-ISTO 5001-2003 standard. The NXSS_0 can be programmed to trace data accesses to the System SRAM0 and the NXSS_1 can be programmed to trace data accesses to the System SRAM1.

NOTE

The auxiliary port and its signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[15:0] and others, are referenced. The device NPC module arbitrates the access of the single auxiliary port. The functions of the NXSS_0 and NXSS_1 modules are described without the interaction of the NPC, as if Nexus has a dedicated auxiliary port, to simplify the description.

39.8.1 NXSS block diagram

Figure 39-16 shows a block diagram of the NXSS. Nexus full-port mode provides 16 MDOs and Nexus reduced-port mode provides 12 MDOs.

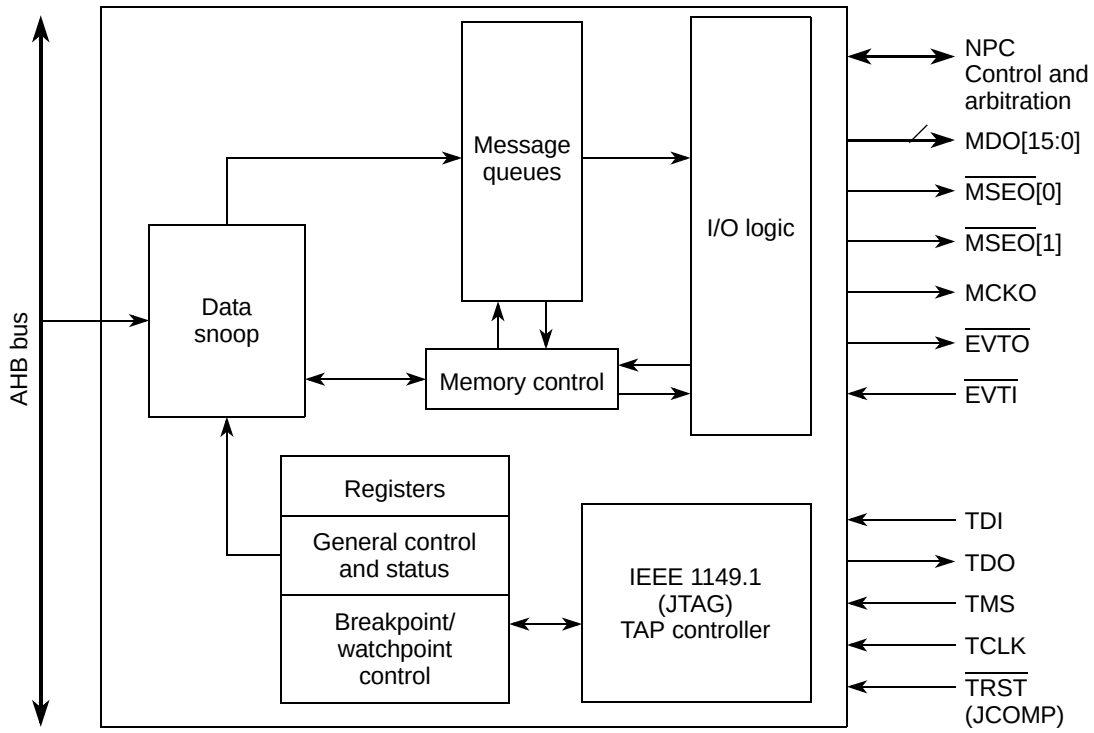


Figure 39-16. NXSS block diagram

39.8.2 Features

Features include the following:

- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to system SRAM0 and system SRAM1.
- Watchpoint messaging via the auxiliary pins
- Watchpoint trigger enable of data trace messaging (DTM)
- Registers for data trace, watchpoint generation, and watchpoint trigger
- All features controllable and configurable via the JTAG port

39.8.3 External signal description

39.8.3.1 Rules for output messages

The NXSS module observes the same rules for output messages as the NPC.

39.8.3.2 Auxiliary port arbitration

The NXSS_0 and NXSS_1 modules arbitrate for the shared Nexus port. This arbitration is handled by the NPC and is based on prioritized requests from the NXSS_0, NXSS_1, and the other Nexus clients sharing the port.

39.8.4 NXSS_0 and NXSS_1 programmer's model

This section describes the programmer's model. Nexus registers are accessed using the JTAG port in compliance with IEEE 1149.1.

Table 39-19. NXSS_0 and NXSS_1 register map

Register	Nexus access opcode	Read/Write	Read address	Write address
Client Select Control (CSC) [1]	0x1	R	0x02	—
Port Configuration Register (PCR) [1]	Refer to NPC	R/W	—	—
Development Control 1 (DC1_*n*)	0x2	R/W	0x04	0x05
Development Control 2 (DC2_*n*)	0x3	R/W	0x05	0x06
Watchpoint Trigger (WT_*n*)	0xB	R/W	0x16	0x17
Data Trace Control (DTC_*n*)	0xD	R/W	0x1A	0x1B
Data Trace Start Address 1 (DTSA1_*n*)	0xE	R/W	0x1C	0x1D
Data Trace Start Address 2 (DTSA2_*n*)	0xF	R/W	0x1E	0x1F
Data Trace End Address 1 (DTEA1_*n*)	0x12	R/W	0x24	0x25
Data Trace End Address 2 (DTEA2_*n*)	0x13	R/W	0x26	0x27
Breakpoint/Watchpoint Control Register 1 (BWC1_*n*)	0x16	R/W	0x2C	0x2D
Breakpoint/Watchpoint Control Register 2 (BWC2_*n*)	0x17	R/W	0x2E	0x2F
Breakpoint/Watchpoint Address Register 1 (BWA1_*n*)	0x1E	R/W	0x3C	0x3D
Breakpoint/Watchpoint Address Register 2 (BWA2_*n*)	0x1F	R/W	0x3E	0x3F
Reserved	0x20–0x3F	—	0x40–0x7E	0x41–0x7F

[1] The CSC and PCR registers are shown in this table as part of the Nexus programmer's model. They are only present at the top level Nexus3 controller (NPC), not in the NXSS_0 or NXSS_1 module. The device's CSC register is readable through Nexus3; the PCR is shown for reference only.

39.8.4.1 Development Control Registers (DC1 and DC2)

The Development Control Registers (DC1 and DC2) control the basic development features of the NXSS_0 and NXSS_1 modules.

Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	0	0	EOC		0	0	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	C	0	0	0	0	0	0	0	0	0	EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39-17. Development Control Register 1 (DC1)

Table 39-20. DC1 field description

Field	Description
31 OPC[1]	Output port mode control. 0 Reduced port mode configuration 1 Full port mode configuration
28–27 EOC	$\overline{\text{EVTO}}$ control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoint (internal or external) 01 $\overline{\text{EVTO}}$ upon entry into system-level debug mode 1X Reserved
24 WEN	Watchpoint trace enable. 0 Watchpoint messaging disabled 1 Watchpoint messaging enabled.
4–3 EIC	$\overline{\text{EVTI}}$ control. 00 $\overline{\text{EVTI}}$ for synchronization (Data Trace) 01 Reserved 10 $\overline{\text{EVTI}}$ disabled for this module 11 Reserved
2–0 TM	Trace mode. 000 No Trace 1XX Reserved X1X Data trace enabled XX1 Reserved

1 The output port mode control bit (OPC) is shown for clarity. This function is controlled globally by the NPC port control register (PCR).

Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EWC								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39-18. Development Control Register 2 (DC2)

Table 39-21. DC2 field description

Field	Description
31–24 EWC[1]	$\overline{\text{EVTO}}$ Watchpoint Configuration 00000000 = No watchpoints trigger $\overline{\text{EVTO}}$ 1XXXXXXX = Reserved X1XXXXXX = Reserved XX1XXXXX = Reserved XXX1XXXX = Reserved XXXX1XXX = Internal watchpoint #1 triggers $\overline{\text{EVTO}}$ XXXXX1XX = Internal watchpoint #2 triggers $\overline{\text{EVTO}}$ XXXXXX1X = Reserved XXXXXXX1 = Reserved

1 The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

39.8.4.2 Watchpoint Trigger Register (WT)

The Watchpoint Trigger Register (WT) allows the watchpoints defined internally to the NXSS_0 and NXSS_1 modules to trigger actions. These watchpoints can control data trace enable and disable. The WT bits can be used to produce an address related window for triggering trace messages.

Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	DTS		DTE				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39-19. Watchpoint Trigger Register (WT)

Table 39-22. WT field description

Field	Description
25–23 DTS	Data trace start control. 000 Trigger disabled 001–100 Reserved 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Reserved
22–20 DTE	Data trace end control 000 Trigger disabled 001–100 Reserved 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Reserved

NOTE

The WT bits only enable data trace if the TM bits within the Development Control register (DC) have not already been set to enable data trace.

39.8.4.3 Data Trace Control Register (DTC)

The Data Trace Control Register (DTC) controls whether DTM messages are restricted to reads, writes, or both for a user-programmable address range. There are two data trace channels controlled by the DTC for the NXSS_0 and NXSS_1 modules.

Access: Read/Write

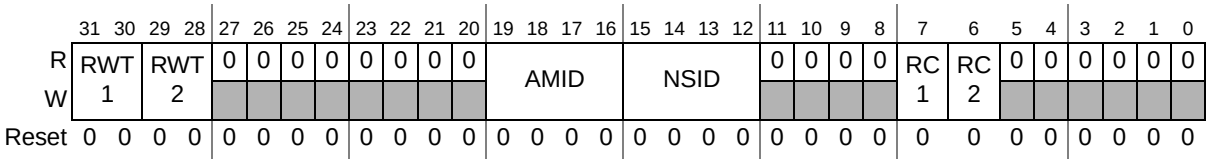


Figure 39-20. Data Trace Control Register (DTC)

Table 39-23. DTC field description

Field	Description
31–30 RWT1	Read/write trace 1 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace
29–28 RWT2	Read/write trace 2 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace
19–16 AMID	AHB Master ID Select This field selects which crossbar master has its accesses to the SRAM traced. Only one master can have its accesses to the SRAM traced at a time. Refer to Table 9-1 (Logical master IDs) for details.
15–12 NSID	Nexus Source ID This field defines the Nexus Source ID that will be used in the Nexus trace messages. This should be set by tools to: LSM decoding 0000–1100 Reserved 1101 NXSS_0 / NXSS_1 1110–-1111 Reserved DPM decoding 0000-1100 Reserved 1101 NXSS_0 1110 NXSS_1 1111 Reserved **Note:** The tool is free to set this bitfield to any value.
7 RC1	Range control 1 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)
6 RC2	Range control 2 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)

39.8.4.4 Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2)

The Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2) define the start addresses for each trace channel.

Access: Read/Write

	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R W	DATA TRACE START ADDRESS							
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 39-21. Data Trace Start Address Registers (DTSA1, DTSA2)

39.8.4.5 Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2)

The Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2) define the end addresses for each trace channel.

Access: Read/Write

	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R W	DATA TRACE END ADDRESS							
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 39-22. Data Trace End Address Registers (DTEA1, DTEA2)

Table 39-24 illustrates the range that is selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 39-24. Data Trace Address range options

Programmed values	Range Control bit value	Range selected
DTSA ≤ DTEA	0	DTSA→ ←DTEA
DTSA ≤ DTEA	1	← DTSA DTEA →
DTSA > DTEA	—	Invalid range, no trace

NOTE

DTSA must be less than or equal to DTEA to enable correct data write/read traces. When the range control bit is 0 (internal range), accesses to DTSA and DTEA addresses are traced. When the range control bit is 1 (external range), accesses to DTSA and DTEA are not traced.

39.8.4.6 Breakpoint/Watchpoint Control Register 1 (BWC1)

The Breakpoint / Watchpoint Control Register 1 (BWC1) controls attributes for generation of NXSS_0 and NXSS_1 watchpoint number 1.

Access: Read/Write

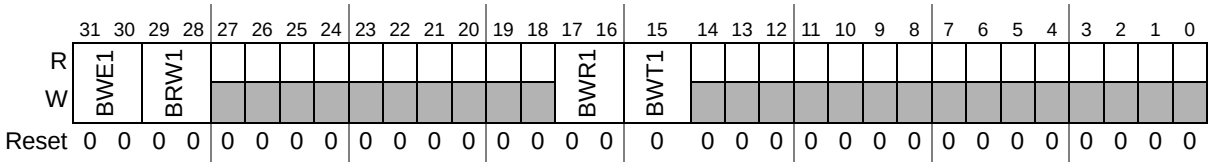


Figure 39-23. Break / Watchpoint Control Register 1 (BWC1)

Table 39-25. BWC1 field description

Field	Description
31–30 BWE1	Breakpoint/watchpoint #1 enable 00 Internal Nexus watchpoint #1 disabled 01–10 Reserved 11 Internal Nexus watchpoint #1 enabled
29–28 BRW1	Breakpoint/watchpoint #1 read/write select 00 Watchpoint #1 hit on read accesses 01 Watchpoint #1 hit on write accesses 10 Watchpoint #1 on read or write accesses 11 Reserved
17–16 BWR1	Breakpoint/watchpoint #1 register compare 00 No register compare (same as BWC1[31:30] = 2'b00) 01 Reserved 10 Compare with BWA1 value 11 Reserved
15 BWT1	Breakpoint/watchpoint #1 type 0 Reserved 1 Watchpoint #1 on data accesses

39.8.4.7 Breakpoint/Watchpoint Control Register 2 (BWC2)

The Breakpoint / Watchpoint Control Register 2 (BWC2) controls attributes for generation of NXSS_0 and NXSS_1 watchpoint number 2.

Access: Read/Write

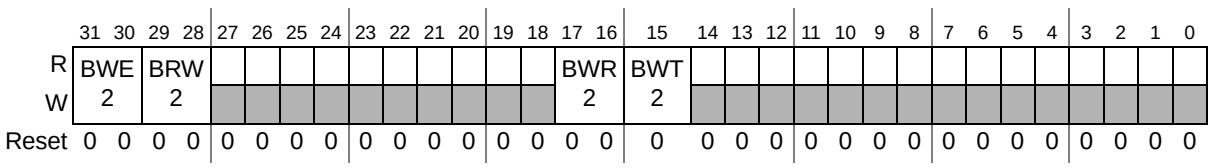


Figure 39-24. Break / Watchpoint Control Register 2 (BWC2)

Table 39-26. BWC2 field description

Field	Description
31–30 BWE2	Breakpoint/watchpoint #2 enable 00 Internal Nexus watchpoint #2 disabled 01–10 Reserved 11 Internal Nexus watchpoint #2 enabled
29–28 BRW2	Breakpoint/watchpoint #2 read/write select 00 Watchpoint #2 hit on read accesses 01 Watchpoint #2 hit on write accesses 10 Watchpoint #2 on read or write accesses 11 Reserved
17–16 BWR2	Breakpoint/watchpoint #2 register compare 00 No register compare (same as BWC1[31:30] = 2'b00) 01 Reserved 10 Compare with BWA2 value 11 Reserved
15 BWT2	Breakpoint/watchpoint #2 Type 0 Reserved 1 Watchpoint #2 on data accesses

39.8.4.8 Breakpoint/Watchpoint Address Registers 1 and 2 (BWA1 and BWA2)

The breakpoint/watchpoint address registers are compared with bus addresses to generate internal watchpoints.

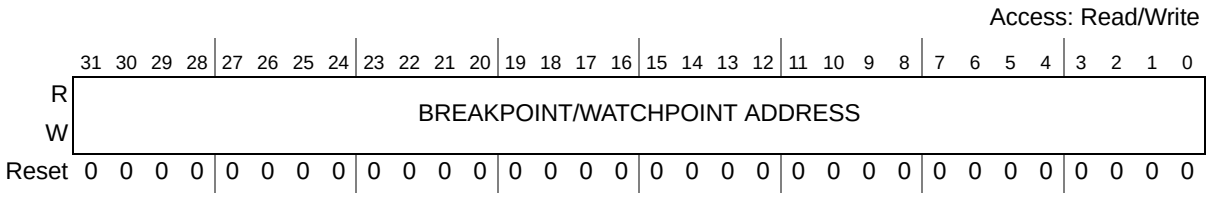


Figure 39-25. Breakpoint/Watchpoint Address Registers (BWA1, BWA2)

39.8.4.9 Unimplemented registers

Unimplemented registers are those with client select and index value combinations other than those listed in Table 39-19. For unimplemented registers, the NXSS_0 and NXSS_1 modules drive TDO to 0 during the "SHIFT-DR" state. It also transmits an error message with the invalid access opcode encoding.

39.8.5 Functional description

39.8.5.1 TCODEs supported by NXSS_0 and NXSS_1

The NXSS_0 and NXSS_1 pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The NXSS_0 and NXSS_1 modules support the public TCODEs as shown in Table 39-27.

Table 39-27. Public TCODEs supported

Message name	Packet size bits		Packet name	Packet type	Packet description
	Min.	Max.			
Data Trace/Data Write Message	6	6	TCODE	Fixed	TCODE number = 5
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 39-29)
	1	32	U-ADDR	Variable	Unique portion of the data write value
	1	64	DATA	Variable	Data write value
Data Trace/Data Read Message	6	6	TCODE	Fixed	TCODE number = 6
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 39-29)
	1	32	U-ADDR	Variable	Unique portion of the data read value
	1	64	DATA	Variable	Data read value
Error Message	6	6	TCODE	Fixed	TCODE number = 8
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	5	5	ECODE	Fixed	Error code (refer to Table 39-28)
Data Trace/Data Write Message with synchronization	6	6	TCODE	Fixed	TCODE number = 13 (0xD)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 39-29)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data write value
Data Trace/Data Read Message with synchronization	6	6	TCODE	Fixed	TCODE number = 14 (0xE)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 39-29)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data read valued
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0xF)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	4	4	WPHIT	Fixed	Number indicating watchpoint sources

Table 39-28. Error Code (ECODE) encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Reserved
00001	Reserved
00010	Data Trace overrun
00011	Reserved
00100	Reserved
00101	Invalid access opcode (Nexus Register unimplemented)
00110	Watchpoint overrun
00111	Reserved
01000	Data Trace and Watchpoint overrun
01001–11111	Reserved

Table 39-29. Data Trace Size (DSZ) encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	Byte
001	Halfword (two bytes)
010	Word (four bytes)
011	Doubleword (eight bytes)
100–111	Reserved

39.8.5.2 Data Trace

This section deals with the data trace mechanism supported by the NXSS_0 and NXSS_1 modules. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM).

39.8.5.3 Data Trace Messaging (DTM)

NXSS_0 and NXSS_1 data trace messaging is accomplished by snooping the NXSS_0 and NXSS_1 data bus, and storing the information for qualifying accesses (based on enabled features and matching AHB Master ID and target addresses). The NXSS module traces all data accesses that meet the selected range and attributes unless the volume of trace information is too great for the trace port bandwidth.

39.8.5.4 DTM message formats

The NXSS module supports five types of DTM Messages — data write, data read, data write synchronization, data read synchronization, and error messages.

39.8.5.4.1 Data Write and Data Read messages

The data write and data read messages contain the data write/read value and the address of the write/read access, relative to the previous data trace message. Data write message and data read message information is messaged out in the following format:

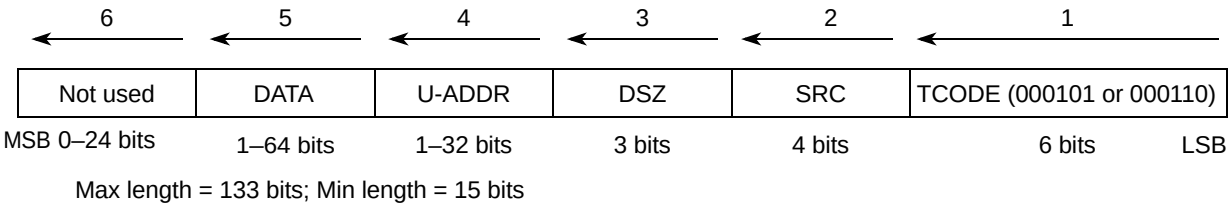


Figure 39-26. Data write/read message format

39.8.5.4.2 DTM Overflow Error messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a data trace message attempts to enter the queue while it is being emptied, the error message incorporates the data trace only error encoding (00010). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

Error information is messaged out in the following format:

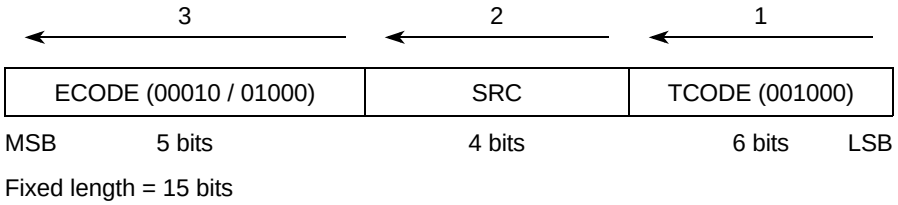


Figure 39-27. Error message format

39.8.5.4.3 Data Trace Synchronization messages

A data trace write/read with synchronization message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (refer to Table 39-30):

- Initial data trace message upon exit from system reset or whenever data trace is enabled is a synchronization message.
- Upon returning from debug mode, the first data trace message is a synchronization message.
- After a queue overrun occurrence (can be caused by any trace message), the first data trace message is a synchronization message.
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred.
- Upon assertion of the Event In ($\overline{\text{EVTI}}$) pin, the first data trace message is a synchronization message if the EIC bits of the DC register have enabled this feature.

- Upon data trace write/read after the previous DTM message was lost due to an attempted access to a secure memory location.
- Upon data trace write/read after the previous DTM message was lost due to a collision entering the FIFO between the DTM message and any of the following: error message, or watchpoint message.

Data trace synchronization messages provide the full address (without leading zeros) and ensure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent DTMs, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read with synchronization messages is as follows:

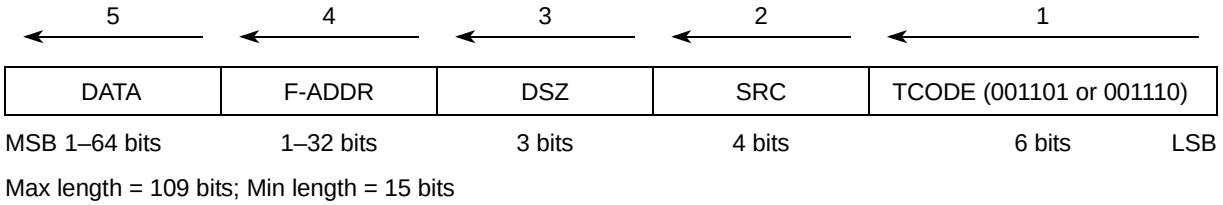


Figure 39-28. Data Write/Read with synchronization message format

Exception conditions that result in data trace synchronization are summarized in Table 39-30.

Table 39-30. Data Trace exception summary

Exception condition	Exception handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NXSS_0 and NXSS_1 module are reset. If data trace is enabled, the first data trace message is a data write/read with synchronization message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Debug	Upon exit from debug mode the next data trace message is converted to a data write/read with synchronization message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates the types of messages that attempted to be queued while the FIFO was being emptied. The next DTM message in the queue is a data write/read with synchronization message.
Periodic Data Trace Synchronization	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read with synchronization message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a data trace write/read with synchronization message upon the next data write/read (if data trace is enabled and the EIC bits of the DC register have enabled this feature).

Table 39-30. Data Trace exception summary (continued)

Exception condition	Exception handling
Attempted Access to Secure Memory	Any attempted read or write to secure memory locations temporarily disable data trace and cause the corresponding DTM to be lost. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: Error → WPM → DTM. A DTM message which attempts to enter the queue at the same time as an error message, or watchpoint message is lost. A subsequent read/write queues a data trace read/write with sync. message.

39.8.5.5 DTM Operation

39.8.5.5.1 Enabling Data Trace Messaging

Data trace messaging can be enabled in one of two ways.

- Setting the DC1[TM] field to enable data trace
- Using the WT[DTS] field to enable data trace on watchpoint hits

39.8.5.5.2 DTM queueing

NXSS_0 and NXSS_1 implement a queue for queuing all messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages must be queued at the same time, watchpoint messages have the highest priority (WPM → DTM).

39.8.5.5.3 Relative addressing

The relative address feature is compliant with IEEE-ISTO Nexus 5001-2003 and is designed to reduce the number of bits transmitted for addresses of data trace messages.

39.8.5.5.4 Data Trace windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All read/write accesses by the selected AHB Master ID (See Section 39.8.4.3, Data Trace Control Register (DTC) for more details) that fall inside or outside these address ranges, as programmed, are candidates to be traced.

39.8.6 Watchpoint support

The NXSS_0 and NXSS_1 module provides watchpoint messaging via the auxiliary pins, as defined by IEEE-ISTO 5001-2003.

Watchpoint messages can be generated using the NXSS_0- and NXSS_1-defined internal watchpoints.

39.8.6.1 Watchpoint messaging

Enabling watchpoint messaging is accomplished by setting the watchpoint messaging enable bit, DC1[WEN]. Using the BWC1 and BWC2 registers, two independently controlled internal watchpoints can be initialized. When the selected AHB Master ID address matches on BWA1 or BWA2, a watchpoint message is transmitted.

The Nexus module provides watchpoint messaging using the TCODE. When either of the two possible watchpoint sources asserts, a message is sent to the queue to be messaged out. This message indicates the watchpoint number.

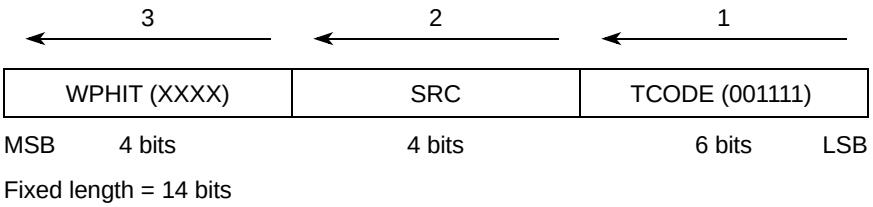


Figure 39-29. Watchpoint message format

Table 39-31. Watchpoint Source description (WPHIT field descriptor)

Watchpoint source (4 bits)	Watchpoint Description
XXX1	Reserved
XX1X	Reserved
X1XX	Internal watchpoint #1 (BWA1 match)
1XXX	Internal watchpoint #2 (BWA2 match)

39.8.6.2 Watchpoint error message

An error message occurs when a new message cannot be queued due to the message queue being full. Messages are discarded until the queue has been completely emptied. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the message queue was being emptied.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If a data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

Error information is messaged out in the following format (refer to Figure 39-30).

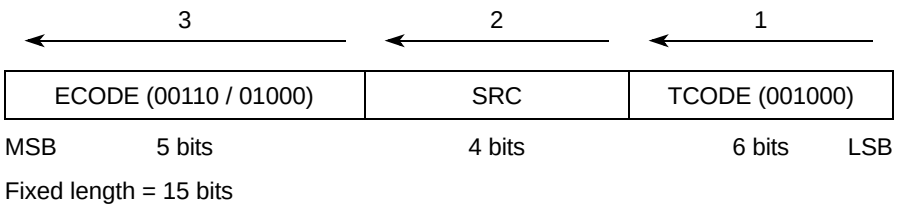


Figure 39-30. Error message format

Chapter 40
Oscillators

This chapter describes the following modules on this device:

- Internal RC oscillator (IRCOSC)
- External oscillator (XOSC)

40.1 XOSC external oscillator

The external crystal oscillator (XOSC) works in the range of 4–40 MHz.

The XOSC digital interface contains control and status registers accessible to applications.

The main features are:

- Oscillator clock available interrupt
- Oscillator bypass mode

40.1.1 Functional description

The crystal oscillator circuit includes an internal oscillator driver and an external crystal circuit. It provides an output clock that can be provided to PLL or used as a reference clock to specific modules depending on system needs.

The crystal oscillator can be controlled by the MC_ME module. The XOSCON bit in the ME_<*mode*>_MC registers controls the power-down of the oscillator based on the current device mode, while the ME_GS[S_XOSC] bit (see Section 35.3.2.1, Global Status Register (ME_GS)) provides the oscillator clock available status.

After system reset, the oscillator is powered down, and software must enable the oscillator when required. Whenever the crystal oscillator is switched on from the off state, the OSCCNT counter starts. When it reaches the value of EOCV × 512, the oscillator clock becomes available to the system. The interrupt pending bit OSC_CTL[I_OSC] is also set. An interrupt is generated if the interrupt mask bit M_OSC is set.

The oscillator circuit can be bypassed by setting OSC_CTL[OSCBYP]. This bit can only be set by the software. System reset is needed to clear this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the XTALOUT pin and the oscillator status is forced to 1. The bypass configuration is independent of the power-down mode of the oscillator.

Table 40-1 shows the truth table of different configurations of the oscillator.

Table 40-1. Truth table of the crystal oscillator

ENABLE	BYP	XTALIN	XTALOUT	CK_OSCM	OSC mode
0	0	No crystal, High-impedance	No crystal, High-impedance	0	Power down
x	1	x	Ext clock	XTALOUT	Bypass, OSC disabled

Table 40-1. Truth table of the crystal oscillator

ENABLE	BYP	XTALIN	XTALOUT	CK_OSCM	OSC mode
1	0	Crystal	Crystal	XTALOUT	Normal, OSC enabled
		Ground	Ext clock	XTALOUT	Normal, OSC enabled

40.2 IRCOSC 16 MHz internal RC oscillator

The RC oscillator has a nominal frequency of 16 MHz. The major features of the oscillator are:

- Glitch-free oscillation
- 6-bit trimming

After power-on reset, the 16 MHz IRC is trimmed to the factory settings that are provided by the flash memory options. The RCTRIM register will show a reset value that is not representative of the factory trim settings. These bits should only be written to if changing from the default value is desired.

40.2.1 Register description

The Oscillator module registers are writable only in supervisor mode.

Table 40-2. Oscillators memory map

Offset from MC_CGM_BASE (0xC3FE_0000)	Register	Access[1]	Reset value	Location
0x0000	Crystal Oscillator Control Register (OSC_CTL)	R/W	0x0080_0000	on page 1465
0x0004–0x005F	Reserved			
0x0060	RC Control Register (RC_CTL)	R/W	0x0000_0000	on page 1466
0x0064–0xFFFF	Reserved			

[1] In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

40.2.1.1 Crystal Oscillator Control Register (OSC_CTL)

Address: Base + 0x0000 Access: Supervisor read/write, User read-only

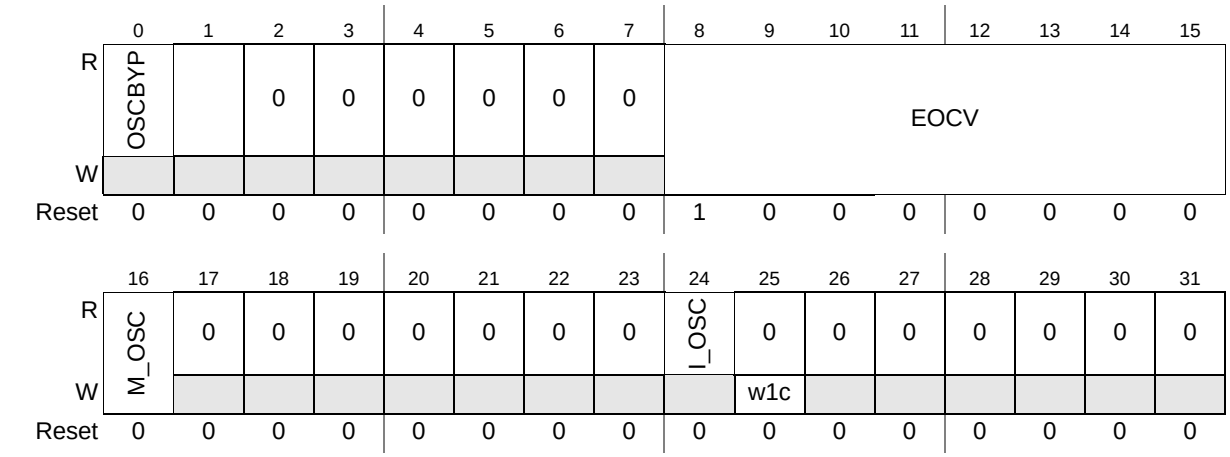


Figure 40-1. Crystal Oscillator Control Register (OSC_CTL)

Table 40-3. OSC_CTL field descriptions

Field	Description
OSCBYP	Crystal Oscillator bypass This bit specifies whether the oscillator should be bypassed or not. Software can only set this bit. System reset is needed to clear this bit. 0 Oscillator output is used as root clock. 1 XTALOUT is used as root clock.
EOCV	End of Count Value These bits specify the end-of-count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state. This counting period ensures that the external oscillator clock signal is stable before it can be selected by the system. When the oscillator counter reaches the value EOCV × 512, the oscillator available interrupt request is generated. The OSCCNT counter is kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask 0 Crystal oscillator clock interrupt is masked. 1 Crystal oscillator clock interrupt is enabled.
I_OSC	Crystal oscillator clock interrupt This bit is set by hardware when OSCCNT counter reaches the count value EOCV × 512. It is cleared by software by writing 1. 0 No oscillator clock interrupt occurred. 1 Oscillator clock interrupt pending.

40.2.1.2 RC Control Register (RC_CTL)

Address: Base + 0x0060 Access: Supervisor read/write, User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCTRIM					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 40-2. RC Control Register (RC_CTL)

Table 40-4. RC_CTL field descriptions

Field	Description
RCTRIM	Main RC trimming bits. This field corresponds (via two's complement) to a trim factor of –16 to +15. A +1 change in RCTRIM decreases the current frequency by Δ_{RCTRIM} (see the device data sheet). A –1 change in RCTRIM increases the current frequency by Δ_{RCTRIM} (see the device data sheet).

Chapter 41
Parallel Digital Interface (PDI)

41.1 Introduction

41.1.1 Overview

The Parallel Digital Interface (PDI) module provides an interface to high-speed external parallel devices such as analog-to-digital converters (ADCs) and image sensors. The PDI is conceived to move external parallel data into system memory or external DRAM with minimum intervention from the host processor.

Figure 41-1 shows the PDI block diagram.

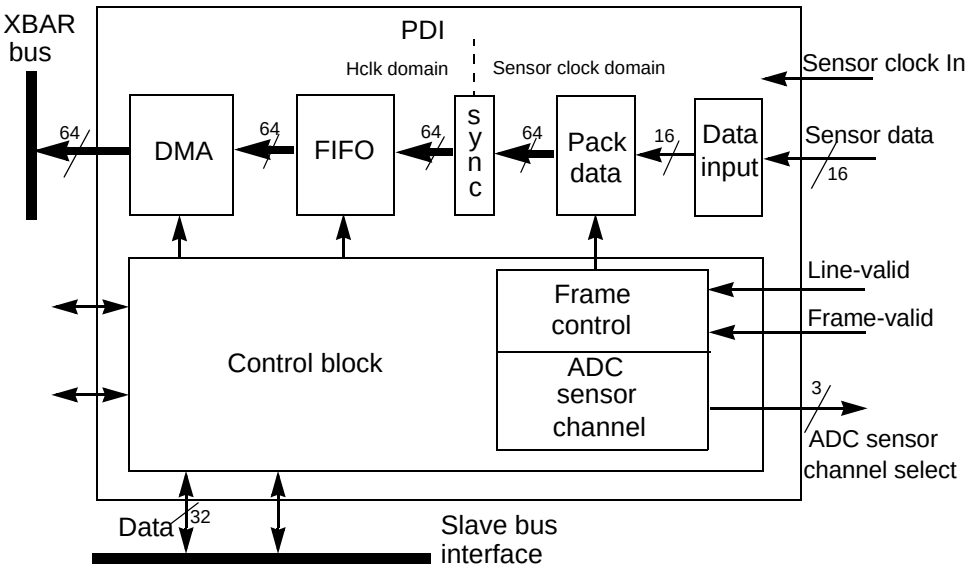


Figure 41-1. PDI block diagram

The PDI supports up to 16 bits of parallel data. Data is latched on either the positive or negative edge of the Sensor Clock In. For image sensors, the Sensor Clock In signal is referred to as the Pixel Clock. The Sensor Clock domain is asynchronous to the System Clock domain in the MCU. There is no phase relationship between these two clocks and hence data needs to be synchronized between the clock domains.

The Interface module uses a partial handshake synchronization scheme. An artifact of this synchronization scheme is that multiple samples are collected or packed and held constant while a request signal is synchronized from the Sensor Clock Domain to the System Clock domain. This synchronized request signal is used as an enable to sample the data in the System Clock domain while the data is held stable in the Sensor Clock domain. The packed samples are then stored directly into the FIFO. Once the FIFO has been filled with 32 bytes of data, the PDI automatically initiates a 32-byte burst transfer to offload the data to the desired target memory location. Once the transfer of 32 bytes is complete, the PDI sends a Data

Transfer Complete signal, which can be used to count the amount of data transferred to memory and to subsequently trigger an IRQ or DMA request to trigger further data processing.

41.1.2 Features

The PDI is designed to interface with an external ADC or image sensor.

41.1.2.1 Features applicable when interfacing to both types of peripherals

- Data capture on both the rising or falling edge of the sensor clock
- Receive FIFO with 32 elements of 64 bits for a total of 256 bytes
- Internal DMA engine to transfer data from FIFO to system memory or external DRAM
- Data Transfer Complete signal indicating successful transfer of 32 bytes of FIFO data to target memory location
- DMA Done interrupt indicates the successful transfer of destination buffer data
- Priority elevation signal intended for use with multi-port DRAM controller, where latencies can be very high and PDI access priority may need to be elevated
- Interface data width is configurable for 8, 10, 12, 14, and 16 bits; data widths 10, 12, or 14 bits can be right- or left-aligned and padded with zeros to make up 16 bits
- Support of sensor clock speeds up to the MCU system clock frequency

41.1.2.2 Additional features applicable when interfacing to ADCs

- Programmable mux select control sequence to support multiplexing of multiple sensor channels

41.1.2.3 Additional features applicable when interfacing to image sensors

- Support for aspect ratio up to 65535 × 4096
- Programmable PDI active frame to capture a subregion of an image frame
- Basic frame control through Frame-Valid and Line-Valid signals
- Host processor synchronization with frame data by Line Complete and Frame Complete interrupts
- Support for monochrome and color image sensors
- Option to discard every second frame (when double exposure used) by either discarding every second frame or every second pixel, depending on image sensor double exposure transmission format

41.1.3 Modes of operation

The PDI has three modes of operation:

- ADC mode
- Image Sensor mode
- Test mode

The ADC and Image Sensor modes interface to different external sensors. The Test mode generates image sensor data streams internal to the PDI without any external image sensor connected.

41.1.3.1 ADC mode

The PDI interfaces to external high-speed parallel ADCs. The PDI has an ADC Sensor Channel Select signal that allows the PDI to control the selection of data from multiple sensor channels. One configuration involves an analog multiplexer whose output drives a single high-speed ADC as shown in Figure 41-2. Analog sensor signals are converted to digital values by the ADC. The analog multiplexer controls which sensor channel data is converted by the ADC through digital select pins that are driven by the PDI's ADC Sensor Channel Select signal. The ADC Sensor Channel Select is a three-pin bus that allows up to 8 sensors to be multiplexed. The ADC Sensor Channel Select can change on every sensor clock and its sequence is programmable. A start and end channel can be defined. When enabled, the ADC Sensor Channel starts from the start channel encoding and linearly increases until the end channel state, and loops back to the start channel. See Figure 41-2.

The ADC Sensor Data can be sampled by the PDI from either the positive edge or the negative edge of Sensor Clock In. When the start channel and end channel are equal, the ADC Sensor Channel Select outputs a constant channel encoding. By reprograming the start and end channels, a new channel can be selected. This is useful when the user wants to sample a large number of samples from each channel instead of sequentially switching the channels at every sensor clock edge.

In ADC mode the PDI latches ADC Sensor Data if both the Line-Valid signal is asserted and if a programable number of sensor clocks (ADCVALIDDELAY) has elapsed after the first valid start channel encoding has been driven out by the PDI on the ADC Sensor Channel Select pins. See Section 41.4.1.1, ADC mode.

Line-Valid is sampled with the ADC Sensor Data and so it must be asserted before the active sensor clock in edge. See Figure 41-16. The polarity of Line-Valid is programmable.

Programming the ADCVALIDDELAY value to zero causes the PDI to latch data solely based on the state of Line-Valid.

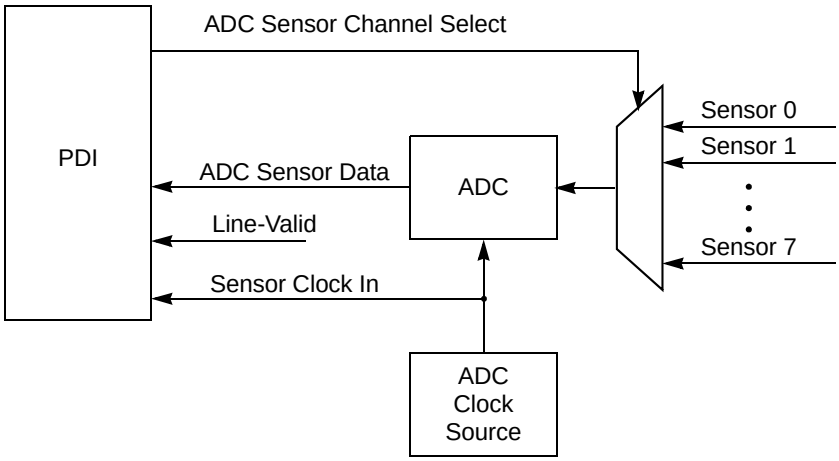


Figure 41-2. External ADC configuration with analog multiplexer

41.1.3.2 Image Sensor mode

The PDI supports both monochrome and color image sensors. The image sensor continuously captures frames. Each frame consists of an array of pixels. For monochrome image sensors, each pixel is represented by a single value or sample representing the degree of luminance. There are many different formats to represent color pixel information. The most common formats include RGB444 and YUV444, both of which require three values or samples to represent each pixel. In the RGB444 format, the first sample represents the Red color plane, the second sample represents the Green color plane and the third sample represents the Blue color plane. In the YUV444 format, the first sample represents the luminance and the second and third samples represent the chrominance. Therefore, for these common color pixel formats it takes three separate sample transfers from the image sensor for each pixel.

As shown in Figure 41-3, the clock source for the image sensor is generated externally. This external clock source is connected to the image sensor. The Pixel Clock, which is connected to the Sensor Clock In pin, is an output clock from the image sensor and an input pin to the PDI. The Sensor Data is framed around the Pixel Clock. Data can be captured on either the positive or negative Pixel Clock edge.

Each frame is scanned from the top line to the bottom line. Each line is scanned from the leftmost pixel to the rightmost pixel. Image sensors have blanking columns and lines that contain invalid data. Two control signals are used to signify that data from the image sensor is valid. Frame-Valid is asserted when a line contains valid data. Line-Valid is asserted when a pixel on a given line contains valid data.

When both Frame-Valid and Line-Valid are asserted, data from the image sensor is valid. The polarity of Frame-Valid and Line-Valid is programmable.

The PDI allows a smaller window, PDI Active Frame, to be targeted for capture from within the valid image sensor frame. This allows a region of interest to be captured from within the larger image sensor frame. See Figure 41-19. This PDI Active Frame is programmable.

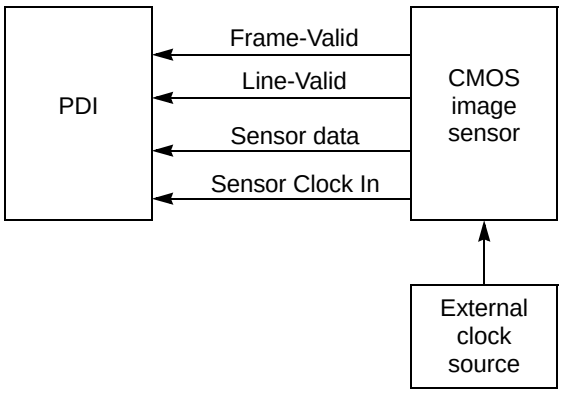


Figure 41-3. Image sensor

41.1.3.3 Test mode

The PDI has a Test mode where a data stream representing color image RGB444 data or a monochrome image can be generated without the need to connect an external sensor.

41.2 External signal description

Table 41-1 shows which of the PDI signals need to be connected to I/O pads.

Table 41-1. External PDI signals

Name	Function	I/O
Sensor Data	Sensor Data 16-bit width	Input
ADC Sensor Channel Select	In Image Sensor mode, this signal is not used. In ADC mode, this signal is an output and is a 3-bit bus that controls which analog sensor channel is to be selected at the input of the ADC.	Output
Frame-Valid	In Image Sensor mode, this signal is an input and is the frame valid control signal. In ADC mode, this signal is not used.	Input
Line-Valid	In Image Sensor mode, this signal is an input and is the line valid frame control signal. In ADC mode, this signal is an input and is signifies that the ADC sensor data is valid.	Input
Sensor Clock In	This is the sensor clock in. No matter if the clock to the sensor is generated internally in the MCU or externally, the sensor clock In is always used to provide the sensor clock to the PDI.	Input

41.2.1 Detailed signal descriptions

41.2.1.1 Sensor Clock In

As shown in Figure 41-2 and Figure 41-3, the Sensor Clock In pin is the clock that frames the Sensor Data. In the case of the CMOS image sensors, this is also commonly referred to as the Pixel Clock.

41.2.1.2 ADC Sensor Channel Select[2:0]

This three-pin output signal provides a digital select signal to choose a particular analog sensor channel in an ADC configuration where multiple analog sensor channels are multiplexed at the input of the ADC. See Section 41.4.1.1, ADC mode.

41.2.1.3 Line-Valid

This input signal is used in both image sensor mode and ADC mode. The polarity of Line-Valid is programmable.

In image sensor mode, Line-Valid is asserted when a valid pixel from within a line is being transferred by the image sensor. The PDI assumes valid data from image sensors when both Line-Valid and Frame-Valid are asserted. For image sensors that have only a Line-Valid signal, the Frame-Valid signal must be pulled up or down to the active state.

In ADC mode, Line-Valid signifies that the data from the ADC is valid.

The internal ADCVALIDDELAY can be disabled by programming ADCVALIDDELAY = 0.

In both the image sensor mode and ADC mode, Line-Valid is asserted along with the sensor data.

41.2.1.4 Frame-Valid

This input signal is only valid in image sensor mode. The polarity of Frame-Valid is programmable. Frame-Valid is asserted when the current line that an image sensor is transferring contains valid data.

Image sensor data is valid when both Frame-Valid and Line-Valid are asserted.

41.2.1.5 Sensor Data[15:0]

The PDI supports up to 16 bits of input data from external sensors. This signal is valid in both image sensor and ADC mode. It is not used in Test mode.

Sensors with data widths that are less than 16 bits must connect only the relevant data bits to the lower bits of the data bus as shown in Table 41-2.

Table 41-2. Sensor data connections

Sensor	Sensor Data Connection
8-bit Sensor	Sensor Data[7:0]
10-bit Sensor	Sensor Data[9:0]
12-bit Sensor	Sensor Data[11:0]
14-bit Sensor	Sensor Data[13:0]
16-bit Sensor	Sensor Data[15:0]

41.3 Memory map and register description

Table 41-3. PDI memory map

Offset from PDI_BASE (0xC3E4_0000)	Register	Access[1]	Reset Value[2]	Section/Page
0x0000	PDI Module Configuration Register (PDIMCR)	R/W	0x0000_1004	on page 1473
0x0004	PDI ADC Configuration Register (PDIADCCR)	R/W	0x0000_0000	on page 1475
0x0008	PDI Status Register (PDISR)	R/W	0x4000_0000	on page 1476
0x000C	PDI Interrupt Enable (PDIINTER)	R/W	0x0000_0000	on page 1478
0x0010	PDI Active Frame Origin Register (PDIAFOR)	R/W	0x0000_0000	on page 1480
0x0014	PDI Active Frame Size Register (PDIAFSR)	R/W	0x0000_0000	on page 1480
0x0018	PDI DMA Base Address Register (PDIDMABAR)	R/W	0x0000_0000	on page 1481
0x001C	PDI DMA Address Status Register (PDIDMAASR)	R	0x0000_0000	on page 1482
0x0020	PDI DMA Size (PDIDMASR)	R/W	0x0000_0000	on page 1482
0x0024	PDI Test 1 Register (PDIT1R)	R/W	0x0000_0000	on page 1483
0x0028	PDI Test 2 Register (PDIT2R)	R/W	0x0000_0000	on page 1484
0x002C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

[2] In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

41.3.1 Register descriptions

Only 32-bit register reads and writes are supported and transactions are at zero wait states.

41.3.1.1 PDI Module Configuration Register (PDIMCR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MODE		DAL	DEN	DMAEN	DMACON	DBIN			WIDTH			SCKE	FVP	LVP	INVD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FIFORST	0	BE	PBS	0	0	0	0	0	0	0	0	0	FIFOHIPRIOTRIG		
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0

Figure 41-4. PDI Configuration Register (PDIMCR)

Table 41-4. PDIMCR field descriptions

Field	Description
MODE	PDI operation mode 00 Reserved. 01 ADC mode. 10 Image Sensor mode. 11 TEST mode.
DAL	Data Alignment Determines alignment of 10, 12, or 14 data bits in the 16-bit data word. 0 Right-align data. Data widths 10, 12, or 14 bits are right-aligned and padded with zeros on the MSBs to make up 16 bits. 1 Left-align data. Data widths 10, 12, or 14 bits are left-aligned and padded with zeros on the LSBs to make up 16 bits.
DEN	Data Enable This field provides a way to disable latching of external sensor data into the PDI even if the external control signals such as Line-Valid or Frame-Valid are asserted. 0 Disable PDI Data Latching. 1 Enable PDI Data Latching.

Table 41-4. PDIMCR field descriptions (continued)

Field	Description
DMAEN	DMA Enable This field enables the DMA engine to transfer FIFO data to desired memory location. 0 Disable DMA. 1 Enable DMA. DMA is activated only when DMAEN is 1 and DMA_SIZE of PDIDMASR is not 0. When disabled, no data is transferred from the FIFO, even if there is enough data to trigger a 32-byte transfer. When DMACON is 0, DMAEN is cleared to 0 by internal logic after DMA done. When DMACON is 1, DMAEN is always high until software clears it to disable DMA.
DMACON	DMA Continue Indicates to the DMA engine that it should continue after transferring all data to the defined destination buffer. When the DMA continues, it re-initialize itself with the base address and size as programmed in the PDIDMABAR and PDIDMASR. 0 Stop after DMA Done. 1 Continue after DMA Done.
DBIN	Data Binning Option to discard every second frame (when double exposure used) by either discarding every second frame or every second pixel, depending on image sensor double exposure transmission format. This feature is valid for Image Sensor mode only. 000 No Data Binning. 001 Discard even-numbered frames. 010 Discard odd-numbered frames. 101 Discard even-numbered pixels/samples. 110 Discard odd-numbered pixels/samples. Other encodings are reserved.
WIDTH	Specifies the port width of the sensor data Data widths 10, 12, or 14 bits can be right- or left-justified and padded with zeros (normal data) or ones (inverted data) on the MSBs or padded with zeros on the LSBs to make up 16 bits. 000 8 bits. 001 10 bits. 010 12 bits. 011 14 bits. 100 16 bits. Other encodings are reserved.
SCKE	Sensor Clock Edge Selects the active sensor clock edge to latch sensor data. 0 Falling edge. 1 Rising edge.
FVP	Frame Valid Polarity Sets the polarity of the Frame Valid signal. 0 Active low. 1 Active high.
LVP	Line Valid Polarity Sets the polarity of the Line Valid signal. 0 Active low. 1 Active high.

Table 41-4. PDIMCR field descriptions (continued)

Field	Description
INVD	Invert the Sensor Data before latching into the PDI 0 Do not invert data. 1 Invert data.
FIFORST	FIFO Reset Resets the contents of the FIFO. 0 Do not perform reset. 1 Perform reset. When a 1 is written, the FIFO contents are reset to initial state. User needs to write 0 to enable FIFO.
BE	Big Endian Enable This field enables/disables big-endianness of a 64-bit wide data. 0 Little-endian. 1 Big-endian.
PBS	Pixel Byte Swap This field enables/disables byte swapping of pixel internal. 0 Byte swapping off. 1 Byte swapping on. This pixel byte swapping field enables byte swapping for packed 10-, 12-, 14-, or 16-bit input data. It has no effect on 8-bit input data.
FIFOHIPRIOTRIG	FIFO High Priority Access Trigger This field represents the number of elements in the FIFO after which a High Priority Access signal is triggered, to elevate PDI access priority. This feature is intended to be used with Multi-Port DRAM Controller, where latencies can be very high and PDI access priority can be elevated when required. 000 Never Trigger High Priority Access. 001 Trigger High Priority Access at FIFO level 1. 010 Trigger High Priority Access at FIFO level 2. 011 Trigger High Priority Access at FIFO level 3. 100 Trigger High Priority Access at FIFO level 4. 101 Trigger High Priority Access at FIFO level 5. 110 Trigger High Priority Access at FIFO level 6. 111 Trigger High Priority Access at FIFO level 7. The FIFO has 8 × 32-byte levels. When the FIFO fills up to level 1, 2, 3, 4, 5, 6, or 7, a High Priority Access signal is triggered to signal that the FIFO needs to be off-loaded to avoid risk of overflow.

41.3.1.2 PDI ADC Configuration Register (PDIADCCR)

The PDI ADC Configuration Register (PDIADCCR) is only used in ADC mode. This register has no effect in Image Sensor mode. It programs the sensor channel mux control for the ADC configuration that has multiple analog sensor channels muxed at the input of the ADC. See Figure 41-2.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADCVALIDDELAY								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	STARTC			0	ENDC		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-5. PDI ADC Configuration Register (PDIADCCR)

Table 41-5. PDIADCCR field descriptions

Field	Description
ADCVALIDDELAY	Specifies the number of sensor clocks in cycles after the first STARTC encoding is driven on the ADC Sensor Channel Select pins before the PDI can start latching data. The PDI will not latch data even if Line-Valid is asserted before the number of ADCVALIDDELAY clock cycles has been reached. 8-bit value for up to 255 sensor clock cycles. A value of 0 disables the ADCVALIDDELAY count.
STARTC	Start Channel of ADC Sensor Channel Select Sequence 000 Sensor channel 0. 001 Sensor channel 1. 010 Sensor channel 2. 011 Sensor channel 3. 100 Sensor channel 4. 101 Sensor channel 5. 110 Sensor channel 6. 111 Sensor channel 7.
ENDC	End Channel This field signifies the end channel encoding of the ADC Sensor Channel Select sequence. When the ENDC is reached the sequence loops back to the STARTC. The channel encoding loops back to 000 when 111 is reached. 000 Sensor channel 0. 001 Sensor channel 1. 010 Sensor channel 2. 011 Sensor channel 3. 100 Sensor channel 4. 101 Sensor channel 5. 110 Sensor channel 6. 111 Sensor channel 7.

41.3.1.3 PDI Status Register (PDISR)

The PDISR provides status information on the state of the FIFO as well as PDI error and interrupt conditions.

The FO, FU, LC, FC, HE, and DD fields in PDISR are sticky. Once asserted these bits can only be cleared by writing a 1 to their bit locations in PDISR.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FF	FE	FO	FU	0	0	0	0	LVS	FVS	LC	FC	0	0	HE	DD
W			w1c	w1c							w1c	w1c			w1c	w1c
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	FIFO_COUNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-6. PDI Status Register (PDISR)

Table 41-6. PDISR field descriptions

Field	Description
FF	FIFO full status This indicates that the FIFO is completely filled with data. This bit is not sticky. Active high. 0 FIFO is not full. 1 FIFO is full.
FE	FIFO empty status This indicates that the FIFO contains no data. The FIFO empty status field is always asserted on reset since the FIFO is always empty after a reset. This bit is not sticky. Active high. 0 FIFO is not empty. 1 FIFO is empty.
FO	FIFO overflow error A FIFO overflow error is asserted when an element is written to the FIFO when it is already full. When the FIFO overflows, no new data is written into the FIFO. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 A FIFO overflow error has not occurred. 1 A FIFO overflow error has occurred.
FU	FIFO underflow error A FIFO underflow error is asserted when there is an attempt to read the FIFO when it is empty. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 A FIFO underflow error has not occurred. 1 A FIFO underflow error has occurred.
LVS	Line-Valid status This bit signifies if Line-valid has been asserted, taking into account the polarity of the Line-Valid as set by the LVP field in the PDIMCR. This bit is not sticky. Active high. 0 Line-valid has not been asserted. 1 Line-valid has been asserted.
FVS	Frame-Valid status This bit signifies if Frame-valid has been asserted taking into account the polarity of Frame-Valid as set by the FVP field in the PDIMCR. This bit is not sticky. Active high. 0 Frame-valid has not been asserted. 1 Frame-valid has been asserted.

Table 41-6. PDISR field descriptions (continued)

Field	Description
LC	Line Complete interrupt Line Complete is asserted after a line of data, as defined by the PDIAFSR, has been transferred to memory. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 Line complete interrupt has not been asserted. 1 Line complete interrupt has been asserted. **Note:** This interrupt is valid only in Image Sensor mode.
FC	Frame Complete interrupt Frame Complete is asserted after a frame of data, as defined by the PDIAFSR, has been transferred to memory. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 Frame complete interrupt has not been asserted. 1 Frame complete interrupt has been asserted. **Note:** This interrupt is valid only in Image Sensor mode.
HE	AHB Transfer Error interrupt AHB transfer error is asserted after PDI receives an error response from XBAR slave. In this case, PDI will not stop and will go on transferring data even though the data in memory may be wrong. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 AHB transfer error has not been asserted. 1 AHB transfer error has been asserted.
DD	DMA Done interrupt DMA Done is asserted after the amount of data defined by the PDIDMASR[DMA_SIZE] field has been transferred to memory. This bit is sticky. After assertion, this bit must remain asserted until a 1 is written to clear it. Active high. 0 DMA done interrupt has not been asserted. 1 DMA done interrupt has been asserted.
FIFO_COUNT	FIFO Count This field indicates the data number in the FIFO. 6-bit value for data number in FIFO. Normally the max value is 32. Value 0 means FIFO empty and value 32 means FIFO full. When larger than 32, the FIFO has overflowed.

41.3.1.4 PDI Interrupt Enable Register (PDIINTER)

The PDIINTER controls whether each individual error or interrupt is enabled or masked. The FO, FU, LC, FC, HE, and DD fields in PDISR are conditions under which interrupts can be generated.

The FOE, FUE, LCE, FCE, HEE, and DDE fields in PDIINTER control whether interrupts are generated when the corresponding conditions in the PDISR are asserted.

A common error interrupt is asserted when (FO and FOE) or (FU and FUE) or (HE and HEE) are asserted.

A global interrupt is asserted when (FO and FOE) or (FU and FUE) or (LC and LCE) or (FC and FCE) or (HE and HEE) or (DD and DDE) are asserted.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FOE	FUE	0	0	0	0	0	BCE	LCE	FCE	0	0	HEE	DDE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-7. PDI Interrupt Enable Register (PDIINTER)

Table 41-7. PDIINTER field descriptions

Field	Description
FOE	FIFO overflow error enable Generate a FIFO overflow Interrupt when FOE and FO field in the PDISR is asserted. Active high. 0 FIFO overflow Interrupt is disabled. 1 FIFO overflow Interrupt is enabled.
FUE	FIFO underflow error enable Generates a FIFO underflow Interrupt when FUE and FU field in the PDISR is asserted. Active high. 0 FIFO underflow Interrupt is disabled. 1 FIFO underflow Interrupt is enabled.
BCE	Burst Complete enable Generates a Burst Complete pulse when a AHB burst (32bytes) transfer is done. Note this is not an interrupt, but an PDI output pulse to Timer block. Active high. 0 Burst complete pulse is disabled. 1 Burst complete pulse is enabled.
LCE	Line Complete interrupt enable Generates a Line Complete interrupt when LCE and LC field in the PDISR is asserted. Active high. 0 Line complete Interrupt is disabled. 1 Line complete Interrupt is enabled.
FCE	Frame Complete interrupt enable Generates a Frame Complete interrupt when FCE and FC field in the PDISR is asserted. Active high. 0 Frame complete Interrupt is disabled. 1 Frame complete Interrupt is enabled.
HEE	AHB Transfer Error interrupt enable Generates an AHB transfer error interrupt when HEE and HE field in the PDISR is asserted. Active high. 0 AHB transfer error Interrupt is disabled. 1 AHB transfer error Interrupt is enabled.
DDE	DMA Done interrupt enable Generates a DMA Done interrupt when DDE and DD field in the PDISR is asserted. Active high. 0 DMA done Interrupt is disabled. 1 DMA done Interrupt is enabled.

41.3.1.5 PDI Active Frame Origin Register (PDIAFOR)

This register specifies the top left corner of the PDI Active Frame relative to the valid image sensor frame as shown in Figure 41-19. This register is only valid in image sensor mode. In all other modes this register can be accessed and changed but the values are not used.

When ACT_FRM_ORG_X, ACT_FRM_ORG_Y fields in (PDIAFOR) and ACT_FRM_HEIGHT, ACT_FRM_WIDTH are all set to zero, the PDI captures the entire sensor image controlled by Frame-Valid and Line-Valid.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	ACT_FRM_ORG_Y											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACT_FRM_ORG_X															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-8. PDI Active Frame Origin Register (PDIAFOR)

Table 41-8. PDIAFOR field descriptions

Field	Description
ACT_FRM_ORG_Y	PDI Active Frame top corner Y-coordinate 12-bit value. Lines or rows are labeled 0 to 4095 lines from top to bottom.
ACT_FRM_ORG_X	PDI Active Frame origin (top left hand corner) X-coordinate For color image sensors each pixel can have multiple samples depending on the pixel format. 16-bit value. Samples are labeled 0 to 65535 from left to right.

41.3.1.6 PDI Active Frame Size Register (PDIAFSR)

This register specifies the width and height of the PDI Active Frame as shown in Figure 41-19. This register is not valid in ADC mode. In ADC mode this register can be accessed, but the values are not used.

The PDIAFOR and the PDIAFSR are used to determine the location and size of the PDI Active Frame. Any pixel that lies within this PDI Active Frame is captured by the PDI. The PDIAFOR and PDIAFSR is specified relative to the valid sensor frame. The valid sensor frame is defined as the frame when both Line-Valid and Frame-Valid signals are asserted. See Figure 41-19.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	ACT_FRM_HEIGHT											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACT_FRM_WIDTH															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-9. PDI Active Frame Size Register (PDIAFSR)

Table 41-9. PDIAFSR field descriptions

Field	Description
ACTIVE_FRM_HEIGHT	PDI Active Frame Height 12-bit value, up to 4096 lines. ACTIVE_FRM_HEIGTH + 1 is the active PDI frame width. See Figure 41-19.
ACTIVE_FRM_WIDTH	PDI Active Frame Width For color image sensors, each pixel has multiple samples depending on the format. 16-bit value, up to 65535 samples. ACTIVE_FRM_WIDTH + 1 is the active PDI frame width. See Figure 41-19.

41.3.1.7 PDI DMA Base Address Register (PDIDMABAR)

The PDI DMA Base Address Register (PDIDMABAR) programs the destination memory address to which the FIFO data is written. This register should be programmed before enabling the DMA. Once the DMA is enabled, this register can be re-programmed to set up the next DMA base address to be used by the DMA once the running DMA task has completed and the DMA Done interrupt has been triggered.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DMA_BASE_ADDR[26:11]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA_BASE_ADDR[10:0]											0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-10. PDI DMA Base Address Register (PDIDMABAR)

Table 41-10. PDIDMABAR field descriptions

Field	Description
DMA_BASE_ADDR	DMA Base Address Register Before enabling the DMA (PDIMCR[DMAEN]), program to memory destination start address. This register is latched only at the time when: • DMA is enabled. • DMACON of PDIMCR is set and DMA is done. 32-bit value, 32-byte-aligned DMA_BASE_ADDR[4:0] = 5'b0 to keep address 32-byte-aligned.

41.3.1.8 PDI DMA Address Status Register (PDIDMAASR)

The PDIDMAASR monitors the destination memory address to which the FIFO data is written. When the DMA is enabled, this register is updated with the contents of the PDIDMABAR. After the completion of each 8-byte transfer, PDIDMAASR is incremented by 8 bytes, allowing the user to monitor the DMA memory accesses.

Address: Base + 0x001C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DMA_ADDR_STAT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA_ADDR_STAT													0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-11. PDI DMA Address Status Register (PDIDMAASR)

Table 41-11. PDIDMAASR field descriptions

Field	Description
DMA_ADDR_STAT	DMA Address Status Register When the DMA is enabled, this register reflects the PDIDMABAR. With each 8-byte transfer, this register is incremented by 8 bytes to reflect the next address in memory where the next 8-byte FIFO data is stored. 32-bit value, 8-byte aligned. DMA_ADDR_STAT[2:0] = 3'b0 because the address is 8-byte aligned.

41.3.1.9 PDI DMA Size Register (PDIDMASR)

The PDIDMASR defines the size of the destination buffer that the DMA should fill in the target memory. This register should be programmed before the DMA is enabled. When the DMA has transferred the amount of data specified by this register, the DMA Done Interrupt is triggered. If DMA_SIZE is less than 32 bytes, then a DMA transfer is triggered when DMA_SIZE bytes are in the FIFO. The transfer contains the valid data, followed by random values to pad up to 32 bytes to complete the 32-byte block needed for the transfer.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DMA_SIZE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DMA_SIZE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-12. PDI DMA Size Register (PDIDMASR)

Table 41-12. PDIDMASR field descriptions

Field	Description
DMA_SIZE	DMA Size Register Number of bytes to be transferred by DMA to destination buffer in target memory.

41.3.1.10 PDI Test 1 Register (PDIT1R)

The PDI Test 1 Register (PDIT1R) is only valid in Test mode. In all other modes, these registers can be read and written, but the values are not used.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TEST_VALUE_1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-13. PDI Test 1 Register (PDIT1R)

Table 41-13. PDIT1R field descriptions

Field	Description
MC	Specifies the internal built-in self-test data pattern to be generated. 0 Color Image Sensor Test pattern. 1 Monochrome Image Sensor Test pattern. The Color Image Sensor data pattern models color pixel information with alternating color and white pixels, which correspond to {0xFFFF, 0xFFFF, 0xFFFF, TEST_VALUE_1, TEST_VALUE_2, TEST_VALUE_3}. The Monochrome Image Sensor Test pattern models alternating black and white pixels, which correspond to {0x0000, TEST_VALUE_1}. These data patterns are continuously generated internal to the PDI as long as the PDI is in Test mode.
TEST_VALUE_1	Test Value 1 used to generate the Built in Test mode data patterns 16-bit test value. For Color Image Sensor test pattern, TEST_VALUE_1 corresponds to the Red value of the RGB format of the color pixel. For Monochrome Image Sensor test pattern, TEST_VALUE_1 corresponds to the luminance of the dark pixel in the checkerboard pattern.

41.3.1.11 PDI Test 2 Register (PDIT2R)

The PDI Test 2 Register (PDIT2R) is only used in Test mode. In all other modes this register can be read and written to but the values are not used.

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TEST_VALUE_2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TEST_VALUE_3															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 41-14. PDI Test 2 Register (PDIT2R)

Table 41-14. PDIT2R field descriptions

Field	Description
TEST_VALUE_2	Test Value 2 used to populate the test mode data patterns. 16-bit test value. For Color Image Sensor test pattern, TEST_VALUE_1 corresponds to the Green value of the RGB format of the color pixel in the checkerboard pattern. For Monochrome Image Sensor, this field is not used.
TEST_VALUE_3	Test Value 3 used to populate the test mode data patterns. 16-bit test value. For Color Image Sensor test pattern, TEST_VALUE_3 corresponds to the Green value of the RGB format of the color pixel in the checkerboard pattern. For Monochrome Image Sensor Test, this field is not used.

41.3.2 AMBA AHB memory map

The PDI has an AMBA AHB bus Master interface that is capable of 32-byte burst write transfers to offload data from the FIFO to the desired target memory.

The PDI supports a WRAP4 burst mode at 64 bits.

Table 41-15. 32-byte burst transaction (4 beats of 64 bits)—big endian

Beat No	Address	Data
1	DMA_ADDR + 0x00	data[63:0] = 0 1 2 3 4 5 6 7
2	DMA_ADDR + 0x08	data[63:0] = 8 9 10 11 12 13 14 15
3	DMA_ADDR + 0x10	data[63:0] = 16 17 18 19 20 21 22 23
4	DMA_ADDR + 0x18	data[63:0] = 24 25 26 27 28 29 30 31

41.3.2.1 Data transfer signal

The PDI module generates a signal to indicate the successful transmission of a FIFO entry (32 bytes) to the target memory location. This signal is connected to aux input 3 of eTimer_2 to allow counting of the amount of data transferred to memory. Subsequently, the timer module can then issue an IRQ and/or a DMA request to trigger further data processing or distribution. It can be disabled by PDIINTER[BCE].

41.4 Functional description

41.4.1 Modes of operation

The PDI provides the following operating modes:

- ADC mode
- Image Sensor mode
- Test mode

These modes are described in the following subsections.

41.4.1.1 ADC mode

In ADC mode, the PDI captures parallel data from external ADC(s). Data can be latched on the positive or negative edge of Sensor Clock. The PDI's ADC Sensor Channel Select bus can be programmed to select data from different sensors through different multiplexer configurations. This allows the PDI to capture interleaving data samples from multiple sensors, which can be de-interleaved in system memory by the DMA engine or the host processor.

In the configuration shown in Figure 41-2, an analog multiplexer is used to mux analog signals from as many as eight sensor channels.

The ADC Sensor Channel Select signal changes on every sensor clock and is a repeating linearly increasing binary count. This count starts from a preprogrammed start channel encoding and continues to increase until the end channel is reached, after which the sequence wraps back to the start channel. If the count value gets to 111, the count wraps around back to the 000 encoding. This provides a sequential sampling of the sensor channels. The start and stop channels in is programed in the PDI ADC Configuration Register (PDIADCCR). See Section 41.3.1.2, PDI ADC Configuration Register (PDIADCCR).

When DMA_EN and DEN are set, DEN needs three sensor clock cycles to be synchronized to sensor clock domain. Once the synchronization is complete, LINE_Valid can be asserted and data starts latching. PDI interface is not able to catch the data during synchronization.

In ADC mode, the PDI latches if both Line-Valid is asserted and if a programmed number of sensor clocks has elapsed after the first start channel encoding has been driven out on the ADC Sensor Channel Select signals after the PDI has been enabled.

For example, if STARTC is set to 110 and the STOPC is set to 010, the following ADC Sensor Channel Select sequence is produced as shown in Figure 41-15.

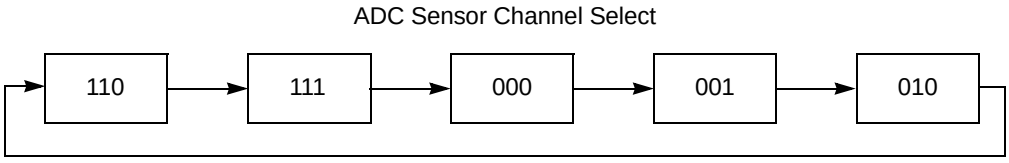


Figure 41-15. ADC sensor channel select sequence for Start Channel = 6, End Channel = 2

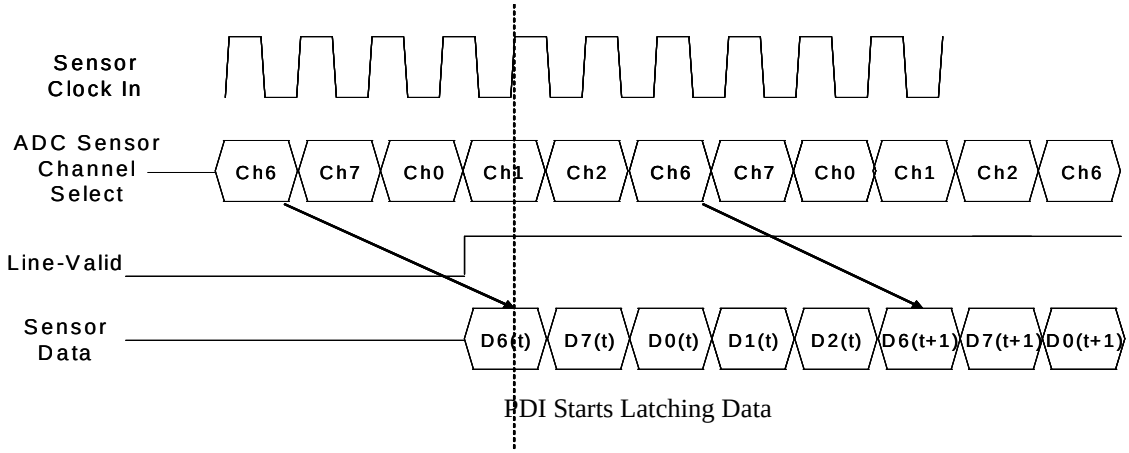


Figure 41-16. ADC data transfer Start Channel = 6, End Channel = 2, Line-Valid polarity active high, ADCVALIDDELAY = 0

Figure 41-16 shows an example of the PDI driving a select sequence through the ADC Sensor Channel Select and capturing data from multiple sensors. In this example, the ADC takes three cycles to perform that analog-to-digital conversion. Once enabled, the PDI drives out an ADC Sensor Channel Select sequence with start channel equal to 6 and stop channel equal to 2. This is the same sequence as illustrated by Figure 41-15. The Line-Valid polarity is active high and ADCVALIDDELAY is set to zero.

The first assertion of the active high Line-Valid signal identifies the first valid ADC Sensor Data, which corresponds to the first data sample from the start channel. Subsequent ADC Sensor Data samples correspond to the channel ordering of the ADC Channel Select sequence.

The sensor data stream latched into the PDI contains interleaved data samples from multiple sensors that the DMA module or host processor needs to de-interleave. This relies on the PDI capturing data from the sensors in the order of the ADC Sensor Channel Select sequence. The PDI stops latching data if Line-Valid is deasserted. Therefore, in order to de-interleave the data stream reliably, it is recommended that Line-Valid remain asserted after the first valid data from the ADC is captured for the duration when the PDI is enabled. If the Line-Valid is deasserted, the PDI stops latching data and the order of the sensor data from the different chanels is corrupted.

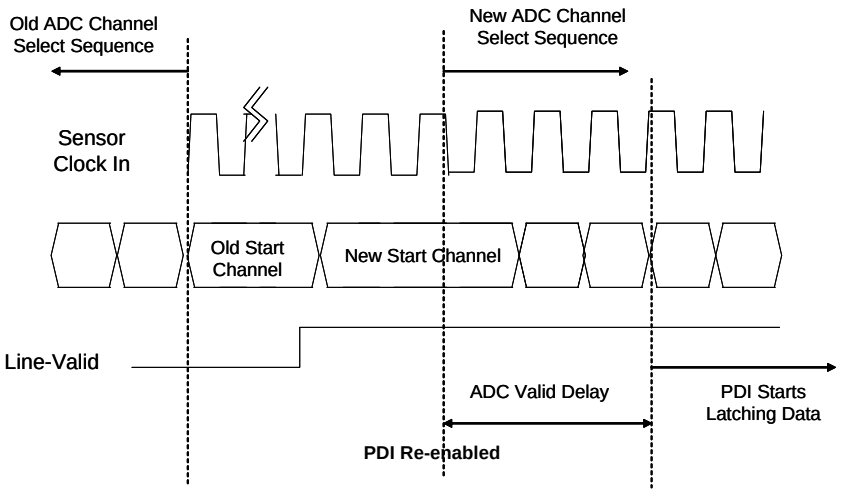


Figure 41-17. Reprogramming ADC channel select sequence
PDI Data Latching controlled by ADCVALIDDELAY = 3 and Line-Valid (active high)

The ADC Sensor Channel Select sequence can be reprogrammed as illustrated in Figure 41-17.

The following is the recommended way to reprogram the ADC Sensor Channel Select sequence:

1. Write PDIMCR[DEN] = 0 so that no new sensor data can be latched. When this occurs, the STARTC field is driven onto the ADC Sensor Channel Select pins.
2. If the current contents of the PDI FIFO are important, wait until the remaining contents of the FIFO have been transferred to memory.
3. Clear the FIFO elements by writing PDIMCR[FIFORST] = 1.
4. Write new values for the PDIADCCR[STARTC] and PDIADCCR[ENDC] fields.
5. If required, write a new value for PDIADCCR[ADCVALIDDELAY].
6. The new STARTC field is driven onto the ADC Sensor Channel Select pins.
7. Write PDIMCR[DEN] = 1. The new ADC Sensor Channel Sequence starts incrementing.
8. The PDI latches data if Line-Valid is asserted and after waiting ADCVALIDDELAY sensor cycles after the PDI is enabled.

It may be desired to sample more than one sample from a sensor before switching to another sensor. This can be achieved by programming the sensor channel that needs to be sampled into both the STARTC and ENDC fields of the PDIADCCR. The ADC Sensor Channel Select pins will not increment and will output the sensor channel encoding that is in both the STARTC and ENDC fields. Once the PDI has latched the required number of samples, reprogram the STARTC and ENDC fields to the next sensor.

41.4.1.2 Image sensor mode

The PDI is designed to interface to a single image sensor. Image sensors convert pictures into electrical signals through an array of photo sensors arranged in a grid-like structure. Each element of the grid is referred to as a pixel. For monochrome sensors, luminance information is collected for each pixel. For color image sensors, both brightness and color information need to be captured. There are many different

formats for color pixels. Some common ones includes RGB444 and YUV444, which have three values per pixel. See Section 41.1.3.2, Image Sensor mode.

The image sensor transfers sensor data one pixel at a time. The image or frame is split into lines and columns. Each line is scanned from top to bottom. Pixel data within a line is transferred left to right. Not every pixel in the image sensor will contain valid data. There are blanking columns and lines that contain invalid data. Two frame control signals from the image sensor indicate whether data from the image sensor is valid or not.

The image sensor scans each line starting from the top of the frame. As shown in Figure 41-18, the image sensor asserts Frame-Valid when the first line containing valid pixels starts, and deasserts when the first blanking line occurs within a frame. Line-Valid is asserted when the first valid pixel within a line occurs and is deasserted when the first blanking pixel occurs. When both Frame-Valid and Line-Valid are asserted, the pixel data from the sensor is valid. The Frame-Valid signal determines which rows of the image are valid and the Line-Valid signal determines which columns of the image are valid. These two signals bound the valid image from the image sensor.

The PDI can capture data from the image sensor either off the rising or falling edge from the Sensor Clock In or pixel clock.

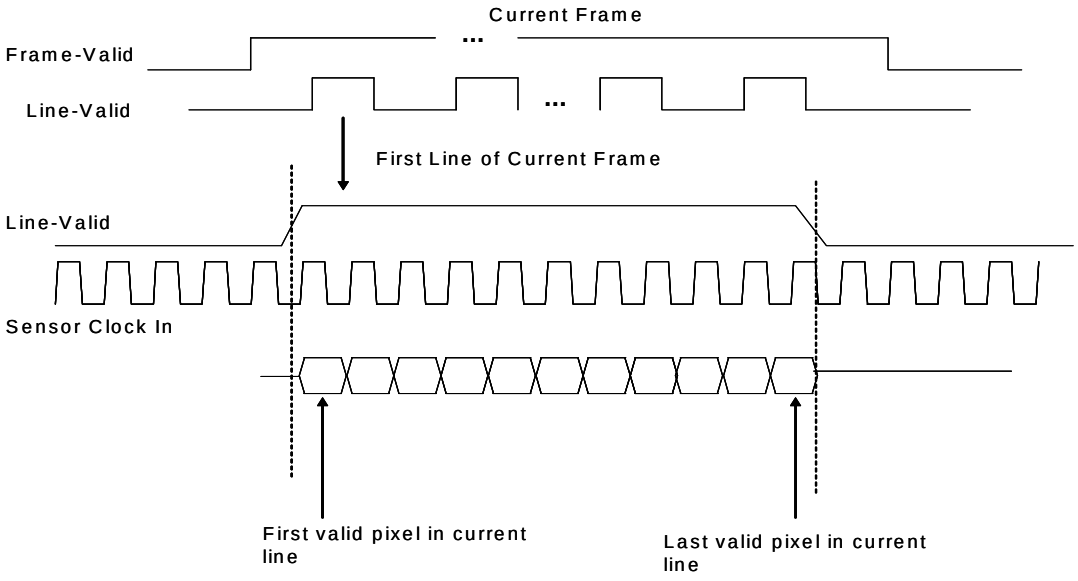


Figure 41-18. Image sensor data transfer—Frame-Valid and Line-Valid active high

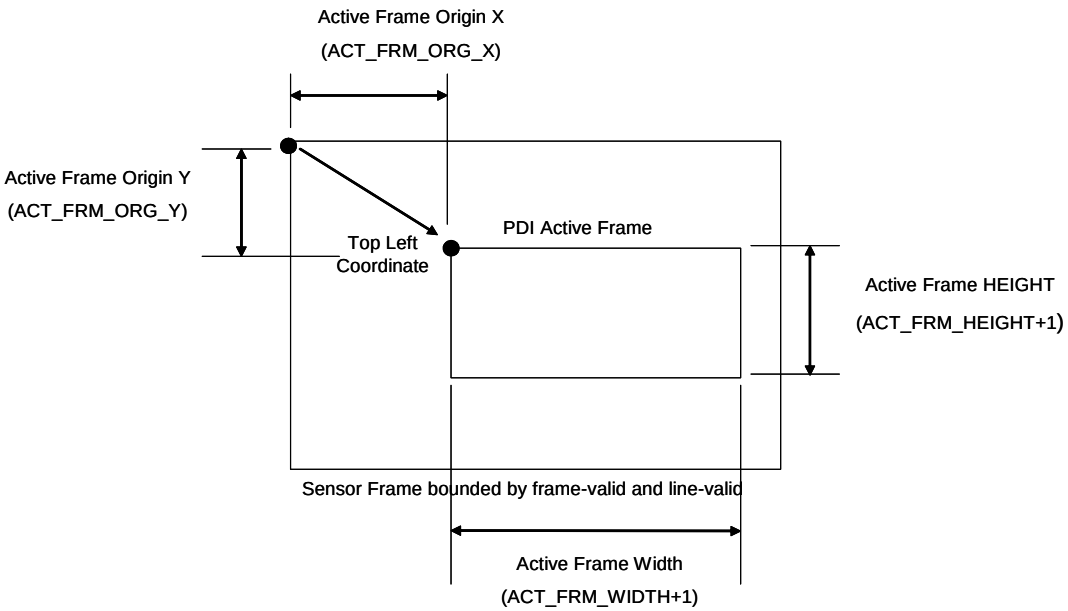


Figure 41-19. PDI active frame

The PDI allows a particular portion of the valid image sensor frame to be targeted for capture as shown in Figure 41-19. Only pixel information within this PDI Active Frame is latched by the PDI.

The PDIAFOR[ACT_FRM_ORG_X] and PDIAFOR[ACT_FRM_ORG_Y] fields determine the top level coordinate of the PDI Active Frame. The size of the PDI Active Frame can be specified by the PDIAFSR[ACT_FRM_HEIGHT] and PDIAFSR[ACT_FRM_WIDTH] fields. If the ACT_FRM_ORG_X, ACT_FRM_ORG_Y, ACT_FRM_HEIGHT, and ACT_FRM_WIDTH are all set to zero, the entire valid sensor frame is captured.

Figure 41-20 shows how the PDI determines which lines lie within the PDI Active Frame. In this illustration the ACT_FRM_ORG_Y determines at which line (Line-Y) the PDI is to start capturing data. The ACT_FRM_WIDTH field determines the last line of data that the PDI is supposed to capture (Line Y + ACT_FRM_HEIGHT + 1).

Figure 41-21 shows the PDI capturing data from within a PDI Active Frame Line. Note that for color image sensors, each pixel can be presented by multiple samples; for example, each YUV pixel has a luminance and two chrominance values. The PDI counts ACT_FRM_ORG_X samples before starting to latch data and continues to latch data until sample ACT_FRM_ORG_X + ACT_FRM_WIDTH + 1.

For color image sensors, the host processor or DMA module needs to split each value or sample from the color pixel for processing. The PDI transfers data from the image sensor in the order that it was scanned.

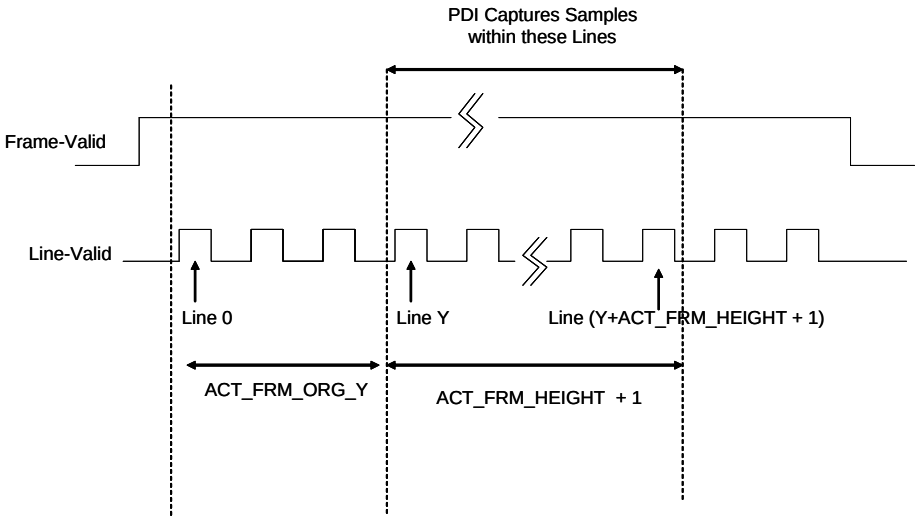


Figure 41-20. PDI active frame—determining PDI active frame lines

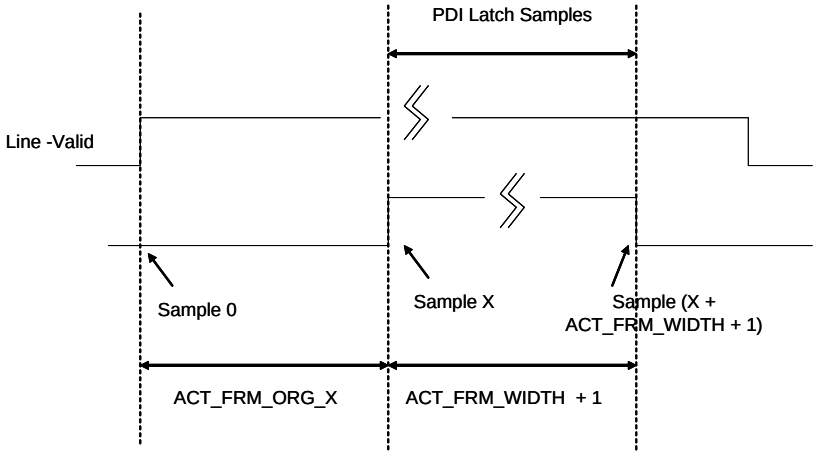


Figure 41-21. PDI active frame—capturing pixels from within PDI active frame line

The PDI Active Frame is designed to be re-programmable.

The recommended way to reprogram new PDI Active Frame Origin and PDI Active Frame Size values is as follows:

1. Disable the PDI through the PDIMCR[DEN] bit. This means that no new pixel data is latched by the PDI block.
2. Update the new PDI Active Frame origin through the PDIAFOR[ACT_FRM_ORG_X] and PDIAFOR[ACT_FRM_ORG_Y] fields.
3. Update the new PDI Active Frame size through the PDIAFSR[ACTIVE_FRM_WIDTH] and PDIAFSR[ACTIVE_FRM_HEIGHT] fields.
4. If the pixel data from the current frame is relevant, wait until the remaining contents of the FIFO have been transferred before resetting the FIFO.
5. Reset the FIFO by writing PDIMCR[FIFORST] = 1.

6. Enable the PDI to latch data by writing PDIMCR[DEN] = 1.

When DEN is enabled, the PDI checks to see if Frame-Valid is asserted. If Frame-Valid is asserted, this means that the image sensor is in the middle of transferring a frame. In this case, the PDI must not latch data until the next frame starts. This prevents partial frames from being captured.

41.4.1.2.1 Transferring image sensor data to memory

The PDI has a built-in DMA engine to facilitate the transfer of data from the FIFO into either system memory or external DRAM memory. When data is transferred into memory, it is critical for the host processor to know:

- When each line has been transferred to memory
  - Indicated by Line Complete IRQ
- When each frame has been transferred to memory
  - Indicated by Frame Complete IRQ
- When the DMA is done transferring the defined block of data
  - Indicated by DMA Done IRQ
- When each 32-byte block of data has been transferred
  - Indicated by Data Transfer Signal

With the above synchronization points, the host processor has full flexibility to synchronize the image processing steps to the incoming image sensor data with minimal host processor overhead.

Figure 41-22 illustrates how a single image frame is transferred to the target destination memory when the Region Of Interest (ROI) of the incoming frame is the same size as the destination buffer size in the target memory. To understand how the ROI is described, see Figure 41-19 and refer to PDIAFOR and PDIAFSR descriptions. The destination buffer is described by the PDIDMABAR and PDIDMASR configuration registers, which control the destination buffer start address and size.

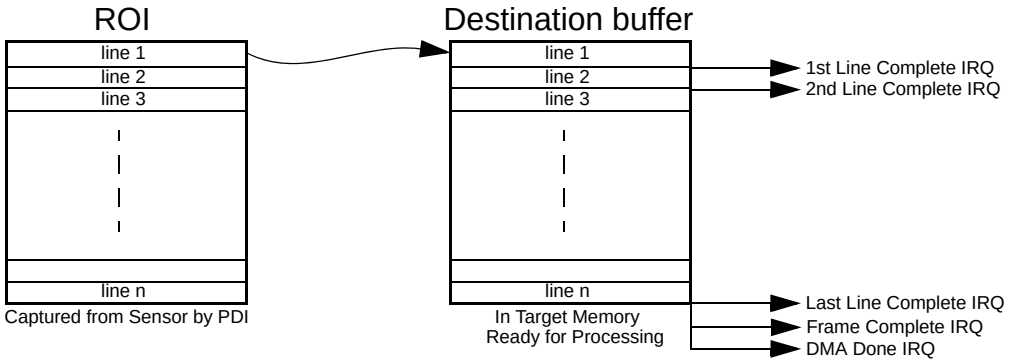


Figure 41-22. Frame transfer when ROI is the same size as the destination buffer

Figure 41-22 illustrates the case where the entire frame ROI is transferred to memory for further processing. In this case, the host processor can keep in sync with the incoming frame data via the Line Complete IRQ and the Frame Complete IRQ. Also, the DMA Done IRQ is triggered at the same time as the Frame Complete IRQ, since the frame ROI and destination buffer are of the same size.

Figure 41-23 illustrates the case where the lines of the frame are processed in real time allowing the destination buffer size to be smaller than the frame ROI size. The destination buffer is conceptually split into two regions, to allow the host processor to process the data in one region while the PDI fills the other region with the next lines of data. In the example shown here, the PDI would start by filling line a and b. The host processor would count the Line Complete IRQs and upon receiving two IRQs, the host processor would then start to process the data in line a and b. In parallel the PDI continues to fill line c and d. Once line d is filled, the host processor will have received another two Line Complete IRQs and will start to process the data stored in line c and d. In parallel, the PDI fills data into line a and b again. This mechanism allows a small memory to be used to store image data when the data can be processed in real time. The entire image does not need to be stored for further processing.

The DMA automatically continues to loop around the circular buffer as long as the DMACON field in the PDIMCR is set. When the destination buffer size of data has been transferred, the DMA Done IRQ is triggered. Also, when the entire frame size has been transferred, the Frame Complete IRQ is triggered.

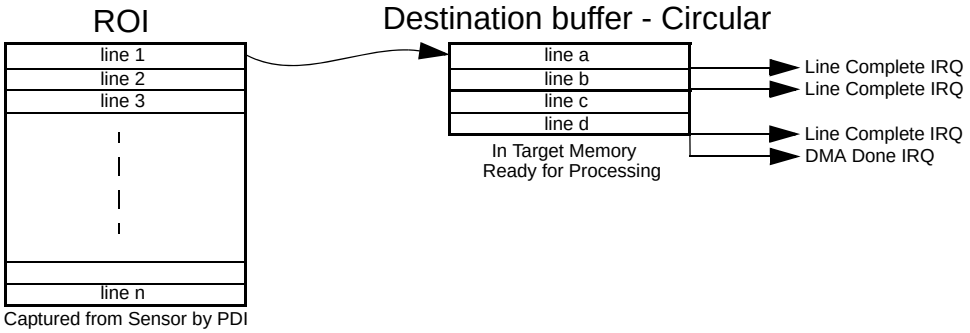


Figure 41-23. Frame transfer when ROI is larger than the destination buffer

Moreover, in the above example, the Data Transfer Signal can be used instead of the Line Complete IRQ to monitor the filling of the destination buffer. When the Data Transfer Signal is connected to an on-chip internal timer module, the signal can be used to count the amount of data that has been transferred, as it is triggered with each 32-byte data transfer. This allows the host processor to be triggered by a timer IRQ after so many blocks of 32 bytes have been filled in the destination buffer, minimizing the interaction between host processor and PDI.

41.4.1.2.2 Destination buffer—32-byte alignment

The PDI DMA engine transfers data from the FIFO to the destination buffer in the target memory by 32-byte burst transfers. Therefore, 32-byte transactions are aligned to a 32-byte address boundary. This ensures that the start of each frame and the start of each new line are 32-byte aligned in the target memory. In the case where the frame size and line length are 32-byte aligned, the data is stored in memory as illustrated in Figure 41-24. The data is stored sequentially in memory, with no gaps between each line.

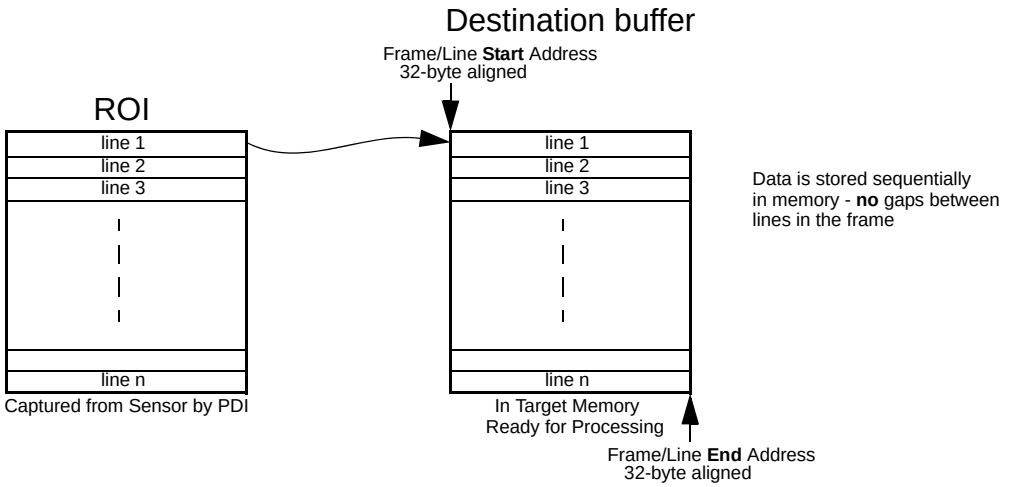


Figure 41-24. Frame size and line length—32-byte aligned

In the case where the frame size and line length are not 32-byte aligned, the data is stored in memory as illustrated in Figure 41-25. The data is stored sequentially in memory, with gaps between each line, to align the start of each new line to a 32-byte address boundary. In this case, when the DMA engine gets to the end of the line and there are fewer than 32 bytes of data to be transferred, the data is transferred in a 32-byte burst containing the line data, followed by zeroes to pad up to 32 bytes to complete the burst transaction.

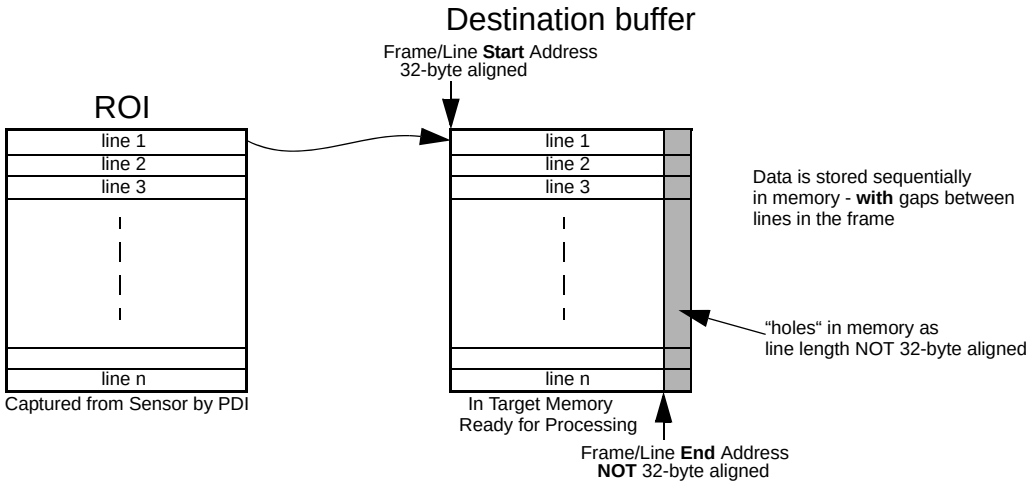


Figure 41-25. Frame size and line length—not 32-byte aligned

41.4.1.3 Test Mode

The PDI Test mode allows the PDI to internally generate sensor data streams without the need to attach an external sensor. No data synchronization is tested in the mode since the data is generated from the same clock domain as the System Clock domain.

This mode is enabled by programming the PDIMCR[MODE] field. In this test mode, the PDI can be used to mimic interfacing to color and monochrome image sensors through the PDIT1R[TEST_MODE].

A data stream of alternating color and white pixels or alternating dark and white pixels is generated for color and monochrome image sensors, respectively. The color data stream consists of the following repeating pattern: {0xFFFF, 0xFFFF, 0xFFFF, TEST_VALUE_1, TEST_VALUE_2, TEST_VALUE_3}, representing the three color planes of a color sensor and then a white color pixel. The monochrome data stream consists of repeating {0x0000, TEST_VALUE_1} values. TEST_VALUE_1, TEST_VALUE_2, and TEST_VALUE_3 are fields in PDIT1R and PDIT2R.

41.4.2 Interface block

The PDI can support as many as 16 bits of parallel data. Data widths other than 8 bits or 16 bits (10, 12, or 14 bits) are right- or left-justified (as defined by the PDIMCR[DAL] field) and the remaining bits padded with zeros. Either four 16-bit samples or eight 8-bit samples are packed into a single 64-bit vector before being stored in the FIFO.

The PDI uses a partial handshaking toggle request scheme to synchronize data between the sensor clock domain and the system clock domain since these domains are asynchronous to each other. Data in the sensor clock domain is held constant while a transfer request is synchronized into the system clock domain. The PDI supports sensor clock frequencies as high as the system clock frequency, since 8-bit or 16-bit input samples are assembled to 64-bit data, which is indicated valid by a one-cycle pulse in the sensor clock domain. The transfer request (data valid pulse) from the sensor clock domain is received in the system clock domain within two system clock cycles. Once the synchronized request signal is received in the system clock domain, the packed data from the sensor clock domain that is held stable is sampled.

When data samples are 8-bit, a total of eight sensor clock cycles is required to build up the 64-bit vector. This 64-bit vector is held for eight sensor clock cycles before the next 64-bit vector is available to be synchronized. Similarly, when data samples are 16-bit samples, four sensor clock cycles are needed to build the 64-bit vector and this 64-bit vector is held for four sensor clock cycles before the next 64-bit vector is synchronized. Once synchronized, the 64-bit vector is written directly into the FIFO.

Figure 41-26 shows how 16-bit samples are packed into the big-endian system memory.

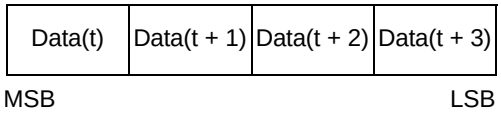


Figure 41-26. Memory view of data packing for 16-bit sensor data

Figure 41-27 shows how 8-bit samples are packed and arranged into the big-endian system memory.

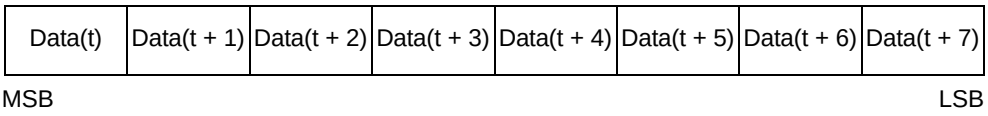


Figure 41-27. Memory view of data packing for 8-bit sensor data

41.4.3 FIFO operation

The FIFO inside the PDI is organized into 32 elements of 64 bits for a total size of 256 bytes, as shown in Table 41-16. Once the FIFO has been filled with 32 bytes of data, the PDI automatically initiates a 32-byte burst transfer to offload the data to the desired target memory, provided that the DMA is enabled (PDIMCR[DMAEN] = 1).

Table 41-16. 256 Byte FIFO—8 × 32 bytes

32-byte Block	Entry	Data[63:0]							
0	0	0	1	2	3	4	5	6	7
	1	8	9	10	11	12	13	14	15
	2	16	17	18	19	20	21	22	23
	3	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 1									
1	4	0	1	2	3	4	5	6	7
	5	8	9	10	11	12	13	14	15
	6	16	17	18	19	20	21	22	23
	7	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 2									
2	8	0	1	2	3	4	5	6	7
	9	8	9	10	11	12	13	14	15
	10	16	17	18	19	20	21	22	23
	11	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 3									
3	12	0	1	2	3	4	5	6	7
	13	8	9	10	11	12	13	14	15
	14	16	17	18	19	20	21	22	23
	15	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 4									
4	16	0	1	2	3	4	5	6	7
	17	8	9	10	11	12	13	14	15
	18	16	17	18	19	20	21	22	23
	19	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 5									
5	20	0	1	2	3	4	5	6	7
	21	8	9	10	11	12	13	14	15
	22	16	17	18	19	20	21	22	23
	23	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 6									
6	24	0	1	2	3	4	5	6	7
	25	8	9	10	11	12	13	14	15
	26	16	17	18	19	20	21	22	23
	27	24	25	26	27	28	29	30	31
HIGH PRIORITY ACCESS TRIGGER LEVEL 7									
7	28	0	1	2	3	4	5	6	7
	29	8	9	10	11	12	13	14	15
	30	16	17	18	19	20	21	22	23
	31	24	25	26	27	28	29	30	31

The FIFO supports seven High Priority Access Trigger Levels, which can be programmed in the PDIMCR[FIFOHIPRIOTRIG] field. When the PDI is required to offload data to an external DRAM, the latencies can be very large. Therefore, to identify that the real time requirements of the streaming client, such as the Image Sensor, are maintained, it is essential to set the High Priority Access Trigger Level at the optimal point. By default, it is set to level 4, which supports the PDI to offload data from its FIFO before the FIFO overflows, given a system clock to sensor clock ratio of 2:1. This is only true when the

PDI is used with a Multi-Port DRAM Controller supporting a priority elevation request signal. For other ratios, the High Priority Access Trigger Level can be set to 1, 2, 3, 5, 6, or 7 to give maximum flexibility to the Multi-Port DRAM Controller to optimize the performance on the external DRAM. When the High Priority Access Trigger Level is hit, the PDI asserts a priority elevation request signal that is connected directly to the Multi-Port DRAM Controller. This allows the DRAM Controller to subsequently place the PDI outstanding request to DRAM as its highest priority. The priority elevation request signal remains asserted as long as the PDI FIFO level is above the High Priority Access Trigger Level.

This implementation allows the Multi-Port DRAM Controller to optimize DRAM access for performance by giving higher bandwidth to other clients, while maintaining the real-time requirements of the PDI, as it is able to alert the Multi-Port DRAM Controller when it has been starved for too long and needs to offload data from its FIFO.

The FIFO has two status flags to indicate when the FIFO has no data and when the FIFO is completely full. These status flags form the FF and FE status flags of the PDI Status Register. If there is an attempt to perform a read from the FIFO when it is empty, a FIFO underflow error is generated. If there is an attempt to write to the FIFO when the FIFO is full, a FIFO overflow error is generated. When a FIFO overflow occurs, no new data can be written into the FIFO. The FIFO underflow and overflow are error bit fields in the PDISR. The FUE and FOE fields in the PDIINTER enable the FIFO underflow and overflow interrupts to be generated.

The DMA will not initiate a transfer of FIFO data until at least 32 bytes are in the FIFO. The DMA will be triggered when up to 32 bytes are in the FIFO. The PDI pads random data into FIFO to 32-byte-align the data of every line.

The contents of the FIFO can be cleared by writing PDIMCR[FIFORST] = 1.

41.4.4 Interrupts

The PDI has four interrupt sources.

- The Line Complete interrupt is generated after a line of data, as defined by the PDI Active Frame Size Register (PDIAFSR), has been transferred to memory. When this occurs, PDISR[LC] is set. If PDIINTER[LCE] = 1, an interrupt is generated. The interrupt can be cleared by writing PDISR[LC] = 1. The interrupt is cleared in a single cycle.
- The Frame Complete interrupt is generated after a frame of data, as defined by the PDI Active Frame Size Register (PDIAFSR), has been transferred to memory. When this occurs, PDISR[FC] is set. If PDIINTER[FCE] = 1, an interrupt is generated. The interrupt can be cleared by writing PDISR[FC] = 1. The interrupt is cleared in a single cycle.
- The DMA Done interrupt is generated after the amount of data defined by the PDI DMA Size register (PDIDMASR) has been transferred to memory. When this occurs, PDISR[DD] is set. If PDIINTER[DDE] = 1, an interrupt is generated. The interrupt can be cleared by writing PDISR[DD] = 1. The interrupt is cleared in a single cycle.
- The PDI has a common error interrupt. The common error interrupt is asserted by performing a logic or operation on all the error bits in the PDISR that have been enabled in the PDIINTER. These errors include FIFO overflow, FIFO underflow, and bus burst transfer error.

This page is intentionally left blank.

Chapter 42
Periodic Interrupt Timer (PIT)

42.1 Introduction

The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels. Figure 42-1 shows the PIT block diagram.

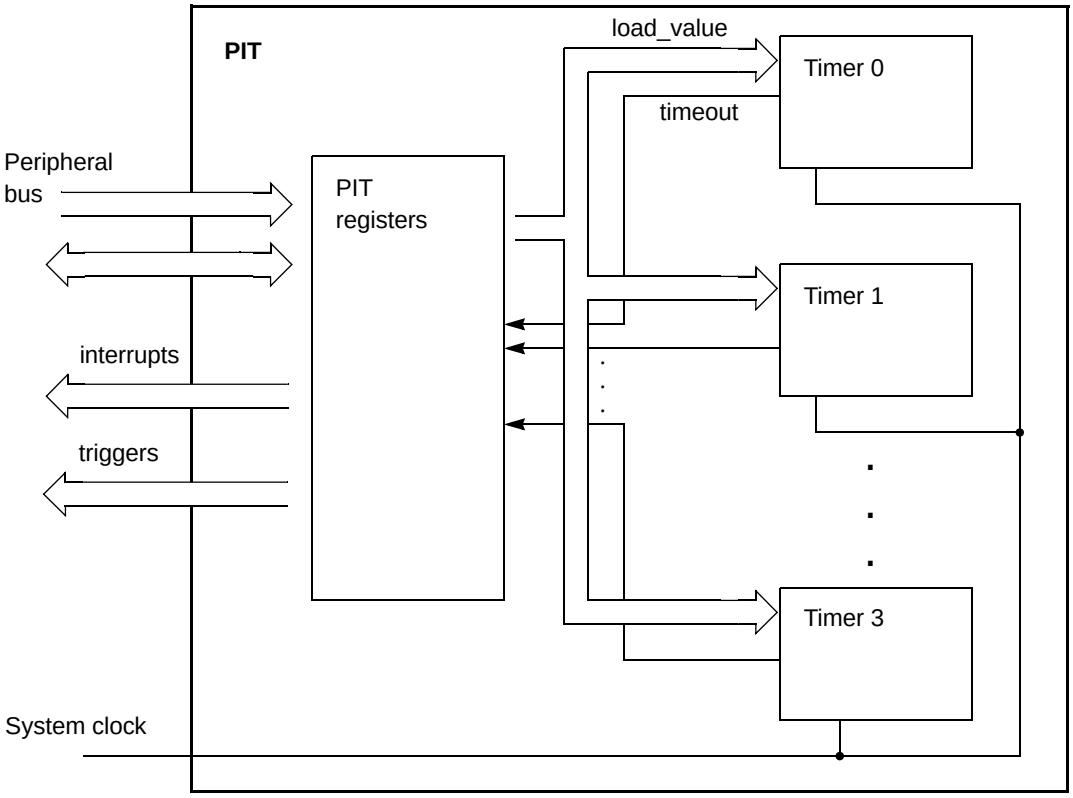


Figure 42-1. PIT block diagram

42.2 Features

The main features of this module are:

- Timers can generate DMA trigger pulses.
- Timers can generate interrupts.
- All interrupts are maskable.
- Independent timeout periods exist for each timer.

42.3 Signal description

The PIT module has no external pins.

42.4 Memory map and register description

This section provides a detailed description of all registers accessible in the PIT module. The PIT registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generate a bus error termination.

42.4.1 Memory map

Table 42-1 shows the PIT memory map.

Table 42-1. PIT memory map

Offset from PIT_BASE (0xC3FF_0000)	Register	Access [1]	Reset value	Location
0x0000	PITMCR—PIT Module Control Register	R/W	0x0000_0000	on page 1501
0x0004–0x00FF	Reserved			
Timer Channel 0				
0x0100	LDVAL0—Timer 0 Load Value Register	R/W	0x0000_0000	on page 1501
0x0104	CVAL0—Timer 0 Current Value Register	R	0x0000_0000	on page 1502
0x0108	TCTRL0—Timer 0 Control Register	R/W	0x0000_0000	on page 1503
0x010C	TFLG0—Timer 0 Flag Register	R/W	0x0000_0000	on page 1503
Timer Channel 1				
0x0110	LDVAL1—Timer 1 Load Value Register	R/W	0x0000_0000	on page 1501
0x0114	CVAL1—Timer 1 Current Value Register	R	0x0000_0000	on page 1502
0x0118	TCTRL1—Timer 1 Control Register	R/W	0x0000_0000	on page 1503
0x011C	TFLG1—Timer 1 Flag Register	R/W	0x0000_0000	on page 1503
Timer Channel 2				
0x0120	LDVAL2—Timer 2 Load Value Register	R/W	0x0000_0000	on page 1501
0x0124	CVAL2—Timer 2 Current Value Register	R	0x0000_0000	on page 1502
0x0128	TCTRL2—Timer 2 Control Register	R/W	0x0000_0000	on page 1503
0x012C	TFLG2—Timer 2 Flag Register	R/W	0x0000_0000	on page 1503
Timer Channel 3				
0x0130	LDVAL3—Timer 3 Load Value Register	R/W	0x0000_0000	on page 1501
0x0134	CVAL3—Timer 3 Current Value Register	R	0x0000_0000	on page 1502
0x0138	TCTRL3—Timer 3 Control Register	R/W	0x0000_0000	on page 1503
0x013C	TFLG3—Timer 3 Flag Register	R/W	0x0000_0000	on page 1503
0x0140–0x3FFF	Reserved			

[1] In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

42.4.2 Register descriptions

This section describes in address order all the PIT registers and their individual bits.

42.4.2.1 PIT Module Control Register (PITMCR)

The PIT Module Control Register (PITMCR) controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 42-2. PIT Module Control Register (PITMCR)

Table 42-2. PITMCR field descriptions

Field	Description
MDIS	Module Disable This bit disables the module clock. This bit should be enabled before any other setup is done. 0 Clock for PIT timers is enabled. 1 Clock for PIT timers is disabled.
FRZ	Freeze Allows the timers to be stopped when the device enters debug mode. 0 Timers continue to run in debug mode. 1 Timers are stopped in debug mode.

42.4.2.2 Timer Load Value Register *n* (LDVAL*n*)

The Timer Load Value Register *n* (LDVAL*n*) selects the timeout period for the timer interrupts.

Address: Base + = 0x0100 (LDVAL0) Base + = 0x0120 (LDVAL2)
Base + = 0x0110 (LDVAL1) Base + = 0x0130 (LDVAL3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSV31	TSV30	TSV29	TSV28	TSV27	TSV26	TSV25	TSV24	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 42-3. Timer Load Value Register *n* (LDVAL*n*)

Table 42-3. LDVAL*n* field descriptions

Field	Description
TSV*n*	Time Start Value Bits These bits set the timer start value. The timer counts down until it reaches 0, then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer. Instead, the value is loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 42-8).

42.4.2.3 Current Timer Value Register *n* (CVAL*n*)

The Current Timer Value Register *n* (CVAL*n*) indicates the current timer position.

Address: Base + = 0x0104 (CVAL0) Base + = 0x0124 (CVAL2)
Base + = 0x0114 (CVAL1) Base + = 0x0134 (CVAL3) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 42-4. Current Timer Value register *n* (CVAL*n*)

Table 42-4. CVAL*n* field descriptions

Field	Description
TVL*n*	Current Timer Value These bits represent the current timer value. Note that the timer uses a downcounter. **Note:** The timer values are frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see Figure 42-2).

42.4.2.4 Timer Control Register *n* (TCTRL*n*)

The Timer Control Register *n* (TCTRL*n*) contains the control bits for each timer.

Address: Base + = 0x0108 (TCTRL0) Base + = 0x0128 (TCTRL2)
Base + = 0x0118 (TCTRL1) Base + = 0x0138 (TCTRL3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 42-5. Timer Control register *n* (TCTRL*n*)

Table 42-5. TCTRL*n* field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0 Interrupt requests from Timer *n* are disabled. 1 Interrupt is requested whenever TIF is set. When an interrupt is pending (TIF set), enabling the interrupt immediately causes an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0 Timer is disabled. 1 Timer is active.

42.4.2.5 Timer Flag Register *n* (TFLG*n*)

The Timer Flag Register *n* (TFLG*n*) contains the PIT interrupt flag.

Address: Base + = 0x010C (TFLG0) Base + = 0x012C (TFLG2)
Base + = 0x011C (TFLG1) Base + = 0x013C (TFLG3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 42-6. Timer Flag register *n* (TFLG*n*)

Table 42-6. TFLG*n* field descriptions

Field	Description
TIF	Time Interrupt Flag TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0 Timeout has not yet occurred. 1 Timeout has occurred.

42.5 Functional description

42.5.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts. Each interrupt is available on a separate interrupt line.

NOTE

The PIT and the DMA need to run at the same frequency (synchronous). For the PIT frequency it must be considered the PIT runs with the peripheral set 0 clock, which clocks also DSPI's, FlexCAN's, and LinFlex.

42.5.1.1 Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach zero. Then each loads its respective start value again. Each time a timer reaches zero, it generates a trigger pulse and sets the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted by first disabling, then enabling, the timer with the TEN bit (see Figure 42-7).

The counter period of a running timer can be modified by first disabling the timer, setting a new load value, and then enabling the timer again (see Figure 42-8).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value is loaded after the next trigger event (see Figure 42-9).

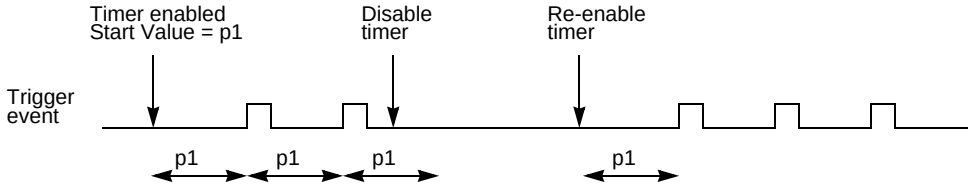


Figure 42-7. Stopping and starting a timer

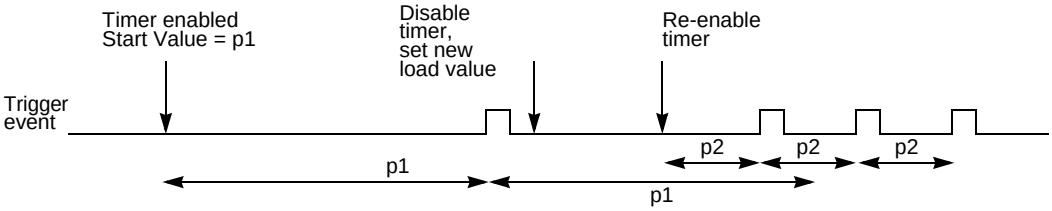


Figure 42-8. Modifying running timer period

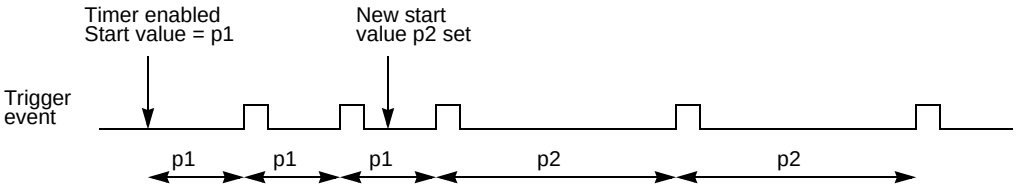


Figure 42-9. Dynamically setting a new load value

42.5.1.2 Debug mode

In Debug mode, the timers can be optionally frozen by setting the PITMCR[FRZ] bit. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, timer values), and then continue the operation. By default, timers are not frozen in Debug mode.

42.5.2 Interrupts

All of the timers support interrupt generation. See Table 32-10 for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to one when a timeout occurs on the associated timer, and are cleared to zero by writing a one to that TIF bit.

42.6 Initialization and application information

42.6.1 Example configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz.
- Timer 1 creates an interrupt every 5.12 ms.
- Timer 3 creates a trigger event every 30 ms.

First the PIT module needs to be activated by writing a zero to the PITMCR[MDIS] bit.

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles and timer 3 every 30 ms/20 ns = 1500000 cycles. The value for the LDVAL register trigger would be calculated as (period / clock period) – 1.

The LDVAL registers must be configured as follows:

- LDVAL for Timer 1: 0x0003_E7FF
- LDVAL for Timer 3: 0x0016_E35F

The interrupt for Timer 1 is enabled by programming TCTRL1[TIE] = 1. The timer is started by programming TCTRL1[TEN] = 1.

Timer 3 is used only for triggering. Therefore, Timer 3 is started by programming TCTRL3[TEN] = 1. The TIE bit stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3 for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

Chapter 43
Power Control Unit (MC_PCU)

43.1 Introduction

43.1.1 Overview

The Power Control Unit (MC_PCU) acts as a bridge for mapping the PMU peripheral to the MC_PCU address space.

Figure 43-1 depicts the MC_PCU block diagram.

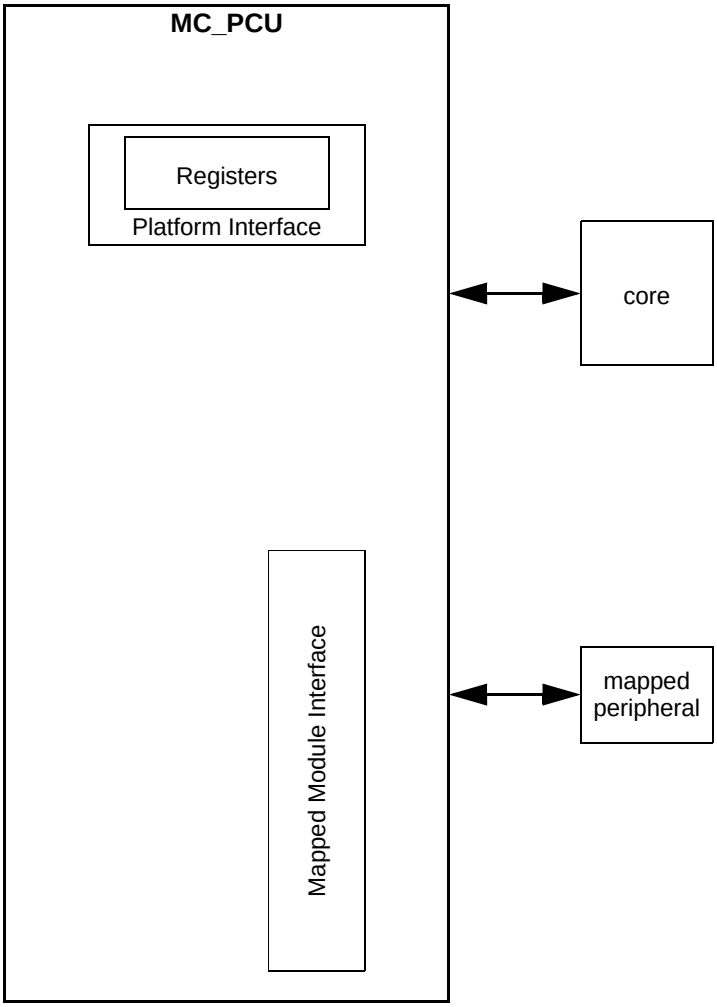


Figure 43-1. MC_PCU block diagram

43.1.2 Features

The MC_PCU includes the following features:

- Maps the PMU registers to the MC_PCU address space.

43.2 External signal description

The MC_PCU has no connections to any external pins.

43.3 Memory map and register description

43.3.1 Memory map

Table 43-1. MC_PCU register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_8040	PCU_PSTAT	Power Domain Status Register	word	read	read	read	on page 1509

NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

Table 43-2. MC_PCU memory map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_80004 … 0xC3FE_803C	reserved																	
0xC3FE_8040	PCU_PSTAT	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PD0
		W																
0x044 … 0x07C	reserved																	

Table 43-2. MC_PCU memory map (continued)

Address	Name	0 16	1 17	2 18	3 19	27 20	5 21	6 22	7 23	8 24	9 25	10 26	11 27	12 28	13 29	14 30	15 31
0xC3FE_8080 … 0xC3FE_80FC	PMU registers																
0xC3FE_8100 … 0xC3FE_BFFC	reserved																

43.3.2 Register descriptions

All registers may be accessed as 32-bit words, 16-bit halfwords, or 8-bit bytes. The bytes are ordered according to big endian. For example, the PD0 field of the PCU_PSTAT register may be accessed as a word at address 0xC3FE_8040, as a half-word at address 0xC3FE_8042, or as a byte at address 0xC3FE_8043.

43.3.2.1 Power Domain Status Register (PCU_PSTAT)

Address 0xC3FE_8040 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 43-2. Power Domain Status Register (PCU_PSTAT)

This register reflects the power status of all available power domains.

Table 43-3. Power Domain Status Register (PCU_PSTAT) field descriptions

Field	Description
PD0	Power status for power domain 0. 0 Power domain is inoperable. 1 Power domain is operable.

This page is intentionally left blank.

Chapter 44
Power Management Controller (PMC)

NOTE

PMC is same as PMU.

44.1 Introduction

The power management controller (PMC) is compatible with 3.3 V or 5 V operation, provides voltage regulation to the microcontroller, and includes advanced power-on reset (POR), precision low voltage detectors (LVD), and a high voltage detector (HVD) supporting safe device operation at any time or condition.

44.2 Features

The PMC contains the following features:

- Power management controller compatible for both 5 V and 3.3 V operations
- Switched Mode Power Supply (SMPS) asynchronous buck regulator
- High-precision Low Voltage Detector (LVD) for:
  - PMC supply voltage VDDREG
  - VDD core voltage supply
  - VDDFLASH flash memory supply voltage
  - VDDADC ADC voltage supply
  - VDDIO I/O voltage supply
- High Voltage Detector for VDD core voltage supply
- A bandgap reference that generates the reference voltages and currents for voltage regulators and LVDs
- A POR monitor checks main regulator supply VDDREG and core supply VDD and enables system-correct function even at very low voltage supply levels.
- No power sequencing constraint applies due to the fact that the PMC has only one positive supply, and only the internal regulator is used to generate the 1.2 V core voltage.
- Direct measurement of PMC internal voltages is available at predefined ADC channels.

44.3 Block diagram

Figure 44-1 shows the block diagram for the PMC.

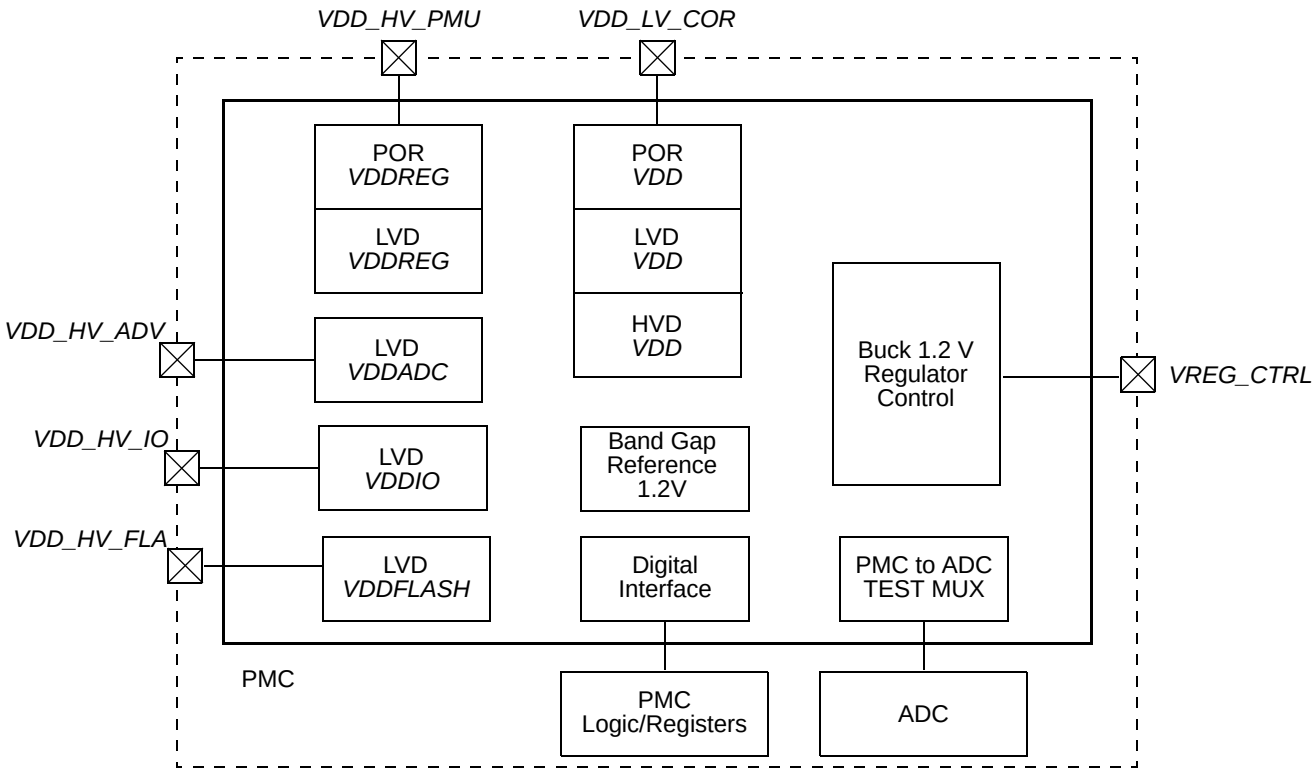


Figure 44-1. PMC block diagram

44.4 External signals description

The PMC external pins are listed below.

- RESET_SUP_B—POR source in external VREG mode
  Not used for internal VREG mode (tie externally to VSS)
- VDD_HV_ADV—ADC voltage 3.3 V supply
- VDD_HV_FLA—Flash memory 3.3 V supply
- VDD_HV_IO—3.3 V input/output supply
- VDD_HV_PMU—Voltage regulator 3.3 V or 5.0 V supply
- VDD_LV_COR—Core logic supply
- VREG_CTRL
  For internal VREG mode, this is the internal VREG output that is used to control the gate of an external SMPS pMOS. Not used for external VREG mode (pull up to VDD_HV_PMU).
- VREG_INT_ENABLE_B—Control signal used by PMU digital block to select VREG mode
  Low level on this pin (VSS) selects internal VREG mode; high level on this pin (VDD_HV_PMU) selects external VREG mode.

Internal and external signal names are provided in Table 44-1.

Table 44-1. External versus internal signal names

External signal (ball name)	Internal signal
VDD_HV_PMU	VDDREG
VDD_LV_COR	VDD
VDD_HV_ADV	VDDADC
VDD_HV_IO	VDDIO
VDD_HV_FLA	VDDFLASH
VREG_CTRL	REGCTL

44.5 Register map

Table 44-2 shows the PMC memory map. The PMC memory maps the following registers for configuring, monitoring, and trimming the LVD monitors.

Table 44-2. PMC memory map

Offset from PMC_BASE (0xC3FE_8080)	Register	Access[1]	Reset Value[2,3]	Location
0x0000	Configuration Register (CFGR)	R/W	0x 000F_0U0U	on page 1513
0x0004	Reserved			
0x0008	Status Register (SR)	R/W	0x8000_0000	on page 1515
0x000C	Reserved			
0x0010	Register for Trim Bits configuration (TRIM1)	R/W	0x103F_3FFF	on page 1517
0x0014	Register for Trim Bits configuration (TRIM2)	R/W	0xFF08_FFFF	on page 1518
0x0018	Register for Trim Bits configuration (TRIM3)	R/W	0x0017_0000	on page 1519
0x001C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 One or more bits (indicated by “U”) are of indeterminate value at Reset. See register for more information.

44.5.1 Configuration Register (CFGR)

The CFGR contains configuration and interrupt enable bits for the LVD monitors.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	REG_TRIM3_LOCK	REG_TRIM2_LOCK	REG_TRIM1_LOCK	0	0	0	REG_CONFIG_LOCK	0	0	0	0	ADC_CHANNEL_SEL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	HVD_1p2_CORE_EN	0	0	0	0	LVD_ADC_3p3_EN	LVD_3p3_MAIN_EN	0	LVD_1p2_CORE_EN	1	1	LVD_3p3_IO_EN	LVD_3p3_FLASH_EN
W																
Reset	0	0	0	x	0	0	0	0	0	0	0	x	1	1	1	1

Figure 44-2. Configuration Register (CFGR)

Table 44-3. CFGR field descriptions

Field	Description
REG_TRIM3_LOCK	TRIM3 Register Write Lock This bit can be written once and used to disable IPS WRITE access to TRIM3 register. 0 IPS WRITE to TRIM3 Register is allowed. 1 IPS WRITE to TRIM3 Register is ignored.
REG_TRIM2_LOCK	TRIM2 Register Write Lock This bit can be written once and used to disable IPS WRITE access to TRIM2 register. 0 IPS WRITE to TRIM2 Register is allowed. 1 IPS WRITE to TRIM2 Register is ignored.
REG_TRIM1_LOCK	TRIM1 Register Write Lock This bit is Once Writable and used to disable IPS WRITE access to TRIM1 register. 0 IPS WRITE to TRIM1 Register is allowed. 1 IPS WRITE to TRIM1 Register is ignored.
REG_CONFIG_LOCK	CONFIG Register Write Lock This bit is Once Writable and used to disable IPS WRITE access to Config register. 0 IPS WRITE to Config Register is allowed. 1 IPS WRITE to Config Register is ignored.
ADC_CHANNEL_SEL	ADC Channel Select Selects ADC channels on which PMU internal high-voltage signals would be muxed for observation.

Table 44-3. CFGR field descriptions (continued)

Field	Description
HVD_1p2_CORE_EN	VDD_LV_COR 1.2 V Core HVD Enable Internal VREG Mode: HVD is always enabled. Writng to this bit has no effect. External VREG Mode: HVD is disabled from reset. HVD can be enabled via IPS WRITE to this bit. PMU Test mode: programmable via this bit. Scan mode: HVD is always disabled. This bit has no effect. 0 HVD is disabled. 1 HVD is enabled.
LVD_ADC3p3_EN	VDD_HV_ADV LVD Enable Internal VREG Mode: LVD is always enabled. Writing to this bit has no effect. External VREG Mode: LVD is always enabled. Writing to this bit has no effect. PMU Test mode: programmable via this bit. Scan mode: LVD is always disabled. This bit has no effect. 0 LVD is disabled. 1 LVD is enabled.
LVD_3p3_MAIN_EN	3.3/5V VDDREG Main LVD Enable Internal VREG Mode: LVD is always enabled. Writing to this bit has no effect. External VREG Mode: LVD is always enabled. Writing to this bit has no effect. PMU Test mode: programmable via this bit. Scan mode: LVD is always disabled. This bit has no effect. 0 LVD is disabled. 1 LVD is enabled.
LVD_1p2_CORE_EN	DD_LV_COR Core LVD Enable Internal VREG Mode: LVD is always enabled. Writing to this bit has no effect. External VREG Mode: LVD is disabled from reset. LVD can be enabled via IPS WRITE to this bit. PMU Test mode: programmable via this bit. Scan mode: LVD is always disabled. This bit has no effect. 0 LVD is disabled. 1 LVD is enabled.
LVD3p3_IO_EN	VDD_HV_IO LVD Enable Internal VREG Mode: LVD is always enabled. Writing to this bit has no effect. External VREG Mode: LVD is always enabled. Writing to this bit has no effect. PMU Test mode: programmable via this bit. Scan mode: LVD is always disabled. This bit has no effect. 0 LVD is disabled. 1 LVD is enabled.
LVD_3p3_FLASH_EN	VDD_HV_FLA LVD Enable Internal VREG Mode: LVD is always enabled. Writing to this bit has no effect. External VREG Mode: LVD is always enabled. Writing to this bit has no effect. PMU Test mode: programmable via this bit. Scan mode: LVD is always disabled. This bit has no effect. 0 LVD is disabled. 1 LVD is enabled.

44.5.2 Status Register (SR)

The SR contains interrupt flag bits for the LVD monitors.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	POR_STATUS	0	0	0	0	0	BANDGAP_RDY	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	HVD_1p2_CORE	0	0	0	0	LVD_ADC_3p3	LVD_3p3_MAIN	0	LVD_1p2_CORE	0	0	LVD3p3_IO	LVD_3p3_FLASH
W				w1c					w1c	w1c		w1c			w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 44-3. Status Register (SR)

Table 44-4. SR field descriptions

Field	Description
POR_STATUS	Indicates occurrence of POR. Write 1 to clear this bit when out of POR.
BANDGAP_RDY	Indicates Bandgap Voltage Status.
HVD_1p2_CORE	Indicates occurrence of 1.2 V Core HVD. Write 1 to clear this bit in case HVD is still not pending.
LVD_ADC_3p3_EN	Indicates occurrence of ADC LVD. Write 1 to clear this bit in case LVD is still not pending.
LVD_3p3_MAIN	Indicates occurrence of 3.3–5 V Main LVD. Write 1 to clear this bit in case LVD is still not pending.
LVD_1p2_CORE	Indicates occurrence of 1.2 V Core LVD. Write 1 to clear this bit in case LVD is still not pending.
LVD_3p3_IO	Indicates occurrence of I/O LVD. Write 1 to clear this bit in case LVD is still not pending.
LVD_3p3_FLASH	Indicates occurrence of Flash LVD. Write 1 to clear this bit in case LVD is still not pending.

44.5.3 Trimming Register 1 (TRIM1)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CLK1p2_TRIM								BANDGAP1p2_ABS_TRIM					
W																
Reset	0	0	0	1	0	0	0	0	0	0	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R			BANDGAP1p2_CRV_TRIM						VREG3p3_TRIM				VREG1p2_TRIM			
W																
Reset	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 44-4. Trimming Register (TRIM1)

Table 44-5. TRIM1 field descriptions

Field	Description
VREG1p2_TRIM	1.2V Core Voltage Regulator Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from Flash NVM, via SSCM Internal Regulation Mode: WRITE to these bits have no effect External Regulation Mode: WRITE to these bits have no effect PMU Test mode: Programmable via IPS WRITE Scan Mode: Set to reset value
VREG3p3_TRIM	3.3-5V Main Voltage Regulator Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from Flash NVM, via SSCM User Functional Mode: WRITE to these bits have no effect PMU Test mode: Programmable via IPS WRITE Scan Mode: Set to reset value
BANDGAP1p2_CRV_TRIM	Bandgap Curvature Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from Flash NVM, via SSCM Internal Regulation Mode: WRITE to these bits have no effect External Regulation Mode: WRITE to these bits have no effect PMU Test mode: Programmable via IPS WRITE Scan Mode: Set to reset value
BANDGAP1p2_ABS_TRIM	Bandgap Absolute Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from Flash NVM, via SSCM Internal Regulation Mode: WRITE to these bits have no effect External Regulation Mode: WRITE to these bits have no effect PMU Test mode: Programmable via IPS WRITE Scan Mode: Set to reset value

Table 44-5. TRIM1 field descriptions (continued)

Field	Description
CLK1p2_TRIM	SMPS Clock Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from Flash NVM, via SSCM Internal Regulation Mode: WRITE to these bits have no effect External Regulation Mode: WRITE to these bits have no effect PMU Test mode: Programmable via IPS WRITE Scan Mode: Set to reset value

44.5.4 Trimming Register 2 (TRIM2)

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVD_3p3_ADC_TRIM				LVD_3p3_MAIN_TRIM				0	0	LVD_1p2_CORE_TRIM					
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	LVD_3p3_IO_TRIM				LVD_3p3_FLASH_TRIM			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 44-5. Trimming Register (TRIM2)

Table 44-6. TRIM2 field descriptions

Field	Description
LVD_3p3_ADC_TRIM	ADC LVD Trim Internal VREG Mode: These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. External VREG Mode: Writing to these bits has no effect. PMU test mode: Programmable via IPS access. Scan mode: Set to reset value.
LVD_3p3_MAIN_TRIM	3.3–5 V Main LVD Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. Internal VREG Mode: Writing to these bits has no effect. External VREG Mode: Writing to these bits has no effect. PMU test mode: Programmable via IPS access. Scan mode: Set to reset value.

Table 44-6. TRIM2 field descriptions (continued)

Field	Description
LVD_1p2_CORE_TRIM	1.2 V Core LVD Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. Internal VREG Mode: Writing to these bits has no effect. External VREG Mode: Programmable via IPS WRITE. PMU test mode: Programmable via IPS access. Scan mode: Set to reset value.
LVD_3p3_IO_TRIM	I/O LVD Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. Internal VREG Mode: Writing to these bits has no effect. External VREG Mode: Writing to these bits has no effect. PMU test mode: Programmable via IPS access. Scan mode: Set to reset value.
LVD_3p3_FLASH_TRIM	Flash LVD Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. Internal VREG Mode: Writing to these bits has no effect. External VREG Mode: Writing to these bits has no effect. PMU test mode: Programmable via IPS access. Scan mode: Set to reset value.

44.5.5 Trimming Register 3 (TRIM3)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	HVD_1p2_CORE_TRIM					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 44-6. Trimming Register 3 (TRIM3)

Table 44-7. TRIM3 field descriptions

Field	Description
HVD_1p2_CORE_TRIM	1.2 V Core HVD Trim These bits are set to their RESET_VALUE at each POR. These bits are loaded with their CHARACTERIZED_DATA from flash memory NVM, via SSCM. Internal VREG Mode: Writing to these bits has no effect. External VREG Mode: Programmable by IPS WRITE. PLL test mode: Trim values programmable via these bits. Scan mode: Set to reset value.

44.6 Power sequencing and startup

PMC supplies can rise in any order. When the internal buck regulator is used to control the VDD core supply, the sequence is as follows:

1. VDDREG rises first.
2. PorReg regulator POR negates.
3. Bandgap reference starts and provides stable reference to LVDs and internal regulator.
4. Internal regulator soft start procedure begins.
5. PorC core POR negates.
6. LVDs are released.
7. Chip is operational.
8. An eventual LVD or HVD event triggers a reset.

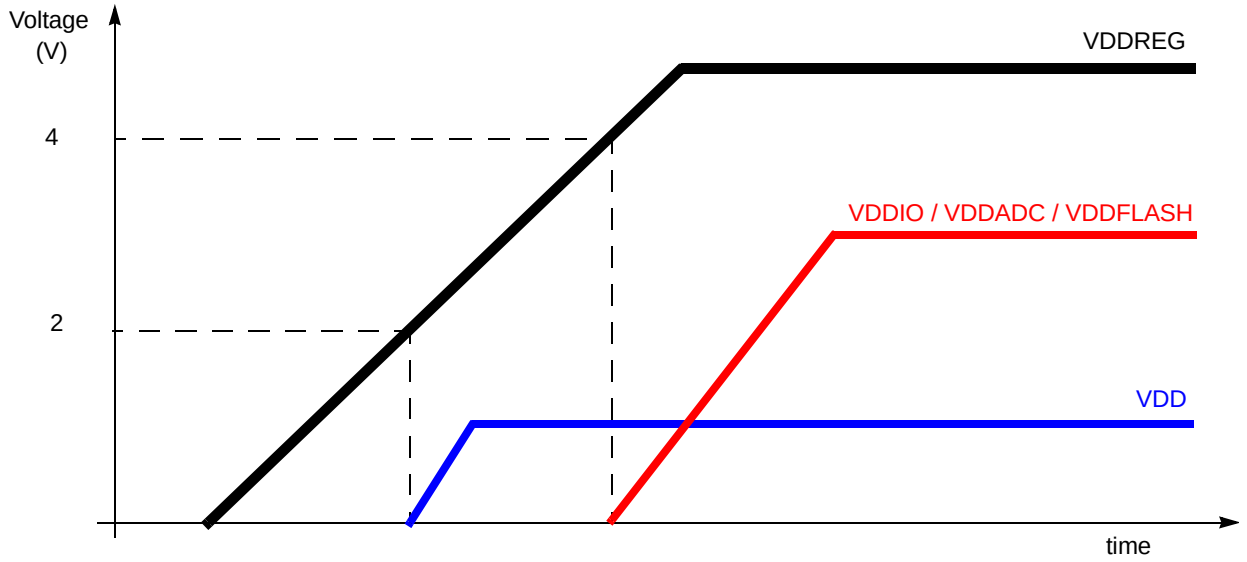


Figure 44-7. PMC internal regulators nominal start-up sequence, rising slope may vary

44.7 VRC SMPS controller

A voltage regulator controller combined with external components as shown in the MPC5675K data sheet might be used to generate the core supply voltage.

The regulator is an asynchronous buck regulator with a nominal switching frequency of 1.1 MHz. It features internal compensation for nominal LC load of 4 µH and 20 µF.

A soft start module reduces the overshoot at startup.

The regulator target voltage is defined in the MPC5675K data sheet and can be programmed via the REG_TRIM register with VddStepC resolution.

44.8 POR on VDD and VDDREG

A power-on reset (POR) circuit monitors its supply voltage, providing a logic reset within the defined low voltage range. Power-on reset asserts as soon as the voltage level of the monitored power supply begins to rise. Each POR negates before its power supply rises into the specified range. The behavior for each POR during power supply ramping is shown in Figure 44-8.

The PORs monitor VDDREG and VDD supplies, with threshold equal to PorReg and PorC, respectively, as specified in the MPC5675K data sheet. When POR is released, all LVDs are in valid state as shown in Figure 44-9. The overlap between POR and LVD enables the correct behavior under all conditions.

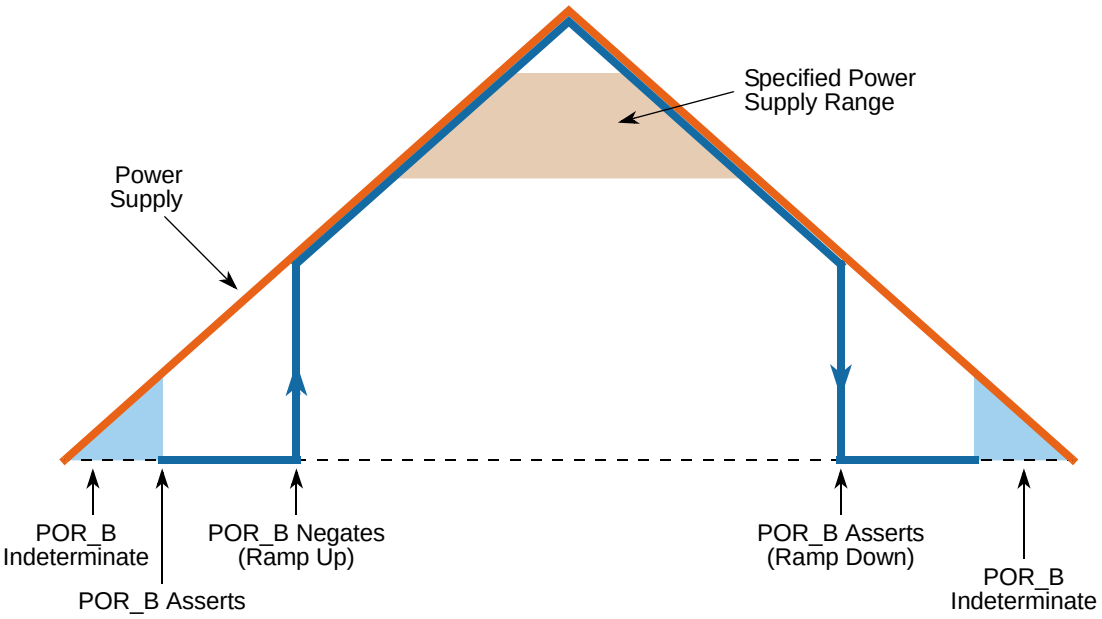


Figure 44-8. POR rising and falling edges

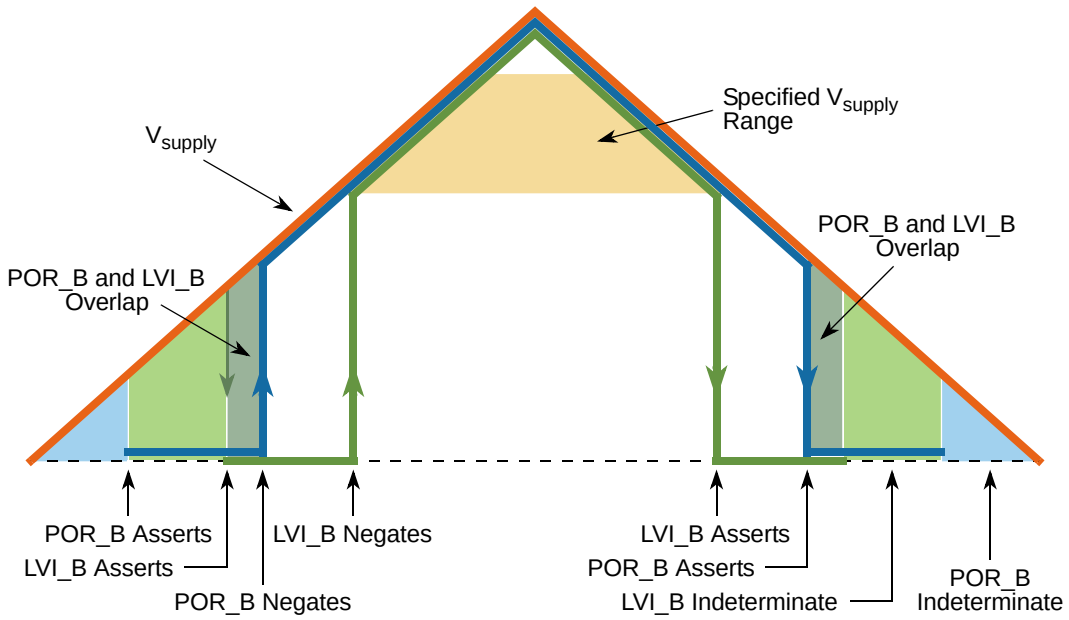


Figure 44-9. POR and LVD rising and falling edges

44.9 LVD core

A programmable low-voltage detector (LVD) monitors the core supply voltage VDD. The LVD threshold voltage is documented under the LvdC symbol in the MPC5675K data sheet.

The LVD features a comparator with hysteresis. It relies on the bandgap reference to determine if the monitored VDD supply is in the correct range. The LVD is asserted until all PORs (core supply VDD and VDDREG) have cleared and until the bandgap reference is stable.

LVD reference voltage can be measured by the ADC by programming the CFGR[ADC_CHANNEL_SEL] field as described in Section 44.12, ADC measure channel. During ADC measurement, the LVD function is disabled.

44.10 HVD core

A programmable high-voltage detector (HVD) monitors the core supply voltage VDD. The HVD threshold voltage is documented under the HvdC symbol in the MPC5675K data sheet.

The HVD features a comparator with hysteresis. It relies on the bandgap reference to determine if the monitored VDD supply is in the correct range. The HVD is asserted until all PORs (core supply VDD and VDDREG) have cleared and until the bandgap reference is stable.

HVD reference voltage can be measured by the ADC by programming the CFGR[ADC_CHANNEL_SEL] field as described in Section 44.12, ADC measure channel. During ADC measurement, the HVD function is disabled.

44.11 LVD 3.3 V

Low-voltage detectors (LVDs) monitor the following supplies:

- VDDADC
- VDDFLASH
- VDDIO
- VDDREG

Rising LVD threshold voltage is documented under the LvdReg symbol in the MPC5675K data sheet.

The LVD features a comparator with hysteresis. It relies on the bandgap reference to determine if the monitored supply is in the correct range. The LVD is asserted until all PORs (core supply VDD and VDDREG) have cleared and until the bandgap reference is stable.

LVD reference voltage can be measured by the ADC by programming the CFGR[ADC_CHANNEL_SEL] field as described in Section 44.12, ADC measure channel. During ADC measurement, the LVD function is disabled.

44.12 ADC measure channel

PMC internal signals are buffered and routed to ADC[0:3] on channel 9.

The signal is selected by programming the CFGR[ADC_CHANNEL_SEL] field. The default field value is 0b1111, which disables the PMC multiplexer and the PMC output buffer.

When a channel is measured, all the LVDs are temporarily disabled in order to avoid false flags caused by the multiplexer switching. LVDs are enabled immediately as CFGR[ADC_CHANNEL_SEL] field is programmed to its default value 0b1111.

ADC mode description:

- 1000: Vbg trimmed bandgap reference
- 1001: LVD core voltage trip point reference
- 1010: HVD core voltage trip point reference
- 1011: Lvd33 ADC scaled supply
- 1100: Lvd33 flash memory scaled supply
- 1101: Lvd33 I/O scaled supply
- 1110: 1.2V SMPS regulator internal supply
- 1111: OFF (default)
- 0*xxx*: Reserved

After measuring the voltage of the 3.3V LVDx, a multiplication factor of 2.3875 can be applied to the result to obtain the actual voltage level.

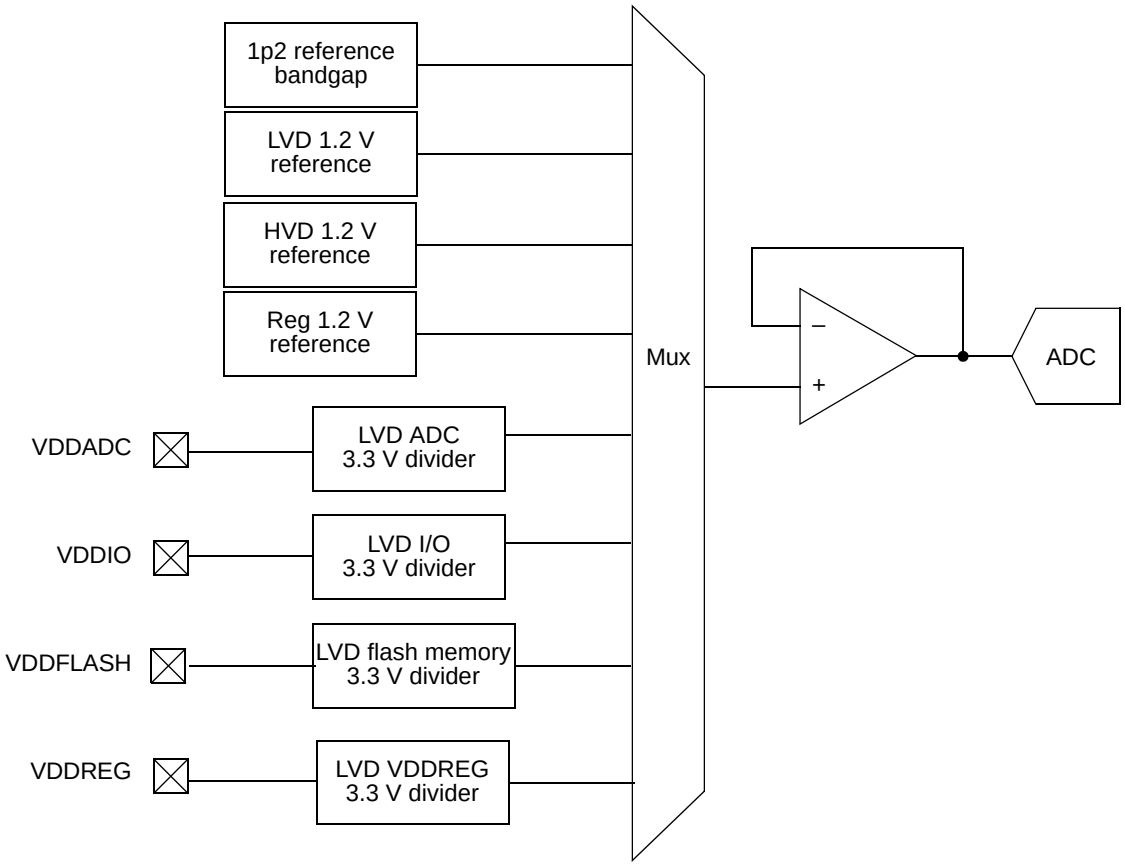


Figure 44-10. ADC test mux block diagram

Chapter 45
Register Protection (REG_PROT)

45.1 Introduction

45.1.1 Overview

The REG_PROT module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The REG_PROT module is located between the module under protection and the PBRIDGE. (For an explanation of the REG_PROT and module base addresses, see Section 45.3, Memory map and register description.) This is shown in Figure 45-1.

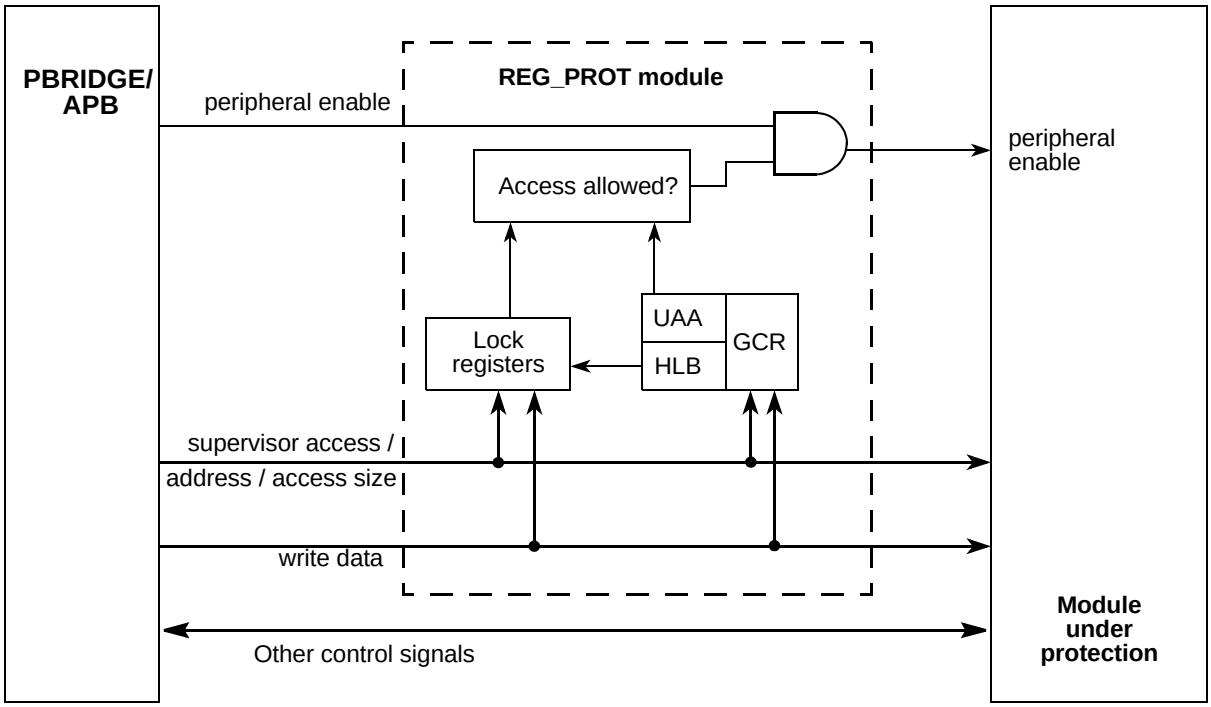


Figure 45-1. REG_PROT block diagram

45.1.2 Features

The REG_PROT module includes these features:

- Restricts write accesses for the module under protection to supervisor mode only
- Locks registers for first 6 KB of memory-mapped address space
- Address mirror automatically sets corresponding lock bit
- Once configured, lock bits can be protected from changes

45.1.3 Modes of operation

The REG_PROT module is operable when the module under protection is operable.

45.2 External signal description

There are no external signals.

45.3 Memory map and register description

This section provides a detailed description of the memory map of a module using the REG_PROT. The original 16 KB module memory space is divided into five areas as shown in Figure 45-2.

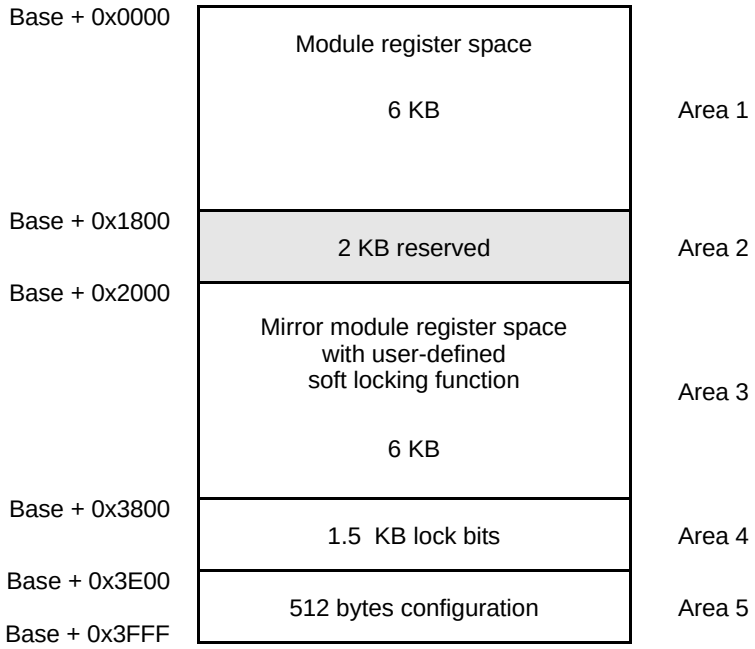


Figure 45-2. REG_PROT memory diagram

Area 1 is 6 KB and holds the normal functional module registers that are accessible for all read/write operations.

Area 2 is 2 KB starting at address 0x1800; it is a reserved area that must not be accessed.

Area 3 is 6 KB, starting at address 0x2000, and is a mirror of area 1. A read/write access to these 0x2000 + X addresses reads or writes the register at address X. As a side effect, a write access to address 0x2000 + X sets the optional Soft Lock bits for this address X in the same cycle as the register at address X is written. Not all registers in area 1 need to have protection defined by associated Soft Lock bits. For unprotected registers at address Y, accesses to address 0x2000 + Y is identical to accesses at address Y. Only registers that are implemented in area 1 and defined as protectable have Soft Lock bits that are available in area 4.

Area 4 is 1.5 KB and holds the Soft Lock bits, one bit per byte in area 1. The four Soft Lock bits associated with one module register word are arranged at byte boundaries in the memory map. The Soft Lock Bit registers can be directly written using a bit mask.

Area 5 is 512 bytes and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module (HLB—Hard Lock Bit) that prevents all further modifications to the Soft Lock bits. Once set, these Hard Lock bits can only be cleared by a system reset. The other bits (UAA—User Access Allowed), if set, allow access to the protected registers in the module.

If any locked byte is accessed with a write transaction, a transfer error is issued to the system and the write transaction is not executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in areas 4 and 5 results in a transfer error.

45.3.1 Memory map

Figure 45-1 gives an overview on the REG_PROT registers implemented.

Table 45-1. REG_PROT memory map

Offset from REG_PROT_BASE (0xFFFF_FFFF)	Register	Access[1]	Reset value[2]	Location
0x0000	Module Register 0 (MR0)	R/W	0x00	on page 1528
0x0001	Module Register 1 (MR1)	R/W	0x00	on page 1528
0x0002	Module Register 2 (MR2)	R/W	0x00	on page 1528
0x0003– 0x17FF	Module Register 3 (MR3)–Module Register 6143 (MR6143)	R/W	0x00	on page 1528
0x1800–0x1FFF	Reserved			
0x2000	Module Register 0 (MR0) + Set Soft Lock Bit 0 (LMR0)			on page 1528
0x2001	Module Register 1 (MR1) + Set Soft Lock Bit 1 (LMR1)			on page 1528
0x2002–0x37FF	Module Register 2 (MR2) + Set Soft Lock Bit 2 (LMR2)–Module Register 6143 (MR6143) + Set Soft Lock Bit 6143 (LMR6143)			on page 1528
0x3800	Soft Lock Bit Register 0 (SLBR0): Soft Lock bits 0-3			on page 1528
0x3801	Soft Lock Bit Register 1 (SLBR1): Soft Lock bits 4-7			on page 1528
0x3802–0x3DFF	Soft Lock Bit Register 2 (SLBR2): Soft Lock bits 8-11 –Soft Lock Bit Register 1535 (SLBR1535): Soft Lock bits 6140-6143			on page 1528
0x3E00–0x3FFB	Reserved			
0x3FFC	Global Configuration Register (GCR)	R/W	0x0000_0000	on page 1529

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the size of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

NOTE

Reserved registers in area 2 are handled according to the protected IP (module under protection).

45.3.2 Register descriptions

This section describes in address order all the REG_PROT registers. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

45.3.2.1 Module registers (MR0–MR6143)

This is the lower 6 KB module memory space that holds all the functional registers of the module that is protected by the REG_PROT module.

45.3.2.2 Module register and set Soft Lock bit (LMR0–LNR6143)

This is memory area 3 that provides mirrored access to the MR0-6143 registers with the side effect of setting Soft Lock bits in case of a write access to an MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in an SLBR*n*[SLB*m*], according to the mapping described in Table 45-2.

45.3.2.3 Soft Lock bit register (SLBR0–SLBR1535)

These registers hold the Soft Lock bits for the protected registers in memory area 1.

Offset: 0x3800-0x3DFF Access: User read-only, Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	SLB0	SLB1	SLB2	SLB3
W	WE0	WE1	WE2	WE3				
Reset	0	0	0	0	0	0	0	0

Figure 45-3. Soft lock bit register (SLBR*n*)

Table 45-2. SLBR*n* field descriptions

Field	Description
WE0 WE1 WE2 WE3	Write Enable Bits for Soft Lock bits (SLB). WE0 enables writing to SLB0 WE1 enables writing to SLB1 WE2 enables writing to SLB2 WE3 enables writing to SLB3 0 SLB is not modified. 1 Value is written to SLB.
SLB0 SLB1 SLB2 SLB3	Soft Lock bits for one MR*n* register. SLB0 can block accesses to MR{*n* × 4 + 0} SLB1 can block accesses to MR{*n* × 4 + 1} SLB2 can block accesses to MR{*n* × 4 + 2} SLB3 can block accesses to MR{*n* × 4 + 3} 0 Associated MR*n* byte is unprotected and writable. 1 Associated MR*n* byte is locked against write accesses.

Table 45-3 gives some examples how SLBR*n*[SLB] and MR*n* go together.

Table 45-3. Soft lock bits vs. protected address

Soft lock bit	Protected address
SLBR0[SLB0]	MR0
SLBR0[SLB1]	MR1
SLBR0[SLB2]	MR2
SLBR0[SLB3]	MR3
SLBR1[SLB0]	MR4
SLBR1[SLB1]	MR5
SLBR1[SLB2]	MR6
SLBR1[SLB3]	MR7
SLBR2[SLB0]	MR8
...	...

45.3.2.4 Global configuration register (GCR)

This register makes global configurations related to the REG_PROT.

Address: Base + 0x3FFC Access: Supervisor read/write, User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HLB	0	0	0	0	0	0	0	UAA	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 45-4. Global configuration register (GCR)

Table 45-4. GCR field descriptions

Field	Description
HLB	Hard Lock Bit. This register cannot be cleared once it is set by software. It can only be cleared by a system reset. 0 All SLB bits are accessible and can be modified. 1 All SLB bits are write protected and cannot be modified.
UAA	User Access Allowed. 0 The registers in the module under protection can only be written in supervisor mode. All write accesses in non-supervisor mode are not executed and a transfer error is issued. This access restriction is in addition to any access restrictions imposed by the protected IP module. 1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions.

NOTE

The GCR[UAA] bit has no effect on the allowed access modes for the registers in the REG_PROT module.

45.4 Functional description

45.4.1 General

This module provides a generic register (address) write-protection mechanism. The protection size can be:

- 32-bit (address == multiples of 4)
- 16-bit (address == multiples of 2)
- 8-bit (address == multiples of 1)
- Unprotected (address == multiples of 1)

The list of protected modules and registers is shown in Section 45.6, MPC5675K registers under protection.

For all addresses that are protected there are SLBR*n*[SLB*m*] bits that specify whether the address is locked. When an address is locked it can only be read but not written in any mode (supervisor/normal). If an

address is unprotected, the corresponding SLBR*n*[SLB*m*] bit is always 0b0 no matter what value software attempts to write to the bit.

45.4.2 Change lock settings

To change the setting whether an address is locked or unlocked, the corresponding SLBR*n*[SLB*m*] bit needs to be changed. This can be done using the following methods:

- Modify the SLBR*n*[SLB*m*] directly by writing to area 4
- Set the SLBR*n*[SLB*m*] bit(s) by writing to the mirror module space (area 3)

Both methods are explained in the following sections.

45.4.2.1 Change lock settings directly via area 4

Memory area 4 contains the lock bits. They can be modified by writing to them. Each SLBR*n*[SLB*m*] bit has a mask bit SLBR*n*[WE*m*] that protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 45-5 shows two modification examples. In the left example, there is a write access to the SLBR*n* register specifying a mask value that allows modification of all SLBR*n*[SLB*m*] bits. The example on the right specifies a mask that only allows modification of the bits SLBR*n*[SLB{3:1}].

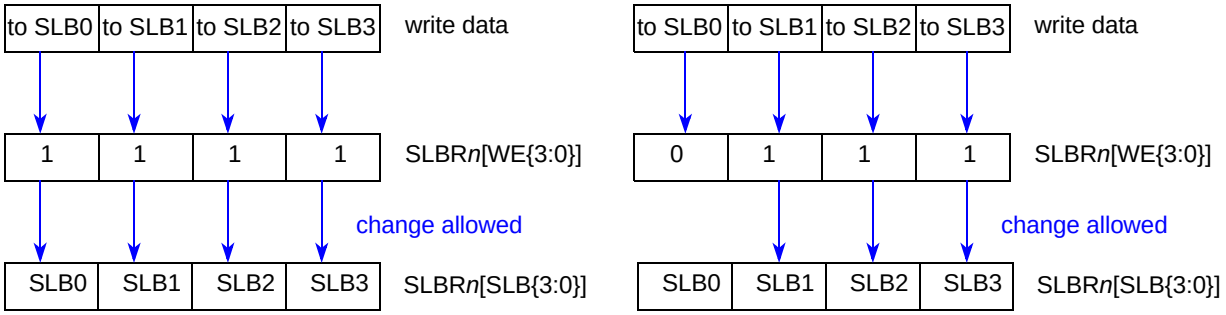


Figure 45-5. Change lock settings directly via area 4

Figure 45-5 shows four registers that can be protected with 8-bit protection. Registers with 16- and 32- bit protection are shown in Figure 45-6 and Figure 45-7, respectively.

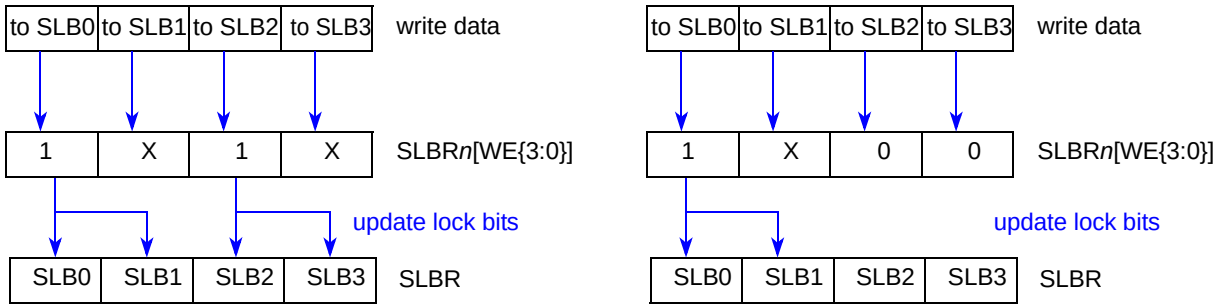


Figure 45-6. Change lock settings for 16-bit protected addresses

On the right side of Figure 45-6 you can see that the data written to SLBR*n*[SLB{0}] is automatically written to SLBR*n*[SLB{1}] as well. This is done because the address reflected by SLBR*n*[SLB{0}] has 16-bit protection. Note that in this case the write enable SLBR*n*[WE{0}] must be set while SLBR*n*[WE{1}] does not matter. As the enable bits SLBR*n*[WE{3:2}] are cleared the lock bits SLBR*n*[SLB{3:2}] remain unchanged.

In the example on the left side of Figure 45-6, the data written to SLBR*n*[SLB{0}] is mirrored to SLBR*n*[SLB{1}] and the data written to SLBR*n*[SLB{2}] is mirrored to SLBR*n*[SLB{3}], because the write enables are set for both registers .

Figure 45-7 shows a register with 32-bit protection. When SLBR*n*[WE{0}] is set, the data written to SLBR*n*[SLB{0}] is automatically written to SLBR*n*[SLB{3:1}] also. Otherwise SLBR*n*[SLB{3:0}] remains unchanged.

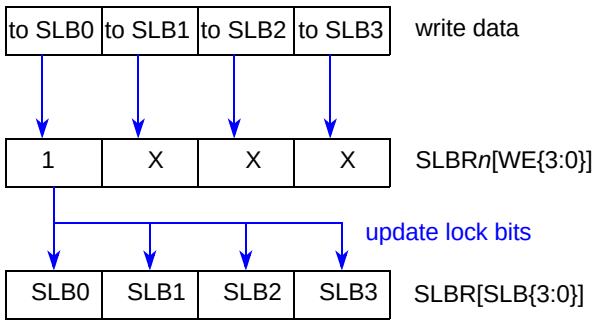


Figure 45-7. Change lock settings for 32-bit protected addresses

Figure 45-8 shows an example of mixed protection size configuration.

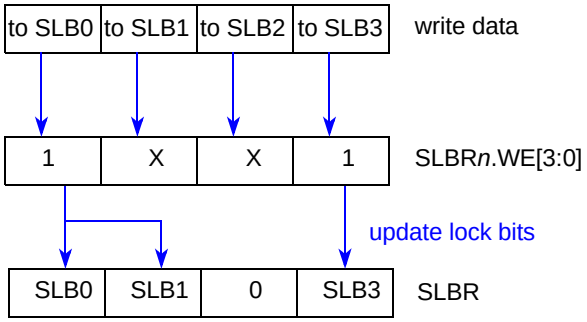


Figure 45-8. Change lock settings for mixed protection

The data written to SLBR*n*[SLB{0}] is mirrored to SLBR*n*[SLB{1}] as the corresponding register is 16-bit protected. The data written to SLBR*n*[SLB{2}] is blocked as the corresponding register is unprotected. The data written to SLBR*n*[SLB{3}] is written to SLBR*n*[SLB{3}].

45.4.2.2 Enable locking via mirror module space (area 3)

It is possible to enable locking for a register after writing to it. To do so, you must use the mirrored module address space. Figure 45-9 shows one example.

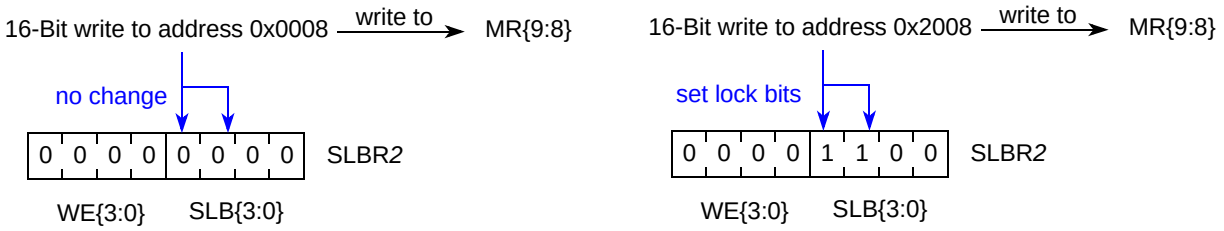


Figure 45-9. Enable locking via mirror module space (area 3)

When writing to address 0x0008, the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits remain unchanged (left part of Figure 45-6).

When writing to address 0x2008, the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits SLBR2[SLB{1:0}] are set while the lock bits SLBR2[SLB{3:2}] remain unchanged (right part of Figure 45-6).

Figure 45-10 shows an example where some addresses are protected and some are not:

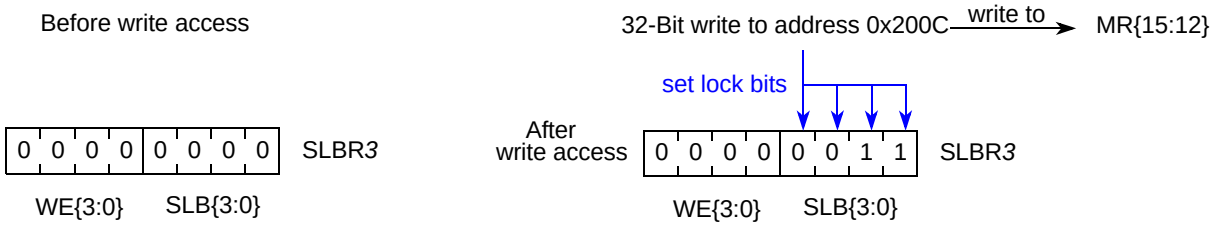


Figure 45-10. Enable locking for protected and unprotected addresses

In the example in Figure 45-10 addresses 0x0C and 0x0D are unprotected. Therefore, their corresponding lock bits SLBR3[SLB{1:0}] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3[SLB{3:2}] are set while bits SLBR3[SLB{1:0}] stay cleared.

NOTE

Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space do not change the lock bits.

45.4.2.3 Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in Section 45.4.2.1, Change lock settings directly via area 4, and Section 45.4.2.2, Enable locking via mirror module space (area 3), is only possible as long as the bit GCR[HLB] is cleared. Once this bit is set the locking bits can no longer be modified until a system reset occurs.

45.4.3 Access errors

The REG_PROT module generates transfer errors under several circumstances. For the area definition refer to Figure 45-2.

1. If accessing area 1 or area 3, the REG_PROT module passes on any access error from the underlying protected module.

2. If user mode is not allowed, user writes to all areas assert a transfer error and the writes are blocked.
3. If accessing the reserved area 2, a transfer error is asserted.
4. If accessing unimplemented 32-bit registers in area 4 and area 5, a transfer error is asserted.
5. If writing to a register in area 1 and area 3 with Soft Lock Bit set for any of the affected bytes, a transfer error is asserted and the write is blocked. Also, the complete write operation to non-protected bytes in this word is ignored.
6. If writing to a Soft Lock Register in area 4 with the Hard Lock Bit set, a transfer error is asserted.
7. Any write operation occurs in any access mode to area 3 while Hard Lock Bit GCR[HLB] is set.

45.5 Initialization/application information

45.5.1 Reset

The reset state of each individual bit is shown within the register description section (see Section 45.3.2, Register descriptions). In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in supervisor mode only.

45.5.2 Writing C code using the register protection scheme

A set of macros provided as part of the device's header file defines the memory map, the peripheral registers, and the fields of these registers. These macros are intended to make working with the register protection scheme easier for the developers of device driver software and/or other application code. This section describes these macros and how to use them.

A first macro is made available to perform a write to the mirrored module register space (area 3). As described in this document, this results in concurrently setting the corresponding Soft Lock bit, while writing to the module register. There are three flavors of this macro to account for the different sizes of the related module register `<thereg>` (the value of `<size>` is 8, 16, or 32 bits):

```
WRITE_WITH_LOCK<size>(<thereg>, <newvalue>)
```

This macro writes the value `<newvalue>` into the register <thereg> assuming a register size of `<size>`. The parameter `<thereg>` must be the name of a register using the notation in the device's header file.

A second set of macros is made available to work with the Soft Lock bits provided by the register protection scheme. The value of these bits associated with a particular module register `<thereg>` can be retrieved, set, and cleared. The macros provided for this purpose are:

```
GET_SOFTLOCK(<thereg>,<dest>)
SET_SOFTLOCK<size>(<thereg>)
CLR_SOFTLOCK<size>(<thereg>)
```

The macro GET_SOFTLOCK retrieves the value of the Soft Lock bit register associated with the given register `<thereg>` and stores it in the variable `<dest>`, which is always an 8-bit value. The other two macros (SET_SOFTLOCK, CLR_SOFTLOCK) set or clear the Soft Lock bits associated with the given register `<thereg>`, assuming this register has a size of 8, 16, or 32 bits as indicated by the macro name. For all three macros, the parameter `<thereg>` must be the name of a register using the notation in the device's header file.

Three more macros are made available to modify bits in the Global Configuration Register (GCR) of the protection gasket for the corresponding block. In contrast to the earlier ones, these macros receive the base address of the corresponding block (usually named <block_name>_BASEADDRESS) as a parameter.

```
SET_HARDLOCK(base)
```

Sets the Hard Lock Bit HLB in the GCR register associated with the module identified by the given base address <base>.

```
USER_ACCESS_FORBIDDEN(base)
```

Clears the User Access Allowed Bit UAA in the GCR register associated with the module identified by the given base address <base>. When cleared, this bit denies any write access to the protected module in user mode and generates a transfer error in case of an attempt to write a protected register.

```
USER_ACCESS_ALLOWED(base)
```

Sets the User Access Allowed Bit UAA in the GCR register associated with the module identified by the given base address <base>. When set, this bit permits write accesses to the protected module in user mode when the corresponding register is not additionally protected by other means.

Finally, two more macros are provided to permit direct access to the registers in the register protection gasket using the same semantic as other register definitions in the header file:

```
LOCK_SLB(thereg)
```

provides the content of the Soft Lock bit register associated with the register <thereg>; the parameter <thereg> must be the name of the corresponding module register using the notation in the device's header file.

```
LOCK_GCR(base)
```

provides the content of the Global Configuration Register GCR for the block identified by its base address <base>; the parameter <base> must be the base address of the corresponding block.

45.6 MPC5675K registers under protection

For registers not under protection from a non-supervisor write access (for example, EBI[MCR] and I2C*n*[IBAD]) (of all the I^2C), the user can verify register contents with a periodic CRC to ensure that the register's values have not been altered during code execution.

Table 45-5. MPC5675K register protection

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
Code flash memory				
Code Flash	MCR	32	0x0000	32
Code Flash	LML	32	0x0004	32
Code Flash	HBL	32	0x0008	32
Code Flash	SLL	32	0x000C	32
Code Flash	BIU0	32	0x001C	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
Code Flash	BIU1	32	0x0020	32
Code Flash	BIU2	32	0x0024	32
Code Flash	UT0	32	0x003C	32
Code Flash	UT1	32	0x0040	32
Code Flash	UT2	32	0x0044	32
Code Flash	UMISR0	32	0x0048	32
Code Flash	UMISR1	32	0x004C	32
Code Flash	UMISR2	32	0x0050	32
Code Flash	UMISR3	32	0x0054	32
Code Flash	UMISR4	32	0x0058	32
Data flash memory				
Data Flash	MCR	32	0x0000	32
Data Flash	LML	32	0x0004	32
Data Flash	SLL	32	0x000C	32
Data Flash	UT0	32	0x003C	32
Data Flash	UT1	32	0x0040	32
Data Flash	UMISR0	32	0x0048	32
Data Flash	UMISR1	32	0x004C	32
SIUL				
SIUL	IRER	32	0x0018	32
SIUL	IREER	32	0x0028	32
SIUL	IFEER	32	0x002C	32
SIUL	IFER	32	0x0030	32
SIUL	PCR0	16	0x0040	16
SIUL	PCR1	16	0x0042	16
SIUL	PCR2	16	0x0044	16
SIUL	PCR3	16	0x0046	16
SIUL	PCR4	16	0x0048	16
SIUL	PCR5	16	0x004A	16
SIUL	PCR6	16	0x004C	16
SIUL	PCR7	16	0x004E	16
SIUL	PCR8	16	0x0050	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR9	16	0x0052	16
SIUL	PCR10	16	0x0054	16
SIUL	PCR11	16	0x0056	16
SIUL	PCR12	16	0x0058	16
SIUL	PCR13	16	0x005A	16
SIUL	PCR14	16	0x005C	16
SIUL	PCR15	16	0x005E	16
SIUL	PCR16	16	0x0060	16
SIUL	PCR17	16	0x0062	16
SIUL	PCR18	16	0x0064	16
SIUL	PCR19	16	0x0066	16
SIUL	PCR20	16	0x0068	16
SIUL	PCR21	16	0x006A	16
SIUL	PCR22	16	0x006C	16
SIUL	PCR23	16	0x006E	16
SIUL	PCR24	16	0x0070	16
SIUL	PCR25	16	0x0072	16
SIUL	PCR26	16	0x0074	16
SIUL	PCR27	16	0x0076	16
SIUL	PCR28	16	0x0078	16
SIUL	PCR29	16	0x007A	16
SIUL	PCR30	16	0x007C	16
SIUL	PCR31	16	0x007E	16
SIUL	PCR32	16	0x0080	16
SIUL	PCR33	16	0x0082	16
SIUL	PCR34	16	0x0084	16
SIUL	PCR36	16	0x0088	16
SIUL	PCR37	16	0x008A	16
SIUL	PCR38	16	0x008C	16
SIUL	PCR39	16	0x008E	16
SIUL	PCR42	16	0x0094	16
SIUL	PCR43	16	0x0096	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR44	16	0x0098	16
SIUL	PCR45	16	0x009A	16
SIUL	PCR46	16	0x009C	16
SIUL	PCR47	16	0x009E	16
SIUL	PCR48	16	0x00A0	16
SIUL	PCR49	16	0x00A2	16
SIUL	PCR50	16	0x00A4	16
SIUL	PCR51	16	0x00A6	16
SIUL	PCR52	16	0x00A8	16
SIUL	PCR53	16	0x00AA	16
SIUL	PCR54	16	0x00AC	16
SIUL	PCR55	16	0x00AE	16
SIUL	PCR56	16	0x00B0	16
SIUL	PCR57	16	0x00B2	16
SIUL	PCR58	16	0x00B4	16
SIUL	PCR59	16	0x00B6	16
SIUL	PCR60	16	0x00B8	16
SIUL	PCR62	16	0x00BC	16
SIUL	PCR64	16	0x00C0	16
SIUL	PCR66	16	0x00C4	16
SIUL	PCR68	16	0x00C8	16
SIUL	PCR69	16	0x00CA	16
SIUL	PCR70	16	0x00CC	16
SIUL	PCR71	16	0x00CE	16
SIUL	PCR73	16	0x00D2	16
SIUL	PCR74	16	0x00D4	16
SIUL	PCR75	16	0x00D6	16
SIUL	PCR76	16	0x00D8	16
SIUL	PCR78	16	0x00DC	16
SIUL	PCR80	16	0x00E0	16
SIUL	PCR84	16	0x00E8	16
SIUL	PCR85	16	0x00EA	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR86	16	0x00EC	16
SIUL	PCR87	16	0x00EE	16
SIUL	PCR88	16	0x00F0	16
SIUL	PCR89	16	0x00F2	16
SIUL	PCR90	16	0x00F4	16
SIUL	PCR91	16	0x00F6	16
SIUL	PCR92	16	0x00F8	16
SIUL	PCR93	16	0x00FA	16
SIUL	PCR94	16	0x00FC	16
SIUL	PCR95	16	0x00FE	16
SIUL	PCR98	16	0x0104	16
SIUL	PCR99	16	0x0106	16
SIUL	PCR100	16	0x0108	16
SIUL	PCR101	16	0x010A	16
SIUL	PCR102	16	0x010C	16
SIUL	PCR103	16	0x010E	16
SIUL	PCR108	16	0x0118	16
SIUL	PCR109	16	0x011A	16
SIUL	PCR110	16	0x011C	16
SIUL	PCR111	16	0x011E	16
SIUL	PCR112	16	0x0120	16
SIUL	PCR113	16	0x0122	16
SIUL	PCR114	16	0x0124	16
SIUL	PCR115	16	0x0126	16
SIUL	PCR116	16	0x0128	16
SIUL	PCR117	16	0x012A	16
SIUL	PCR118	16	0x012C	16
SIUL	PCR119	16	0x012E	16
SIUL	PCR120	16	0x0130	16
SIUL	PCR121	16	0x0132	16
SIUL	PCR122	16	0x0134	16
SIUL	PCR123	16	0x0136	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR124	16	0x0138	16
SIUL	PCR125	16	0x013A	16
SIUL	PCR126	16	0x013C	16
SIUL	PCR127	16	0x013E	16
SIUL	PCR128	16	0x0140	16
SIUL	PCR129	16	0x0142	16
SIUL	PCR130	16	0x0144	16
SIUL	PCR131	16	0x0146	16
SIUL	PCR132	16	0x014 8	16
SIUL	PCR133	16	0x014A	16
SIUL	PCR134	16	0x014C	16
SIUL	PCR135	16	0x014E	16
SIUL	PCR136	16	0x0150	16
SIUL	PCR137	16	0x0152	16
SIUL	PCR138	16	0x0154	16
SIUL	PCR139	16	0x0156	16
SIUL	PCR140	16	0x0158	16
SIUL	PCR141	16	0x015A	16
SIUL	PCR142	16	0x015C	16
SIUL	PCR143	16	0x015E	16
SIUL	PCR144	16	0x0160	16
SIUL	PCR145	16	0x0162	16
SIUL	PCR146	16	0x0164	16
SIUL	PCR147	16	0x0166	16
SIUL	PCR148	16	0x0168	16
SIUL	PCR149	16	0x016A	16
SIUL	PCR150	16	0x016C	16
SIUL	PCR151	16	0x016E	16
SIUL	PCR152	16	0x0170	16
SIUL	PCR153	16	0x0172	16
SIUL	PCR154	16	0x0174	16
SIUL	PCR155	16	0x0176	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR156	16	0x0178	16
SIUL	PCR157	16	0x017A	16
SIUL	PCR158	16	0x017C	16
SIUL	PCR159	16	0x017E	16
SIUL	PCR160	16	0x0180	16
SIUL	PCR161	16	0x0182	16
SIUL	PCR162	16	0x0184	16
SIUL	PCR163	16	0x0186	16
SIUL	PCR164	16	0x0188	16
SIUL	PCR165	16	0x018A	16
SIUL	PCR166	16	0x018C	16
SIUL	PCR167	16	0x018E	16
SIUL	PCR168	16	0x0190	16
SIUL	PCR169	16	0x0192	16
SIUL	PCR170	16	0x0194	16
SIUL	PCR171	16	0x0196	16
SIUL	PCR172	16	0x0198	16
SIUL	PCR173	16	0x019A	16
SIUL	PCR174	16	0x019C	16
SIUL	PCR175	16	0x019E	16
SIUL	PCR176	16	0x01A0	16
SIUL	PCR177	16	0x01A2	16
SIUL	PCR178	16	0x01A4	16
SIUL	PCR179	16	0x01A6	16
SIUL	PCR180	16	0x01A8	16
SIUL	PCR181	16	0x01AA	16
SIUL	PCR182	16	0x01AC	16
SIUL	PCR183	16	0x01AE	16
SIUL	PCR184	16	0x01B0	16
SIUL	PCR185	16	0x01B2	16
SIUL	PCR186	16	0x01B4	16
SIUL	PCR187	16	0x01B6	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR188	16	0x01B8	16
SIUL	PCR189	16	0x01BA	16
SIUL	PCR190	16	0x01BC	16
SIUL	PCR191	16	0x01BE	16
SIUL	PCR192	16	0x01C0	16
SIUL	PCR193	16	0x01C2	16
SIUL	PCR194	16	0x01C4	16
SIUL	PCR195	16	0x01C6	16
SIUL	PCR196	16	0x01C8	16
SIUL	PCR197	16	0x01CA	16
SIUL	PCR198	16	0x01CC	16
SIUL	PCR199	16	0x01CE	16
SIUL	PCR200	16	0x01D0	16
SIUL	PCR201	16	0x01D2	16
SIUL	PCR202	16	0x01D4	16
SIUL	PCR203	16	0x01D6	16
SIUL	PCR204	16	0x01D8	16
SIUL	PCR205	16	0x01DA	16
SIUL	PCR206	16	0x01DC	16
SIUL	PCR207	16	0x01DE	16
SIUL	PCR208	16	0x01E0	16
SIUL	PCR209	16	0x01E2	16
SIUL	PCR210	16	0x01E4	16
SIUL	PCR211	16	0x01E6	16
SIUL	PCR212	16	0x01E8	16
SIUL	PCR213	16	0x01EA	16
SIUL	PCR214	16	0x01EC	16
SIUL	PCR215	16	0x01EE	16
SIUL	PCR216	16	0x01F0	16
SIUL	PCR217	16	0x01F2	16
SIUL	PCR218	16	0x01F4	16
SIUL	PCR219	16	0x01F6	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PCR220	16	0x01F8	16
SIUL	PCR221	16	0x01FA	16
SIUL	PCR222	16	0x01FC	16
SIUL	PCR223	16	0x01FE	16
SIUL	PCR224	16	0x0200	16
SIUL	PCR225	16	0x0202	16
SIUL	PCR226	16	0x0204	16
SIUL	PCR227	16	0x0206	16
SIUL	PCR228	16	0x0208	16
SIUL	PCR229	16	0x020A	16
SIUL	PCR230	16	0x020C	16
SIUL	PCR231	16	0x020E	16
SIUL	PCR232	16	0x0210	16
SIUL	PCR233	16	0x0212	16
SIUL	PCR234	16	0x0214	16
SIUL	PSMI0_3	32	0x0500	32
SIUL	PSMI4_7	32	0x0504	32
SIUL	PSMI8_11	32	0x0508	32
SIUL	PSMI12_15	32	0x050C	32
SIUL	PSMI16_19	32	0x0510	32
SIUL	PSMI20_23	32	0x0514	32
SIUL	PSMI24_27	32	0x0518	32
SIUL	PSMI28_31	32	0x051C	32
SIUL	PSMI32_35	32	0x0520	32
SIUL	PSMI36_39	32	0x0524	32
SIUL	PSMI40_43	32	0x0526	32
SIUL	PSMI44_47	32	0x052C	32
SIUL	PSMI48_51	32	0x0530	32
SIUL	PSMI52_55	32	0x0534	32
SIUL	PSMI56_59	32	0x0538	32
SIUL	PSMI60	32	0x053C	32
SIUL	GPDO0_3	32	0x0600	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	GPDO4_7	32	0x0604	32
SIUL	GPDO8_11	32	0x0608	32
SIUL	GPDO12_15	32	0x060C	32
SIUL	GPDO16_19	32	0x0610	32
SIUL	GPDO20_23	32	0x0614	32
SIUL	GPDO24_27	32	0x0618	32
SIUL	GPDO28_31	32	0x061C	32
SIUL	GPDO32_35	32	0x0620	32
SIUL	GPDO36_39	32	0x0624	32
SIUL	GPDO40_43	32	0x0628	32
SIUL	GPDO44_47	32	0x062C	32
SIUL	GPDO48_51	32	0x0630	32
SIUL	GPDO52_55	32	0x0634	32
SIUL	GPDO56_59	32	0x0638	32
SIUL	GPDO60_63	32	0x063C	32
SIUL	GPDO64_67	32	0x0640	32
SIUL	GPDO68_71	32	0x0644	32
SIUL	GPDO72_75	32	0x0648	32
SIUL	GPDO76_79	32	0x064C	32
SIUL	GPDO80_83	32	0x0650	32
SIUL	GPDO84_87	32	0x0654	32
SIUL	GPDO88_91	32	0x0658	32
SIUL	GPDO92_95	32	0x065C	32
SIUL	GPDO96_99	32	0x0660	32
SIUL	GPDO100_103	32	0x0664	32
SIUL	GPDO108_111	32	0x066C	32
SIUL	GPDO112_115	32	0x0670	32
SIUL	GPDO116_119	32	0x0674	32
SIUL	GPDO120_123	32	0x0678	32
SIUL	GPDO124_127	32	0x067C	32
SIUL	GPDO128_131	32	0x0680	32
SIUL	GPDO132_135	32	0x0684	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	GPDO136_139	32	0x0688	32
SIUL	GPDO140_143	32	0x068C	32
SIUL	GPDO144_147	32	0x0690	32
SIUL	GPDO148_151	32	0x0694	32
SIUL	GPDO152_155	32	0x0698	32
SIUL	GPDO156_159	32	0x069C	32
SIUL	GPDO160_163	32	0x06A0	32
SIUL	GPDO164_167	32	0x06A4	32
SIUL	GPDO168_171	32	0x06A8	32
SIUL	GPDO172_175	32	0x06AC	32
SIUL	GPDO176_179	32	0x06B0	32
SIUL	GPDO180_183	32	0x06B4	32
SIUL	GPDO184_187	32	0x06B8	32
SIUL	GPDO188_191	32	0x06BC	32
SIUL	GPDO192_195	32	0x06C0	32
SIUL	GPDO196_199	32	0x06C4	32
SIUL	GPDO200_203	32	0x06C8	32
SIUL	GPDO204_207	32	0x06CC	32
SIUL	GPDO208_211	32	0x06D0	32
SIUL	GPDO212_215	32	0x06D4	32
SIUL	GPDO216_219	32	0x06D8	32
SIUL	GPDO220_223	32	0x06DC	32
SIUL	GPDO224_227	32	0x06E0	32
SIUL	GPDO228_231	32	0x06E4	32
SIUL	GPDO232_233	32	0x06E8	32
SIUL	PGPDO0	32	0x0C00	32
SIUL	PGPDO1	32	0x0C04	32
SIUL	PGPDO2	32	0x0C08	32
SIUL	PGPDO3	32	0x0C0C	32
SIUL	PGPDO4	32	0x0C10	32
SIUL	PGPDO5	32	0x0C14	32
SIUL	PGPDO6	32	0x0C18	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	PGPDO7	32	0x0C1C	32
SIUL	IFMC0	32	0x1000	32
SIUL	IFMC1	32	0x1004	32
SIUL	IFMC2	32	0x1008	32
SIUL	IFMC3	32	0x100C	32
SIUL	IFMC4	32	0x1010	32
SIUL	IFMC5	32	0x1014	32
SIUL	IFMC6	32	0x1018	32
SIUL	IFMC7	32	0x101C	32
SIUL	IFMC8	32	0x1020	32
SIUL	IFMC9	32	0x1024	32
SIUL	IFMC10	32	0x1028	32
SIUL	IFMC11	32	0x102C	32
SIUL	IFMC12	32	0x1030	32
SIUL	IFMC13	32	0x1034	32
SIUL	IFMC14	32	0x1038	32
SIUL	IFMC15	32	0x103C	32
SIUL	IFMC16	32	0x1040	32
SIUL	IFMC17	32	0x1044	32
SIUL	IFMC18	32	0x1048	32
SIUL	IFMC19	32	0x104C	32
SIUL	IFMC20	32	0x1050	32
SIUL	IFMC21	32	0x1054	32
SIUL	IFMC22	32	0x1058	32
SIUL	IFMC23	32	0x105C	32
SIUL	IFMC24	32	0x1060	32
SIUL	IFMC25	32	0x1064	32
SIUL	IFMC26	32	0x1068	32
SIUL	IFMC27	32	0x106C	32
SIUL	IFMC28	32	0x1070	32
SIUL	IFMC29	32	0x1074	32
SIUL	IFMC30	32	0x1078	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
SIUL	IFMC31	32	0x107C	32
SIUL	IFCP	32	0x1080	32
MC_ME				
MC ME	ME	32	0x0008	32
MC_ME	IM	32	0x0010	32
MC ME	TEST_MC	32	0x0024	32
MC ME	SAFE_MC	32	0x0028	32
MC ME	DRUN_MC	32	0x002C	32
MC ME	RUN0_MC	32	0x0030	32
MC ME	RUN1_MC	32	0x0034	32
MC ME	RUN2_MC	32	0x0038	32
MC ME	RUN3_MC	32	0x003C	32
MC_ME	HALT0_MC	32	0x0040	32
MC_ME	STOP0_MC	32	0x0048	32
MC_ME	RUN_PC0	32	0x0080	32
MC_ME	RUN_PC1	32	0x0084	32
MC_ME	RUN_PC2	32	0x0088	32
MC_ME	RUN_PC3	32	0x008C	32
MC_ME	RUN_PC4	32	0x0090	32
MC_ME	RUN_PC5	32	0x0094	32
MC_ME	RUN_PC6	32	0x0098	32
MC_ME	RUN_PC7	32	0x009C	32
MC_ME	LP_PC0	32	0x00A0	32
MC_ME	LP_PC1	32	0x00A4	32
MC_ME	LP_PC2	32	0x00A8	32
MC_ME	LP_PC3	32	0x00AC	32
MC_ME	LP_PC4	32	0x00B0	32
MC_ME	LP_PC5	32	0x00B4	32
MC_ME	LP_PC6	32	0x00B8	32
MC_ME	LP_PC7	32	0x00BC	32
MC_ME	PCTL4	8	0x00C4	8
MC_ME	PCTL5	8	0x00C5	8

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
MC_ME	PCTL6	8	0x00C6	8
MC_ME	PCTL16	8	0x00D0	8
MC_ME	PCTL17	8	0x00D1	8
MC_ME	PCTL18	8	0x00D2	8
MC_ME	PCTL19	8	0x00D3	8
MC_ME	PCTL24	8	0x00D8	8
MC_ME	PCTL32	8	0x00E0	8
MC_ME	PCTL33	8	0x00E1	8
MC_ME	PCTL35	8	0x00E3	8
MC_ME	PCTL38	8	0x00E6	8
MC_ME	PCTL39	8	0x00E7	8
MC_ME	PCTL40	8	0x00E8	8
MC_ME	PCTL41	8	0x00E9	8
MC_ME	PCTL42	8	0x00EA	8
MC_ME	PCTL48	8	0x00F0	8
MC_ME	PCTL49	8	0x00F1	8
MC_ME	PCTL50	8	0x00F2	8
MC_ME	PCTL51	8	0x00F3	8
MC_ME	PCTL58	8	0x00FA	8
MC_ME	PCTL92	8	0x011C	8
XOSC				
XOSC	OSC_CTL	32	0x0000	32
IRC_OSC				
IRC_OSC	RC_CTL	32	0x0000	32
FMPLL*n*				
FMPLL*n*	CR	32	0x0000	32
FMPLL*n*	MR	32	0x0004	32
I2C*n*				
I2C*n*	IBFD	8	0x0001	8
I2C*n*	IBCR	8	0x0002	8
I2C*n*	IBIC	8	0x0005	8
LINFlexD*n*				

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
LINFlexD*n*	LINCR1	32	0x0000	32
LINFlexD*n*	LINIER	32	0x0004	32
LINFlexD*n*	UARTCR	32	0x0010	32
LINFlexD*n*	LINTCSR	32	0x0018	32
LINFlexD*n*	LINOCR	32	0x001C	32
LINFlexD*n*	LINTOCR	32	0x0020	32
LINFlexD*n*	LINFBRR	32	0x0024	32
LINFlexD*n*	LINIBRR	32	0x0028	32
LINFlexD*n*	LINCR2	32	0x0030	32
LINFlexD*n*	BIDR	32	0x0034	32
LINFlexD*n*	IFER	32	0x0040	32
LINFlexD*n*	IFMR	32	0x0048	32
LINFlexD*n*	IFCR20	32	0x004C	32
LINFlexD*n*	IFCR21	32	0x0050	32
LINFlexD*n*	IFCR22	32	0x0054	32
LINFlexD*n*	IFCR23	32	0x0058	32
LINFlexD*n*	IFCR24	32	0x005C	32
LINFlexD*n*	IFCR25	32	0x0060	32
LINFlexD*n*	IFCR26	32	0x0064	32
LINFlexD*n*	IFCR27	32	0x0068	32
LINFlexD*n*	IFCR28	32	0x006C	32
LINFlexD*n*	IFCR29	32	0x0070	32
LINFlexD*n*	IFCR210	32	0x0074	32
LINFlexD*n*	IFCR211	32	0x0078	32
LINFlexD*n*	IFCR212	32	0x007C	32
LINFlexD*n*	IFCR213	32	0x0080	32
LINFlexD*n*	IFCR214	32	0x0084	32
LINFlexD*n*	IFCR215	32	0x0088	32
LINFlexD*n*	GCR	32	0x008C	32
LINFlexD*n*	UARTPTO	32	0x0090	32
LINFlexD*n*	DMATXE	32	0x0098	32
LINFlexD*n*	DMARXE	32	0x009C	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
CMU*n*				
CMU*n*	CSR	32	0x0000	32
CMU*n*	HFREFR_A	32	0x0008	32
CMU*n*	LFREFR_A	32	0x000C	32
CMU*n*	MDR	32	0x0018	32
PIT				
PIT	PIT_TFLG3	32	0x013C	32
MC_CGM				
MC CGM	OC_EN	32	0x0000	32
MC CGM	OCDS_SC	32	0x0004	32
MC CGM	SC_DC0	8	0x000C	8
MC CGM	AC0_SC	32	0x0010	32
MC CGM	AC0_DC0	8	0x0014	8
MC CGM	AC1_SC	32	0x0018	32
MC CGM	AC1_DC0	8	0x001C	8
MC CGM	AC2_SC	32	0x0020	32
MC CGM	AC2_DC0	8	0x0024	8
MC CGM	AC3_SC	32	0x0028	32
MC CGM	AC3_DC0	8	0x002C	8
MC CGM	AC4_SC	32	0x0030	32
MC_RGM				
MC RGM	FERD	16	0x0004	16
MC RGM	DERD	16	0x0006	16
MC RGM	FEAR	16	0x0010	16
MC RGM	FESS	16	0x0018	16
MC RGM	FBRE	16	0x001C	16
PMC				
PMC	PMC_CFGR	32	0x0000	32
PMC	PMC_TRIM1	32	0x0010	32
PMC	PMC_TRIM2	32	0x0014	32
PMC	PMC_TRIM3	32	0x0018	32
PIT				

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
PIT	PITMCR	32	0x0000	32
PIT	LDVAL0	32	0x0100	32
PIT	TCTRL0	32	0x0108	32
PIT	LDVAL1	32	0x0110	32
PIT	TCTRL1	32	0x0118	32
PIT	LDVAL2	32	0x0120	32
PIT	TCTRL2	32	0x0128	32
PIT	LDVAL3	32	0x0130	32
PIT	TCTRL3	32	0x0138	32
ADC*n*				
ADC*n*	MCR	32	0x0000	32
ADC*n*	IMR	32	0x0020	32
ADC*n*	CIMR0	32	0x0024	32
ADC*n*	WTIMR	32	0x0034	32
ADC*n*	DMAE	32	0x0040	32
ADC*n*	DMAR0	32	0x0044	32
ADC*n*	THRHLR0	32	0x0060	32
ADC*n*	THRHLR1	32	0x0064	32
ADC*n*	THRHLR2	32	0x0068	32
ADC*n*	THRHLR3	32	0x006C	32
ADC*n*	PSCR	32	0x0080	32
ADC*n*	PSR0	32	0x0084	32
ADC*n*	CTR0	32	0x0094	32
ADC*n*	NCMR0	32	0x00A4	32
ADC*n*	JCMR0	32	0x00B4	32
ADC*n*	PDEDR	32	0x00C8	32
ADC*n*	THRHLR4	32	0x0280	32
ADC*n*	THRHLR5	32	0x0284	32
ADC*n*	THRHLR6	32	0x0288	32
ADC*n*	THRHLR7	32	0x028C	32
ADC*n*	THRHLR8	32	0x0290	32
ADC*n*	THRHLR9	32	0x0294	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
ADC*n*	THRHLR10	32	0x0298	32
ADC*n*	THRHLR11	32	0x029C	32
ADC*n*	THRHLR12	32	0x02A0	32
ADC*n*	THRHLR13	32	0x02A4	32
ADC*n*	THRHLR14	32	0x02A8	32
ADC*n*	THRHLR15	32	0x02AC	32
ADC*n*	CWSELR0	32	0x02B0	32
ADC*n*	CWSELR1	32	0x02B4	32
ADC*n*	CWENR0	32	0x02E0	32
ADC*n*	AWORR0	32	0x02F0	32
ADC*n*	STCR1	32	0x0340	32
ADC*n*	STCR2	32	0x0344	32
ADC*n*	STCR3	32	0x0348	32
ADC*n*	STBRR	32	0x034C	32
ADC*n*	STAW0R	32	0x0380	32
ADC*n*	STAW1AR	32	0x0384	32
ADC*n*	STAW1BR	32	0x0388	32
ADC*n*	STAW2R	32	0x038C	32
ADC*n*	STAW3R	32	0x0390	32
ADC*n*	STAW4R	32	0x0394	32
eTimer 0				
eTimer 0	COMP10	16	0x0000	16
eTimer 0	COMP20	16	0x0002	16
eTimer 0	LOAD0	16	0x0008	16
eTimer 0	CTRL10	16	0x000E	16
eTimer 0	CTRL20	16	0x0010	16
eTimer 0	CTRL30	16	0x0012	16
eTimer 0	INTDMA0	16	0x0016	16
eTimer 0	CMPLD10	16	0x0018	16
eTimer 0	CMPLD20	16	0x001A	16
eTimer 0	CCCTRL0	16	0x001C	16
eTimer 0	FILT0	16	0x001E	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 0	COMP11	16	0x0020	16
eTimer 0	COMP21	16	0x0022	16
eTimer 0	LOAD1	16	0x0028	16
eTimer 0	CTRL11	16	0x002E	16
eTimer 0	CTRL21	16	0x0030	16
eTimer 0	CTRL31	16	0x0032	16
eTimer 0	INTDMA1	16	0x0036	16
eTimer 0	CMPLD11	16	0x0038	16
eTimer 0	CMPLD21	16	0x003A	16
eTimer 0	CCCTRL1	16	0x003C	16
eTimer 0	FILT1	16	0x003E	16
eTimer 0	COMP12	16	0x0040	16
eTimer 0	COMP22	16	0x0042	16
eTimer 0	LOAD2	16	0x0048	16
eTimer 0	CTRL12	16	0x004E	16
eTimer 0	CTRL22	16	0x0050	16
eTimer 0	CTRL32	16	0x0052	16
eTimer 0	INTDMA2	16	0x0056	16
eTimer 0	CMPLD12	16	0x0058	16
eTimer 0	CMPLD22	16	0x005A	16
eTimer 0	CCCTRL2	16	0x005C	16
eTimer 0	FILT2	16	0x005E	16
eTimer 0	COMP13	16	0x0060	16
eTimer 0	COMP23	16	0x0062	16
eTimer 0	LOAD3	16	0x0068	16
eTimer 0	CTRL13	16	0x006E	16
eTimer 0	CTRL23	16	0x0070	16
eTimer 0	CTRL33	16	0x0072	16
eTimer 0	INTDMA3	16	0x0076	16
eTimer 0	CMPLD13	16	0x0078	16
eTimer 0	CMPLD23	16	0x007A	16
eTimer 0	CCCTRL3	16	0x007C	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 0	FILT3	16	0x007E	16
eTimer 0	COMP14	16	0x0080	16
eTimer 0	COMP24	16	0x0082	16
eTimer 0	LOAD4	16	0x0088	16
eTimer 0	CTRL14	16	0x008E	16
eTimer 0	CTRL24	16	0x0090	16
eTimer 0	CTRL34	16	0x0092	16
eTimer 0	INTDMA4	16	0x0096	16
eTimer 0	CMPLD14	16	0x0098	16
eTimer 0	CMPLD24	16	0x009A	16
eTimer 0	CCCTRL4	16	0x009C	16
eTimer 0	FILT4	16	0x009E	16
eTimer 0	COMP15	16	0x00A0	16
eTimer 0	COMP25	16	0x00A2	16
eTimer 0	LOAD5	16	0x00A8	16
eTimer 0	CTRL15	16	0x00AE	16
eTimer 0	CTRL25	16	0x00B0	16
eTimer 0	CTRL35	16	0x00B2	16
eTimer 0	INTDMA5	16	0x00B6	16
eTimer 0	CMPLD15	16	0x00B8	16
eTimer 0	CMPLD25	16	0x00BA	16
eTimer 0	CCCTRL5	16	0x00BC	16
eTimer 0	FILT5	16	0x00BE	16
eTimer 0	WDTOL	16	0x0100	16
eTimer 0	WDTOH	16	0x0102	16
eTimer 0	ENBL	16	0x010C	16
eTimer 0	DREQ0	16	0x0110	16
eTimer 0	DREQ1	16	0x0112	16
eTimer 1				
eTimer1	COMP10	16	0x0000	16
eTimer 1	COMP20	16	0x0002	16
eTimer 1	LOAD0	16	0x0008	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 1	CTRL10	16	0x000E	16
eTimer 1	CTRL20	16	0x0010	16
eTimer 1	CTRL30	16	0x0012	16
eTimer 1	INTDMA0	16	0x0016	16
eTimer 1	CMPLD10	16	0x0018	16
eTimer 1	CMPLD20	16	0x001A	16
eTimer 1	CCCTRL0	16	0x001C	16
eTimer 1	FILT0	16	0x001E	16
eTimer 1	COMP11	16	0x0020	16
eTimer 1	COMP21	16	0x0022	16
eTimer 1	LOAD1	16	0x0028	16
eTimer 1	CTRL11	16	0x002E	16
eTimer 1	CTRL21	16	0x0030	16
eTimer 1	CTRL31	16	0x0032	16
eTimer 1	INTDMA1	16	0x0036	16
eTimer 1	CMPLD11	16	0x0038	16
eTimer 1	CMPLD21	16	0x003A	16
eTimer 1	CCCTRL1	16	0x003C	16
eTimer 1	FILT1	16	0x003E	16
eTimer 1	COMP12	16	0x0040	16
eTimer 1	COMP22	16	0x0042	16
eTimer 1	LOAD2	16	0x0048	16
eTimer 1	CTRL12	16	0x004E	16
eTimer 1	CTRL22	16	0x0050	16
eTimer 1	CTRL32	16	0x0052	16
eTimer 1	INTDMA2	16	0x0056	16
eTimer 1	CMPLD12	16	0x0058	16
eTimer 1	CMPLD22	16	0x005A	16
eTimer 1	CCCTRL2	16	0x005C	16
eTimer 1	FILT2	16	0x005E	16
eTimer 1	COMP13	16	0x0060	16
eTimer 1	COMP23	16	0x0062	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 1	LOAD3	16	0x0068	16
eTimer 1	CTRL13	16	0x006E	16
eTimer 1	CTRL23	16	0x0070	16
eTimer 1	CTRL33	16	0x0072	16
eTimer 1	INTDMA3	16	0x0076	16
eTimer 1	CMPLD13	16	0x0078	16
eTimer 1	CMPLD23	16	0x007A	16
eTimer 1	CCCTRL3	16	0x007C	16
eTimer 1	FILT3	16	0x007E	16
eTimer 1	COMP14	16	0x0080	16
eTimer 1	COMP24	16	0x0082	16
eTimer 1	LOAD4	16	0x0088	16
eTimer 1	CTRL14	16	0x008E	16
eTimer 1	CTRL24	16	0x0090	16
eTimer 1	CTRL34	16	0x0092	16
eTimer 1	INTDMA4	16	0x0096	16
eTimer 1	CMPLD14	16	0x0098	16
eTimer 1	CMPLD24	16	0x009A	16
eTimer 1	CCCTRL4	16	0x009C	16
eTimer 1	FILT4	16	0x009E	16
eTimer 1	COMP15	16	0x00A0	16
eTimer 1	COMP25	16	0x00A2	16
eTimer 1	LOAD5	16	0x00A8	16
eTimer 1	CTRL15	16	0x00AE	16
eTimer 1	CTRL25	16	0x00B0	16
eTimer 1	CTRL35	16	0x00B2	16
eTimer 1	INTDMA5	16	0x00B6	16
eTimer 1	CMPLD15	16	0x00B8	16
eTimer 1	CMPLD25	16	0x00BA	16
eTimer 1	CCCTRL5	16	0x00BC	16
eTimer 1	FILT5	16	0x00BE	16
eTimer 1	ENBL	16	0x010C	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 1	DREQ0	16	0x0110	16
eTimer 1	DREQ1	16	0x0112	16
eTimer 2				
eTimer 2	COMP10	16	0x0000	16
eTimer 2	COMP20	16	0x0002	16
eTimer 2	LOAD0	16	0x0008	16
eTimer 2	CTRL10	16	0x000E	16
eTimer 2	CTRL20	16	0x0010	16
eTimer 2	CTRL30	16	0x0012	16
eTimer 2	INTDMA0	16	0x0016	16
eTimer 2	CMPLD10	16	0x0018	16
eTimer 2	CMPLD20	16	0x001A	16
eTimer 2	CCCTRL0	16	0x001C	16
eTimer 2	FILT0	16	0x001E	16
eTimer 2	COMP11	16	0x0020	16
eTimer 2	COMP21	16	0x0022	16
eTimer 2	LOAD1	16	0x0028	16
eTimer 2	CTRL11	16	0x002E	16
eTimer 2	CTRL21	16	0x0030	16
eTimer 2	CTRL31	16	0x0032	16
eTimer 2	INTDMA1	16	0x0036	16
eTimer 2	CMPLD11	16	0x0038	16
eTimer 2	CMPLD21	16	0x003A	16
eTimer 2	CCCTRL1	16	0x003C	16
eTimer 2	FILT1	16	0x003E	16
eTimer 2	COMP12	16	0x0040	16
eTimer 2	COMP22	16	0x0042	16
eTimer 2	LOAD2	16	0x0048	16
eTimer 2	CTRL12	16	0x004E	16
eTimer 2	CTRL22	16	0x0050	16
eTimer 2	CTRL32	16	0x0052	16
eTimer 2	INTDMA2	16	0x0056	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 2	CMPLD12	16	0x0058	16
eTimer 2	CMPLD22	16	0x005A	16
eTimer 2	CCCTRL2	16	0x005C	16
eTimer 2	FILT2	16	0x005E	16
eTimer 2	COMP13	16	0x0060	16
eTimer 2	COMP23	16	0x0062	16
eTimer 2	LOAD3	16	0x0068	16
eTimer 2	CTRL13	16	0x006E	16
eTimer 2	CTRL23	16	0x0070	16
eTimer 2	CTRL33	16	0x0072	16
eTimer 2	INTDMA3	16	0x0076	16
eTimer 2	CMPLD13	16	0x0078	16
eTimer 2	CMPLD23	16	0x007A	16
eTimer 2	CCCTRL3	16	0x007C	16
eTimer 2	FILT3	16	0x007E	16
eTimer 2	COMP14	16	0x0080	16
eTimer 2	COMP24	16	0x0082	16
eTimer 2	LOAD4	16	0x0088	16
eTimer 2	CTRL14	16	0x008E	16
eTimer 2	CTRL24	16	0x0090	16
eTimer 2	CTRL34	16	0x0092	16
eTimer 2	INTDMA4	16	0x0096	16
eTimer 2	CMPLD14	16	0x0098	16
eTimer 2	CMPLD24	16	0x009A	16
eTimer 2	CCCTRL4	16	0x009C	16
eTimer 2	FILT4	16	0x009E	16
eTimer 2	COMP15	16	0x00A0	16
eTimer 2	COMP25	16	0x00A2	16
eTimer 2	LOAD5	16	0x00A8	16
eTimer 2	CTRL15	16	0x00AE	16
eTimer 2	CTRL25	16	0x00B0	16
eTimer 2	CTRL35	16	0x00B2	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
eTimer 2	INTDMA5	16	0x00B6	16
eTimer 2	CMPLD15	16	0x00B8	16
eTimer 2	CMPLD25	16	0x00BA	16
eTimer 2	CCCTRL5	16	0x00BC	16
eTimer 2	FILT5	16	0x00BE	16
eTimer 2	ENBL	16	0x010C	16
eTimer 2	DREQ0	16	0x0110	16
eTimer 2	DREQ1	16	0x0112	16
DMACHMUX				
DMACHMUX	CHCONFIG0	8	0x0000	8
DMACHMUX	CHCONFIG1	8	0x0001	8
DMACHMUX	CHCONFIG2	8	0x0002	8
DMACHMUX	CHCONFIG3	8	0x0003	8
DMACHMUX	CHCONFIG4	8	0x0004	8
DMACHMUX	CHCONFIG5	8	0x0005	8
DMACHMUX	CHCONFIG6	8	0x0006	8
DMACHMUX	CHCONFIG7	8	0x0007	8
DMACHMUX	CHCONFIG8	8	0x0008	8
DMACHMUX	CHCONFIG9	8	0x0009	8
DMACHMUX	CHCONFIG10	8	0x000A	8
DMACHMUX	CHCONFIG11	8	0x000B	8
DMACHMUX	CHCONFIG12	8	0x000C	8
DMACHMUX	CHCONFIG13	8	0x000D	8
DMACHMUX	CHCONFIG14	8	0x000E	8
DMACHMUX	CHCONFIG15	8	0x000F	8
DRAMC				
DRAMC	sys_config	32	0x0000	32
DRAMC	time_config0	32	0x0004	32
DRAMC	time_config1	32	0x0008	32
DRAMC	time_config2	32	0x000C	32
DRAMC	command	32	0x0010	32
DRAMC	self_refresh_cmd_0	32	0x0018	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
DRAMC	self_refresh_cmd_1	32	0x001C	32
DRAMC	self_refresh_cmd_2	32	0x0020	32
DRAMC	self_refresh_cmd_3	32	0x0024	32
DRAMC	self_refresh_cmd_4	32	0x0028	32
DRAMC	self_refresh_cmd_5	32	0x002C	32
DRAMC	self_refresh_cmd_6	32	0x0030	32
DRAMC	self_refresh_cmd_7	32	0x0034	32
DRAMC	DQS_Config_Offset_Count	32	0x0038	32
DRAMC	DQS_Config_Offset_time	32	0x003C	32
DRAMC	Aux_config	32	0x0044	32
DRAMC	SELF_TEST_DATA_OUT_HIGH	32	0x0048	32
DRAMC	SELF_TEST_DATA_OUT_LO	32	0x004C	32
DRAMC	SELF_TEST_DATA_EXPECT_HIGH	32	0x0050	32
DRAMC	SELF_TEST_DATA_EXPECT_LOW	32	0x0054	32
DRAMC	compact_command	32	0x0014	32
EBI				
EBI	EBI_BMCR	32	0x000C	32
EBI	EBI_BR0	32	0x0010	32
EBI	EBI_OR0	32	0x0014	32
EBI	EBI_BR1	32	0x0018	32
EBI	EBI_OR1	32	0x001C	32
EBI	EBI_BR2	32	0x0020	32
EBI	EBI_OR2	32	0x0024	32
EBI	EBI_BR3	32	0x0028	32
EBI	EBI_OR03	32	0x002C	32
EBI	EBI_CAL_BR0	32	0x0040	32
EBI	EBI_CAL_OR0	32	0x0044	32
EBI	EBI_CAL_BR	32	0x0048	32
EBI	EBI_CAL_OR1	32	0x004C	32
EBI	EBI_CAL_BR2	32	0x0050	32
EBI	EBI_CAL_OR2	32	0x0054	32
EBI	EBI_CAL_BR3	32	0x0058	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
EBI	EBI_CAL_OR3	32	0x005C	32
FCCU				
FCCU	CFG	32	0x0008	32
FCCU	CF_CFG0	32	0x000C	32
FCCU	CF_CFG1	32	0x0010	32
FCCU	CF_CFG2	32	0x0014	32
FCCU	CF_CFG3	32	0x0018	32
FCCU	NCF_CFG0	32	0x001C	32
FCCU	NCF_CFG1	32	0x0020	32
FCCU	NCF_CFG2	32	0x0024	32
FCCU	NCF_CFG3	32	0x0028	32
FCCU	CFS_CFG0	32	0x002C	32
FCCU	CFS_CFG1	32	0x0030	32
FCCU	CFS_CFG2	32	0x0034	32
FCCU	CFS_CFG3	32	0x0038	32
FCCU	CFS_CFG4	32	0x003C	32
FCCU	CFS_CFG5	32	0x0040	32
FCCU	CFS_CFG6	32	0x0044	32
FCCU	CFS_CFG7	32	0x0048	32
FCCU	NCFS_CFG0	32	0x004C	32
FCCU	NCFS_CFG1	32	0x0050	32
FCCU	NCFS_CFG2	32	0x0054	32
FCCU	NCFS_CFG3	32	0x0058	32
FCCU	NCFS_CFG4	32	0x005C	32
FCCU	NCFS_CFG5	32	0x0060	32
FCCU	NCFS_CFG6	32	0x0064	32
FCCU	NCFS_CFG7	32	0x0068	32
FCCU	NCFE0	32	0x0094	32
FCCU	NCFE1	32	0x0098	32
FCCU	NCFE2	32	0x009C	32
FCCU	NCFE3	32	0x00A0	32
FCCU	NCF_TOE0	32	0x00A4	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FCCU	NCF_TOE1	32	0x00A8	32
FCCU	NCF_TOE2	32	0x00AC	32
FCCU	NCF_TOE3	32	0x00B0	32
FCCU	NCF_TO	32	0x00B4	32
FCCU	CFG_TO	32	0x00B8	32
FCCU	EINOUT	32	0x00BC	32
FCCU	CFF	32	0x00D8	32
FCCU	NCFF	32	0x00DC	32
FCCU	IRQ_EN	32	0x00E4	32
FEC				
FEC	EIMR	32	0x0008	32
FEC	RDAR	32	0x0010	32
FEC	TDAR	32	0x0014	32
FEC	ECR	32	0x0024	32
FEC	MMFR	32	0x0040	32
FEC	MSCR	32	0x0044	32
FEC	MIBC	32	0x0064	32
FEC	RCR	32	0x0084	32
FEC	TCR	32	0x00C4	32
FEC	PALR	32	0x00E4	32
FEC	PAUR	32	0x00E8	32
FEC	OPD	32	0x00EC	32
FEC	IAUR	32	0x0118	32
FEC	IALR	32	0x011C	32
FEC	GAUR	32	0x0120	32
FEC	GALR	32	0x0124	32
FEC	RANDOM	32	0x0128	32
FEC	RAND1	32	0x012C	32
FEC	FIFO_ID	32	0x0140	32
FEC	TFWR	32	0x0144	32
FEC	FRBR	32	0x014C	32
FEC	FRSR	32	0x0150	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FEC	ERDSR	32	0x0180	32
FEC	ETDSR	32	0x0184	32
FEC	EMRBR	32	0x0188	32
FlexPWM				
FlexPWM	INIT0	16	0x0002	16
FlexPWM	CTRL20	16	0x0004	16
FlexPWM	CTRL10	16	0x0006	16
FlexPWM	VAL_00	16	0x0008	16
FlexPWM	VAL_10	16	0x000A	16
FlexPWM	VAL_20	16	0x000C	16
FlexPWM	VAL_30	16	0x000E	16
FlexPWM	VAL_40	16	0x0010	16
FlexPWM	VAL_50	16	0x0012	16
FlexPWM	OCTRL0	16	0x0018	16
FlexPWM	INTEN0	16	0x001C	16
FlexPWM	DMAEN0	16	0x001E	16
FlexPWM	TCTRL0	16	0x0020	16
FlexPWM	DISMAP0	16	0x0022	16
FlexPWM	DTCNT00	16	0x0024	16
FlexPWM	DTCNT10	16	0x0026	16
FlexPWM	CAPTCTRLX0	16	0x0030	16
FlexPWM	CAPTCMPX0	16	0x0032	16
FlexPWM	INIT1	16	0x0050	16
FlexPWM	CTRL21	16	0x0054	16
FlexPWM	CTRL11	16	0x0056	16
FlexPWM	VAL_01	16	0x0058	16
FlexPWM	VAL_11	16	0x005A	16
FlexPWM	VAL_21	16	0x005C	16
FlexPWM	VAL_31	16	0x005E	16
FlexPWM	VAL_41	16	0x0060	16
FlexPWM	VAL_51	16	0x0062	16
FlexPWM	OCTRL1	16	0x0068	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexPWM	INTEN1	16	0x006C	16
FlexPWM	DMAEN1	16	0x006E	16
FlexPWM	TCTRL1	16	0x0070	16
FlexPWM	DISMAP1	16	0x0072	16
FlexPWM	DTCNT01	16	0x0074	16
FlexPWM	DTCNT11	16	0x0076	16
FlexPWM	CAPTCTRLX1	16	0x0080	16
FlexPWM	CAPTCMPX1	16	0x0082	16
FlexPWM	INIT2	16	0x00A2	16
FlexPWM	CTRL22	16	0x00A4	16
FlexPWM	CTRL12	16	0x00A6	16
FlexPWM	VAL_02	16	0x00A8	16
FlexPWM	VAL_12	16	0x00AA	16
FlexPWM	VAL_22	16	0x00AC	16
FlexPWM	VAL_32	16	0x00AE	16
FlexPWM	VAL_42	16	0x00B0	16
FlexPWM	VAL_52	16	0x00B2	16
FlexPWM	OCTRL2	16	0x00B8	16
FlexPWM	INTEN2	16	0x00BC	16
FlexPWM	DMAEN2	16	0x00BE	16
FlexPWM	TCTRL2	16	0x00C0	16
FlexPWM	DISMAP2	16	0x00C2	16
FlexPWM	DTCNT02	16	0x00C4	16
FlexPWM	DTCNT12	16	0x00C6	16
FlexPWM	CAPTCTRLX2	16	0x00D0	16
FlexPWM	CAPTCMPX2	16	0x00D2	16
FlexPWM	INIT3	16	0x00F2	16
FlexPWM	CTRL23	16	0x00F4	16
FlexPWM	CTRL13	16	0x00F6	16
FlexPWM	VAL_03	16	0x00F8	16
FlexPWM	VAL_13	16	0x00FA	16
FlexPWM	VAL_23	16	0x00FC	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexPWM	VAL_33	16	0x00FE	16
FlexPWM	VAL_43	16	0x0100	16
FlexPWM	VAL_53	16	0x0102	16
FlexPWM	OCTRL3	16	0x0108	16
FlexPWM	INTEN3	16	0x010C	16
FlexPWM	DMAEN3	16	0x010E	16
FlexPWM	TCTRL3	16	0x0110	16
FlexPWM	DISMAP3	16	0x0112	16
FlexPWM	DTCNT03	16	0x0114	16
FlexPWM	DTCNT13	16	0x0116	16
FlexPWM	CAPTCTRLX3	16	0x0120	16
FlexPWM	CAPTCMPX3	16	0x0122	16
FlexPWM	OUTEN	16	0x0140	16
FlexPWM	MASK	16	0x0142	16
FlexPWM	SWCOUT	16	0x0144	16
FlexPWM	DTSRCSEL	16	0x0146	16
FlexPWM	MCTRL	16	0x0148	16
FlexPWM	FCTRL	16	0x014C	16
DSPI*n*				
DSPI*n*	MCR	32	0x0000	32
DSPI*n*	TCR	32	0x0008	32
DSPI*n*	CTAR0	32	0x000C	32
DSPI*n*	DSPI_CTAR1	32	0x0010	32
DSPI*n*	CTAR2	32	0x0014	32
DSPI*n*	CTAR3	32	0x0018	32
DSPI*n*	RSER	32	0x0030	32
CRC				
CRC	CFG0	32	0x0000	32
CRC	CFG1	32	0x0010	32
CRC	CFG2	32	0x0020	32
CTU*n*				
CTU*n*	TGSISR	32	0x0000	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
CTU*n*	TGSCR	16	0x0004	16
CTU*n*	T0CR	16	0x0006	16
CTU*n*	T1CR	16	0x0008	16
CTU*n*	T2CR	16	0x000A	16
CTU*n*	T3CR	16	0x000C	16
CTU*n*	T4CR	16	0x000E	16
CTU*n*	T5CR	16	0x0010	16
CTU*n*	T6CR	16	0x0012	16
CTU*n*	T7CR	16	0x0014	16
CTU*n*	TGSCCR	16	0x0016	16
CTU*n*	TGSCRR	16	0x0018	16
CTU*n*	CLCR1	32	0x001C	32
CTU*n*	CLCR2	32	0x0020	32
CTU*n*	THCR1	32	0x0024	32
CTU*n*	THCR2	32	0x0028	32
CTU*n*	CLR1	16	0x002C	16
CTU*n*	CLR2	16	0x002E	16
CTU*n*	CLR3	16	0x0030	16
CTU*n*	CLR4	16	0x0032	16
CTU*n*	CLR5	16	0x0034	16
CTU*n*	CLR6	16	0x0036	16
CTU*n*	CLR7	16	0x0038	16
CTU*n*	CLR8	16	0x003A	16
CTU*n*	CLR9	16	0x003C	16
CTU*n*	CLR10	16	0x003E	16
CTU*n*	CLR11	16	0x0040	16
CTU*n*	CLR12	16	0x0042	16
CTU*n*	CLR13	16	0x0044	16
CTU*n*	CLR14	16	0x0046	16
CTU*n*	CLR15	16	0x0048	16
CTU*n*	CLR16	16	0x004A	16
CTU*n*	CLR17	16	0x004C	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
CTU*n*	CLR18	16	0x004E	16
CTU*n*	CLR19	16	0x0050	16
CTU*n*	CLR20	16	0x0052	16
CTU*n*	CLR21	16	0x0054	16
CTU*n*	CLR22	16	0x0056	16
CTU*n*	CLR23	16	0x0058	16
CTU*n*	CLR24	16	0x005A	16
CTU*n*	CR	16	0x006C	16
CTU*n*	FCR	32	0x0070	32
CTU*n*	TH1	32	0x0074	32
CTU*n*	CTUIR	16	0x00C4	16
CTU*n*	COTR	16	0x00C6	16
CTU*n*	CTUCR	16	0x00C8	16
CTU*n*	FILTER	16	0x00CA	16
CTU*n*	EXPECTED_A	16	0x00CC	16
CTU*n*	EXPECTED_B	16	0x00CE	16
CTU*n*	CNT_RANGE	16	0x00D0	16
FlexCAN*n*				
FlexCAN*n*	MCR	32	0x0000	32
FlexCAN*n*	CTRL	32	0x0004	32
FlexCAN*n*	RXGMASK	32	0x0010	32
FlexCAN*n*	RX14MASK	32	0x0014	32
FlexCAN*n*	RX15MASK	32	0x0018	32
FlexCAN*n*	IMASK1	32	0x0028	32
FlexCAN*n*	MSG0_CS	32	0x0080	32
FlexCAN*n*	MSG0_ID	32	0x0084	32
FlexCAN*n*	MSG1_CS	32	0x0090	32
FlexCAN*n*	MSG1_ID	32	0x0094	32
FlexCAN*n*	MSG2_CS	32	0x00A0	32
FlexCAN*n*	MSG2_ID	32	0x00A4	32
FlexCAN*n*	MSG3_CS	32	0x00B0	32
FlexCAN*n*	MSG3_ID	32	0x00B4	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexCAN*n*	MSG4_CS	32	0x00C0	32
FlexCAN*n*	MSG4_ID	32	0x00C4	32
FlexCAN*n*	MSG5_CS	32	0x00D0	32
FlexCAN*n*	MSG5_ID	32	0x00D4	32
FlexCAN*n*	MSG6_CS	32	0x00E0	32
FlexCAN*n*	MSG6_ID	32	0x00E4	32
FlexCAN*n*	MSG7_CS	32	0x00F0	32
FlexCAN*n*	MSG7_ID	32	0x00F4	32
FlexCAN*n*	MSG8_CS	32	0x0100	32
FlexCAN*n*	MSG8_ID	32	0x0104	32
FlexCAN*n*	MSG9_CS	32	0x0110	32
FlexCAN*n*	MSG9_ID	32	0x0114	32
FlexCAN*n*	MSG10_CS	32	0x0120	32
FlexCAN*n*	MSG10_ID	32	0x0124	32
FlexCAN*n*	MSG11_CS	32	0x0130	32
FlexCAN*n*	MSG11_ID	32	0x0134	32
FlexCAN*n*	MSG12_CS	32	0x0140	32
FlexCAN*n*	MSG12_ID	32	0x0144	32
FlexCAN*n*	MSG13_CS	32	0x0150	32
FlexCAN*n*	MSG13_ID	32	0x0154	32
FlexCAN*n*	MSG14_CS	32	0x0160	32
FlexCAN*n*	MSG14_ID	32	0x0164	32
FlexCAN*n*	MSG15_CS	32	0x0170	32
FlexCAN*n*	MSG15_ID	32	0x0174	32
FlexCAN*n*	MSG16_CS	32	0x0180	32
FlexCAN*n*	MSG16_ID	32	0x0184	32
FlexCAN*n*	MSG17_CS	32	0x0190	32
FlexCAN*n*	MSG17_ID	32	0x0194	32
FlexCAN*n*	MSG18_CS	32	0x01A0	32
FlexCAN*n*	MSG18_ID	32	0x01A4	32
FlexCAN*n*	MSG19_CS	32	0x01B0	32
FlexCAN*n*	MSG19_ID	32	0x01B4	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexCAN*n*	MSG20_CS	32	0x01C0	32
FlexCAN*n*	MSG20_ID	32	0x01C4	32
FlexCAN*n*	MSG21_CS	32	0x01D0	32
FlexCAN*n*	MSG21_ID	32	0x01D4	32
FlexCAN*n*	MSG22_CS	32	0x01E0	32
FlexCAN*n*	MSG22_ID	32	0x01E4	32
FlexCAN*n*	MSG23_CS	32	0x01F0	32
FlexCAN*n*	MSG23_ID	32	0x01F4	32
FlexCAN*n*	MSG24_CS	32	0x0200	32
FlexCAN*n*	MSG24_ID	32	0x0204	32
FlexCAN*n*	MSG25_CS	32	0x0210	32
FlexCAN*n*	MSG25_ID	32	0x0214	32
FlexCAN*n*	MSG26_CS	32	0x0220	32
FlexCAN*n*	MSG26_ID	32	0x0224	32
FlexCAN*n*	MSG27_CS	32	0x0230	32
FlexCAN*n*	MSG27_ID	32	0x0234	32
FlexCAN*n*	MSG28_CS	32	0x0240	32
FlexCAN*n*	MSG28_ID	32	0x0244	32
FlexCAN*n*	MSG29_CS	32	0x0250	32
FlexCAN*n*	MSG29_ID	32	0x0254	32
FlexCAN*n*	MSG30_CS	32	0x0260	32
FlexCAN*n*	MSG30_ID	32	0x0264	32
FlexCAN*n*	MSG31_CS	32	0x0270	32
FlexCAN*n*	MSG31_ID	32	0x0274	32
FlexCAN*n*	RXIMR0	32	0x0880	32
FlexCAN*n*	RXIMR1	32	0x0884	32
FlexCAN*n*	RXIMR2	32	0x0888	32
FlexCAN*n*	RXIMR3	32	0x088C	32
FlexCAN*n*	RXIMR4	32	0x0890	32
FlexCAN*n*	RXIMR5	32	0x0894	32
FlexCAN*n*	RXIMR6	32	0x0898	32
FlexCAN*n*	RXIMR7	32	0x089C	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexCAN*n*	RXIMR8	32	0x08A0	32
FlexCAN*n*	RXIMR9	32	0x08A4	32
FlexCAN*n*	RXIMR10	32	0x08A8	32
FlexCAN*n*	RXIMR11	32	0x08AC	32
FlexCAN*n*	RXIMR12	32	0x08B0	32
FlexCAN*n*	RXIMR13	32	0x08B4	32
FlexCAN*n*	RXIMR14	32	0x08B8	32
FlexCAN*n*	RXIMR15	32	0x08BC	32
FlexCAN*n*	RXIMR16	32	0x08C0	32
FlexCAN*n*	RXIMR17	32	0x08C4	32
FlexCAN*n*	RXIMR18	32	0x08C8	32
FlexCAN*n*	RXIMR19	32	0x08CC	32
FlexCAN*n*	RXIMR20	32	0x08D0	32
FlexCAN*n*	RXIMR21	32	0x08D4	32
FlexCAN*n*	RXIMR22	32	0x08D8	32
FlexCAN*n*	RXIMR23	32	0x08DC	32
FlexCAN*n*	RXIMR24	32	0x08E0	32
FlexCAN*n*	RXIMR25	32	0x08E4	32
FlexCAN*n*	RXIMR26	32	0x08E8	32
FlexCAN*n*	RXIMR27	32	0x08EC	32
FlexCAN*n*	RXIMR28	32	0x08F0	32
FlexCAN*n*	RXIMR29	32	0x08F4	32
FlexCAN*n*	RXIMR30	32	0x08F8	32
FlexCAN*n*	RXIMR31	32	0x08FC	32
FlexRay				
FlexRay	MCR	16	0x0002	16
FlexRay	SYMBADHR	16	0x0004	16
FlexRay	SYMBADLR	16	0x0006	16
FlexRay	STBSCR	16	0x0008	16
FlexRay	MBDSR	16	0x000C	16
FlexRay	MBSSUTR	16	0x000E	16
FlexRay	POCR	16	0x0014	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	GIFER	16	0x0016	16
FlexRay	PIER0	16	0x001C	16
FlexRay	PIER1	16	0x001E	16
FlexRay	SYMATOR	16	0x003E	16
FlexRay	SFTOR	16	0x0042	16
FlexRay	SFTCCSR	16	0x0044	16
FlexRay	SFIDRFR	16	0x0046	16
FlexRay	SFIDAFVR	16	0x0048	16
FlexRay	SFIDAFMR	16	0x004A	16
FlexRay	NMVLR	16	0x0058	16
FlexRay	TICCR	16	0x005A	16
FlexRay	TI1CYSR	16	0x005C	16
FlexRay	TI1MTOR	16	0x005E	16
FlexRay	TI2CR0	16	0x0060	16
FlexRay	TI2CR1	16	0x0062	16
FlexRay	MTSACFR	16	0x0080	16
FlexRay	MTSBCFR	16	0x0082	16
FlexRay	RFRFCTR	16	0x009A	16
FlexRay	PCR0	16	0x00A0	16
FlexRay	PCR1	16	0x00A2	16
FlexRay	PCR2	16	0x00A4	16
FlexRay	PCR3	16	0x00A6	16
FlexRay	PCR4	16	0x00A8	16
FlexRay	PCR5	16	0x00AA	16
FlexRay	PCR6	16	0x00AC	16
FlexRay	PCR7	16	0x00AE	16
FlexRay	PCR8	16	0x00B0	16
FlexRay	PCR9	16	0x00B2	16
FlexRay	PCR10	16	0x00B4	16
FlexRay	PCR11	16	0x00B6	16
FlexRay	PCR12	16	0x00B8	16
FlexRay	PCR13	16	0x00BA	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	PCR14	16	0x00BC	16
FlexRay	PCR15	16	0x00BE	16
FlexRay	PCR16	16	0x00C0	16
FlexRay	PCR17	16	0x00C2	16
FlexRay	PCR18	16	0x00C4	16
FlexRay	PCR19	16	0x00C6	16
FlexRay	PCR20	16	0x00C8	16
FlexRay	PCR21	16	0x00CA	16
FlexRay	PCR22	16	0x00CC	16
FlexRay	PCR23	16	0x00CE	16
FlexRay	PCR24	16	0x00D0	16
FlexRay	PCR25	16	0x00D2	16
FlexRay	PCR26	16	0x00D4	16
FlexRay	PCR27	16	0x00D6	16
FlexRay	PCR28	16	0x00D8	16
FlexRay	PCR29	16	0x00DA	16
FlexRay	PCR30	16	0x00DC	16
FlexRay	MBCCFR0	16	0x0802	16
FlexRay	MBFIDR0	16	0x0804	16
FlexRay	MBCCFR1	16	0x080A	16
FlexRay	MBFIDR1	16	0x080C	16
FlexRay	MBCCFR2	16	0x0812	16
FlexRay	MBFIDR2	16	0x0814	16
FlexRay	MBCCFR3	16	0x081A	16
FlexRay	MBFIDR3	16	0x081C	16
FlexRay	MBCCFR4	16	0x0822	16
FlexRay	MBFIDR4	16	0x0824	16
FlexRay	MBCCFR5	16	0x082A	16
FlexRay	MBFIDR5	16	0x082C	16
FlexRay	MBCCFR6	16	0x0832	16
FlexRay	MBFIDR6	16	0x0834	16
FlexRay	MBCCFR7	16	0x083A	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	MBFIDR7	16	0x083C	16
FlexRay	MBCCFR8	16	0x0842	16
FlexRay	MBFIDR8	16	0x0844	16
FlexRay	MBCCFR9	16	0x084A	16
FlexRay	MBFIDR9	16	0x084C	16
FlexRay	MBCCFR10	16	0x0852	16
FlexRay	MBFIDR10	16	0x0854	16
FlexRay	MBCCFR11	16	0x085A	16
FlexRay	MBFIDR11	16	0x085C	16
FlexRay	MBCCFR12	16	0x0862	16
FlexRay	MBFIDR12	16	0x0864	16
FlexRay	MBCCFR13	16	0x086A	16
FlexRay	MBFIDR13	16	0x086C	16
FlexRay	MBCCFR14	16	0x0872	16
FlexRay	MBFIDR14	16	0x0874	16
FlexRay	MBCCFR15	16	0x087A	16
FlexRay	MBFIDR15	16	0x087C	16
FlexRay	MBCCFR16	16	0x0882	16
FlexRay	MBFIDR16	16	0x0884	16
FlexRay	MBCCFR17	16	0x088A	16
FlexRay	MBFIDR17	16	0x088C	16
FlexRay	MBCCFR18	16	0x0892	16
FlexRay	MBFIDR18	16	0x0894	16
FlexRay	MBCCFR19	16	0x089A	16
FlexRay	MBFIDR19	16	0x089C	16
FlexRay	MBCCFR20	16	0x08A2	16
FlexRay	MBFIDR20	16	0x08A4	16
FlexRay	MBCCFR21	16	0x08AA	16
FlexRay	MBFIDR21	16	0x08AC	16
FlexRay	MBCCFR22	16	0x08B2	16
FlexRay	MBFIDR22	16	0x08B4	16
FlexRay	MBCCFR23	16	0x08BA	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	MBFIDR23	16	0x08BC	16
FlexRay	MBCCFR24	16	0x08C2	16
FlexRay	MBFIDR24	16	0x08C4	16
FlexRay	MBCCFR25	16	0x08CA	16
FlexRay	MBFIDR25	16	0x08CC	16
FlexRay	MBCCFR26	16	0x08D2	16
FlexRay	MBFIDR26	16	0x08D4	16
FlexRay	MBCCFR27	16	0x08DA	16
FlexRay	MBFIDR27	16	0x08DC	16
FlexRay	MBCCFR28	16	0x08E2	16
FlexRay	MBFIDR28	16	0x08E4	16
FlexRay	MBCCFR29	16	0x08EA	16
FlexRay	MBFIDR29	16	0x08EC	16
FlexRay	MBCCFR30	16	0x08F2	16
FlexRay	MBFIDR30	16	0x08F4	16
FlexRay	MBCCFR31	16	0x08FA	16
FlexRay	MBFIDR31	16	0x08FC	16
FlexRay	MBCCFR32	16	0x0902	16
FlexRay	MBFIDR32	16	0x0904	16
FlexRay	MBCCFR33	16	0x090A	16
FlexRay	MBFIDR33	16	0x090C	16
FlexRay	MBCCFR34	16	0x0912	16
FlexRay	MBFIDR34	16	0x0914	16
FlexRay	MBCCFR35	16	0x091A	16
FlexRay	MBFIDR35	16	0x091C	16
FlexRay	MBCCFR36	16	0x0922	16
FlexRay	MBFIDR36	16	0x0924	16
FlexRay	MBCCFR37	16	0x092A	16
FlexRay	MBFIDR37	16	0x092C	16
FlexRay	MBCCFR38	16	0x0932	16
FlexRay	MBFIDR38	16	0x0934	16
FlexRay	MBCCFR39	16	0x093A	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	MBFIDR39	16	0x093C	16
FlexRay	MBCCFR40	16	0x0942	16
FlexRay	MBFIDR40	16	0x0944	16
FlexRay	MBCCFR41	16	0x094A	16
FlexRay	MBFIDR41	16	0x094C	16
FlexRay	MBCCFR42	16	0x0952	16
FlexRay	MBFIDR42	16	0x0954	16
FlexRay	MBCCFR43	16	0x095A	16
FlexRay	MBFIDR43	16	0x095C	16
FlexRay	MBCCFR44	16	0x0962	16
FlexRay	MBFIDR44	16	0x0964	16
FlexRay	MBCCFR45	16	0x096A	16
FlexRay	MBFIDR45	16	0x096C	16
FlexRay	MBCCFR46	16	0x0972	16
FlexRay	MBFIDR46	16	0x0974	16
FlexRay	MBCCFR47	16	0x097A	16
FlexRay	MBFIDR47	16	0x097C	16
FlexRay	MBCCFR48	16	0x0982	16
FlexRay	MBFIDR48	16	0x0984	16
FlexRay	MBCCFR49	16	0x098A	16
FlexRay	MBFIDR49	16	0x098C	16
FlexRay	MBCCFR50	16	0x0992	16
FlexRay	MBFIDR50	16	0x0994	16
FlexRay	MBCCFR51	16	0x099A	16
FlexRay	MBFIDR51	16	0x099C	16
FlexRay	MBCCFR52	16	0x09A2	16
FlexRay	MBFIDR52	16	0x09A4	16
FlexRay	MBCCFR53	16	0x09AA	16
FlexRay	MBFIDR53	16	0x09AC	16
FlexRay	MBCCFR54	16	0x09B2	16
FlexRay	MBFIDR54	16	0x09B4	16
FlexRay	MBCCFR55	16	0x09BA	16

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
FlexRay	MBFIDR55	16	0x09BC	16
FlexRay	MBCCFR56	16	0x09C2	16
FlexRay	MBFIDR56	16	0x09C4	16
FlexRay	MBCCFR57	16	0x09CA	16
FlexRay	MBFIDR57	16	0x09CC	16
FlexRay	MBCCFR58	16	0x09D2	16
FlexRay	MBFIDR58	16	0x09D4	16
FlexRay	MBCCFR59	16	0x09DA	16
FlexRay	MBFIDR59	16	0x09DC	16
FlexRay	MBCCFR60	16	0x09E2	16
FlexRay	MBFIDR60	16	0x09E4	16
FlexRay	MBCCFR61	16	0x09EA	16
FlexRay	MBFIDR61	16	0x09EC	16
FlexRay	MBCCFR62	16	0x09F2	16
FlexRay	MBFIDR62	16	0x09F4	16
FlexRay	MBCCFR63	16	0x09FA	16
FlexRay	MBFIDR63	16	0x09FC	16
WKPU				
WKPU	NCR	32	0x0008	32
SSCM				
SSCM	ERROR	16	0x0006	16
SSCM	DEBUGPORT	16	0x0008	16
SSCM	SCTR	32	0x0024	32
PDI				
PDI	PDIMCR	32	0x0000	32
PDI	PDIADCCR	32	0x0004	32
PDI	PDISR	32	0x0008	32
PDI	PDIINTER	32	0x000C	32
PDI	PDIAFOR	32	0x0010	32
PDI	PDIAFSR	32	0x0014	32
PDI	PDIDMABAR	32	0x0018	32

Table 45-5. MPC5675K register protection (continued)

Module	Register	Register size (bits)	Register Offset (from module base)	Protected size (bits)
PDI	PDIT1R	32	0x0024	32
PDI	PDIT2R	32	0x0028	32

This page is intentinally left blank.

Chapter 46
Reset Generation Module (MC_RGM)

46.1 Introduction

The MC_RGM centralizes the different reset sources and manages the reset sequence of the device. It provides a register interface and the reset sequencer. Various registers are available to monitor and control the device reset sequence. The reset sequencer is a state machine that controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and controls the reset signals generated in the system.

Figure 46-1 shows the MC_RGM block diagram.

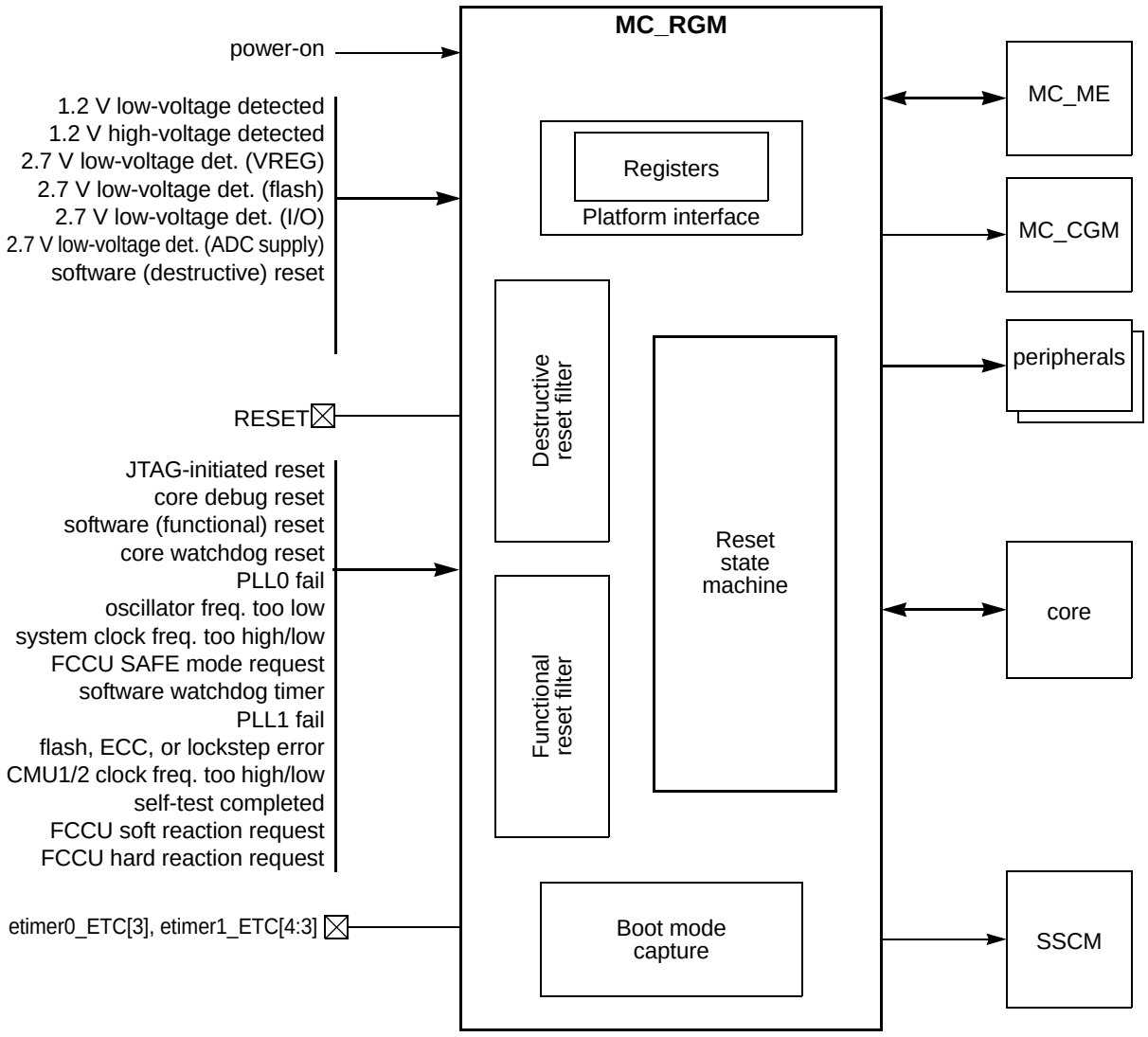


Figure 46-1. MC_RGM block diagram

46.1.1 Features

The MC_RGM contains the functionality for the following features:

- ‘Destructive’ reset management
- ‘Functional’ reset management
- Signaling of reset events after each reset sequence (reset status flags)
- Conversion of reset events to SAFE mode or interrupt request events
- Short reset sequence configuration
- Bidirectional reset behavior configuration
- Boot mode capture on $\overline{\text{RESET}}$ deassertion

46.1.2 Reset sources

The different reset sources are organized into two families: ‘destructive’ and ‘functional.’

- A ‘destructive’ reset source is associated with an event related to a critical—usually hardware—error or dysfunction. When a ‘destructive’ reset event occurs, the full reset sequence is applied to the device starting from PHASE0. This resets the full device ensuring a safe startup state for both digital and analog modules, and the memory content must be considered to be unknown. ‘Destructive’ resets include the following:
  - Power-on reset
  - 1.2 V low-voltage detected
  - 1.2 V high-voltage detected
  - 2.7 V low-voltage det. (VREG)
  - 2.7 V low-voltage det. (flash)
  - 2.7 V low-voltage det. (I/O)
  - 2.7 V low-voltage det. (ADC supply)
  - software (destructive) reset
- A ‘functional’ reset source is associated with an event related to a less critical—usually non-hardware—error or dysfunction. When a ‘functional’ reset event occurs, a partial reset sequence is applied to the device starting from PHASE1. In this case, most digital modules are reset normally, and the memory content must be considered to be unknown, while the state of analog modules or specific digital modules (for example, debug modules and flash memory modules) as well as the system memory content is preserved. ‘Functional’ resets include the following:
  - external reset
  - JTAG-initiated reset
  - core debug reset
  - software (functional) reset
  - core watchdog reset
  - PLL0 fail
  - oscillator freq. too low

- system clock freq. too high/low
- FCCU SAFE mode request
- software watchdog timer
- PLL1 fail
- flash, ECC, or lockstep error
- CMU1/2 clock freq. too high/low
- self-test completed
- FCCU soft reaction request
- FCCU hard reaction request

When a reset is triggered, the MC_RGM state machine is activated and proceeds through the different phases (that is, PHASE*n* states). Each phase is associated with a particular device reset being provided to the system. A phase is completed when all corresponding phase completion gates either from the system or internal to the MC_RGM are acknowledged. The device reset associated with the phase is then released and the state machine proceeds to the next phase, up to entering the IDLE phase. During this entire process, the MC_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some functional reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt issued to the core (see Section 46.3.1.3, Functional Event Reset Disable Register (RGM_FERD), and Section 46.3.1.5, Functional Event Alternate Request Register (RGM_FEAR), for ‘functional’ resets).

46.2 External signal description

The MC_RGM interfaces to the bidirectional reset pin $\overline{\text{RESET}}$ and the boot mode pins etimer0_ETC[4:3] and etimer1_ETC[3].

46.3 Memory map and register description

Table 46-1. MC_RGM register description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_4000	RGM_FES	Functional Event Status	halfword	read	read/write[1]	read/write[1]	on page 1584
0xC3FE_4002	RGM_DES	Destructive Event Status	halfword	read	read/write[1]	read/write[1]	on page 1586
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	halfword	read	read/write[2]	read/write[2]	on page 1588
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	halfword	read	read	read	on page 1590

Table 46-1. MC_RGM register description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_4008–0xC3FE_400E		Reserved					
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	halfword	read	read/write	read/write	on page 1591
0xC3FE_4012–0xC3FE_4016		Reserved					
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	halfword	read	read/write	read/write	on page 1592
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	halfword	read	read/write	read/write	on page 1594
0xC3FE_4020–0xC3FE_7FFF		Reserved					

1 Individual bits cleared on writing 1.

2 Write once: 0 = enable, 1 = disable.

NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

Table 46-2. MC_RGM memory map

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4000	RGM_FES / RGM_DES	R	F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG
		W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	F_SOFT_DEST	0	0	0	0	0	0	F_LVD27_ADC	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
		W	w1c	w1c							w1c	w1c	w1c	w1c			w1c	w1c

Table 46-2. MC_RGM memory map (continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	27 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FE_4004	RGM_FERD/ RGM_DERD	R	D_EXR	D_FCCU_HARD	D_FCCU_SOFT	D_ST_DONE	D_CMU12_FHL	D_FL_ECC_RCC	D_PLL1	D_SWT	D_FCCU_SAFE	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CWD	D_SOFT_FUNC	D_CORE	D_JTAG
		W																
		R	0	D_SOFT_DEST	0	0	0	0	0	0	D_LVD27_ADC	D_LVD27_IO	D_LVD27_FLASH	D_VREGLVD27_IO	0	0	D_HVD12	D_LVD12
		W																
0xC3FE_4008 ... 0xC3FE_400C	reserved																	
0xC3FE_4010	RGM_FEAR	R	0	0	0	0	AR_CMU12_FHL	0	AR_PLL1	0	0	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CWD	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_4014	reserved																	
0xC3FE_4018	RGM_FESS	R	SS_EXR	SS_FCCU_HARD	SS_FCCU_SOFT	SS_ST_DONE	SS_CMU12_FHL	SS_FL_ECC_RCC	SS_PLL1	SS_SWT	0	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CWD	SS_SOFT_FUNC	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 46-2. MC_RGM memory map (continued)

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_401C	RGM_FBRE	R	BE_EXR	BE_FCCU_HARD	BE_FCCU_SOFT	BE_ST_DONE	BE_CMU12_FHL	BE_FL_ECC_RCC	BE_PLL1	BE_SWT	0	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CWD	BE_SOFT_FUNC	BE_CORE	BE_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_4020 … 0xC3FE_7FFC	reserved																	

46.3.1 Register descriptions

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit halfwords, or 8-bit bytes. The bytes are ordered according to big endian. For example, the RGM_DES[8:15] register bits may be accessed as a word at address 0xC3FE_4000, as a halfword at address 0xC3FE_4002, or as a byte at address 0xC3FE_4003.

NOTE

Some fields may be read-only, and their reset value of 1 or 0 and the corresponding behavior cannot be changed.

46.3.1.1 Functional Event Status Register (RGM_FES)

This register contains the status of the last asserted functional reset sources. It can be accessed read/write in either supervisor mode or test mode. It can be accessed read-only in user mode. Register bits are cleared by writing 1 if the triggering event has already been cleared at the source.

“FCCU hard reaction request” and “FCCU soft reaction request” correspond to “long reset reaction” and “short reset reaction” of the FCCU module, respectively.

NOTE

If a ‘functional’ reset source is configured to generate a SAFE mode request or an interrupt request, software needs to clear the event in the source module at least three system clock cycles before it clears the associated RGM_FES status bit in order to avoid multiple SAFE mode requests or interrupts for the same event. In order to avoid having to count cycles, it is good practice for software to check whether the RGM_FES has been properly cleared, and if not, to clear it again.

Address 0xC3FE_4000 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	F_FCCU_HARD	F_FCCU_SOFT	F_ST_DONE	F_CMU12_FHL	F_FL_ECC_RCC	F_PLL1	F_SWT	F_FCCU_SAFE	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CWD	F_SOFT_FUNC	F_CORE	F_JTAG
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 46-2. Functional Event Status Register (RGM_FES)

Table 46-3. RGM_FES field descriptions

Field	Description
F_EXR	Flag for External Reset 0 No external reset event has occurred since either the last clear or the last destructive reset assertion. 1 An external reset event has occurred.
F_FCCU_HARD	Flag for FCCU hard reaction request 0 No FCCU hard reaction request event has occurred since either the last clear or the last destructive reset assertion. 1 An FCCU hard reaction request event has occurred.
F_FCCU_SOFT	Flag for FCCU soft reaction request 0 No FCCU soft reaction request event has occurred since either the last clear or the last destructive reset assertion. 1 An FCCU soft reaction request event has occurred.
F_ST_DONE	Flag for self-test completed 0 No self-test completed event has occurred since either the last clear or the last destructive reset assertion. 1 A self-test completed event has occurred.
F_CMU12_FHL	Flag for CMU1/2 clock freq. too high/low 0 No CMU1/2 clock freq. too high/low event has occurred since either the last clear or the last destructive reset assertion. 1 A CMU1/2 clock freq. too high/low event has occurred.
F_FL_ECC_RCC	Flag for flash, ECC, or lockstep error 0 No flash, ECC, or lockstep error event has occurred since either the last clear or the last destructive reset assertion. 1 A flash, ECC, or lockstep error event has occurred.
F_PLL1	Flag for PLL1 fail 0 No PLL1 fail event has occurred since either the last clear or the last destructive reset assertion. 1 A PLL1 fail event has occurred.
F_SWT	Flag for software watchdog timer 0 No software watchdog timer event has occurred since either the last clear or the last destructive reset assertion. 1 A software watchdog timer event has occurred.

Table 46-3. RGM_FES field descriptions (continued)

Field	Description
F_FCCU_SAFE	Flag for FCCU SAFE mode request 0 No FCCU SAFE mode request event has occurred since either the last clear or the last destructive reset assertion. 1 An FCCU SAFE mode request event has occurred.
F_CMU0_FHL	Flag for system clock freq. too high/low 0 No system clock freq. too high/low event has occurred since either the last clear or the last destructive reset assertion. 1 A system clock freq. too high/low event has occurred.
F_CMU0_OLR	Flag for oscillator freq. too low 0 No oscillator freq. too low event has occurred since either the last clear or the last destructive reset assertion. 1 An oscillator freq. too low event has occurred.
F_PLL0	Flag for PLL0 fail 0 No PLL0 fail event has occurred since either the last clear or the last destructive reset assertion. 1 A PLL0 fail event has occurred.
F_CWD	Flag for core watchdog reset 0 No core watchdog reset event has occurred since either the last clear or the last destructive reset assertion. 1 A core watchdog reset event has occurred.
F_SOFT_FUNC	Flag for software (functional) reset This bit is set when a mode change request has been made with the target mode = "0000". 0 No software (functional) reset event has occurred since either the last clear or the last destructive reset assertion. 1 A software (functional) reset event has occurred.
F_CORE	Flag for core debug reset 0 No debug control core reset event has occurred since either the last clear or the last destructive reset assertion 1 A debug control core reset event has occurred; this event can only be asserted when the DBCR0[RST] field is set by an external debugger. See the "Debug Support" chapter of the core reference manual for more details.
F_JTAG	Flag for JTAG-initiated reset 0 No JTAG-initiated reset event has occurred since either the last clear or the last destructive reset assertion. 1 A JTAG-initiated reset event has occurred.

46.3.1.2 Destructive Event Status Register (RGM_DES)

This register contains the status of the last asserted destructive reset sources. It can be accessed read/write in either supervisor mode or test mode. It can be accessed read-only in user mode. Register bits are cleared by writing 1.

NOTE

- The F_POR flag is also set when a low voltage is detected on the 1.2 V supply, even if the low voltage is detected after power-on has completed.

- The F_LVD27_VREG flag may still have the value "0" after a dip has occurred on the 2.7 V supply during a non-monotonic power-on sequence. The F_POR flag will, however, still be set in this case as expected after each power-on sequence.

- The F_HVD12 flag may still have the value "0" after an overshoot has occurred on the 1.2 V supply during a non-monotonic power-on sequence. The F_POR flag will, however, still be set in this case as expected after each power-on sequence.

- During power-up phase, the RGM.F_POR will be set to 1 if the bit is read after RESET_B negation. Additionally, it is possible that other LVD/HVD flags will be set and can be considered invalid if the F_POR bit is set at the same time.

Address 0xC3FE_4002 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	F_SOFT_DEST	0	0	0	0	0	0	F_LVD27_ADC	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	0	F_HVD12	F_LVD12
W	w1c	w1c							w1c	w1c	w1c	w1c			w1c	w1c
POR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 46-3. Destructive Event Status Register (RGM_DES)

Table 46-4. RGM_DES field descriptions

Field	Description
F_POR	Flag for power-on reset 0 No power-on event has occurred since the last clear. 1 A power-on event has occurred.
F_SOFT_DEST	Flag for software (destructive) reset 0 No software (destructive) reset event has occurred since either the last clear or the last power-on reset assertion. 1 A software (destructive) reset event has occurred.
F_LVD27_ADC	Flag for 2.7 V low-voltage det. (ADC supply) 0 No 2.7 V low-voltage det. (ADC supply) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage det. (ADC supply) event has occurred.
F_LVD27_IO	Flag for 2.7 V low-voltage det. (I/O) 0 No 2.7 V low-voltage det. (I/O) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage det. (I/O) event has occurred.

Table 46-4. RGM_DES field descriptions (continued)

Field	Description
F_LVD27_FLASH	Flag for 2.7 V low-voltage det. (flash) 0 No 2.7 V low-voltage det. (flash) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage det. (flash) event has occurred.
F_LVD27_VREG	Flag for 2.7 V low-voltage det. (VREG) 0 No 2.7 V low-voltage det. (VREG) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage det. (VREG) event has occurred.
F_HVD12	Flag for 1.2 V high-voltage detected 0 No 1.2 V high-voltage detected event has occurred since either the last clear or the last power-on reset assertion. 1 A 1.2 V high-voltage detected event has occurred.
F_LVD12	Flag for 1.2 V low-voltage detected 0 No 1.2 V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion. 1 A 1.2 V low-voltage detected event has occurred.

46.3.1.3 Functional Event Reset Disable Register (RGM_FERD)

This register provides dedicated bits to disable functional reset sources.When a functional reset source is disabled, the associated functional event triggers either a SAFE mode request or an interrupt request (see Section 46.3.1.5, Functional Event Alternate Request Register (RGM_FEAR)). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read-only in user mode. Second and third bytes of this register can be written only once after power-on reset.

CAUTION

It is important to clear the RGM_FES register before setting any of the bits in the RGM_FERD register to 1. Otherwise a redundant SAFE mode request or interrupt request may occur.

Address 0xC3FE_4004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	D_FCCU_HARD	D_FCCU_SOFT	D_ST_DONE	D_CMU12_FHL	D_FL_ECC_RCC	D_PLL1	D_SWT	D_FCCU_SAFE	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CWD	D_SOFT_FUNC	D_CORE	D_JTAG
W																
POR	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 46-4. Functional Event Reset Disable Register (RGM_FERD)

Table 46-5. RGM_FERD field descriptions

Field	Description
D_EXR	Disable External Reset 0 An external reset event triggers a reset sequence. 1 Reserved
D_FCCU_HARD	Disable FCCU hard reaction request 0 A FCCU hard reaction request event triggers a reset sequence. 1 Reserved
D_FCCU_SOFT	Disable FCCU soft reaction request 0 An FCCU soft reaction request event triggers a reset sequence. 1 Reserved
D_ST_DONE	Disable self-test completed 0 A self-test completed event triggers a reset sequence. 1 Reserved
D_CMU12_FHL	Disable CMU1/2 clock freq. too high/low 0 A CMU1/2 clock freq. too high/low event triggers a reset sequence. 1 A CMU1/2 clock freq. too high/low event generates either a SAFE mode or an interrupt request, depending on the value of RGM_FEAR[AR_CMU12_FHL].
D_FL_ECC_RCC	Disable flash, ECC, or lockstep error 0 A flash, ECC, or lockstep error event triggers a reset sequence. 1 A flash, ECC, or lockstep error event generates a SAFE mode request.
D_PLL1	Disable PLL1 fail 0 A PLL1 fail event triggers a reset sequence. 1 A PLL1 fail event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_PLL1].
D_SWT	Disable software watchdog timer 0 A software watchdog timer event triggers a reset sequence. 1 Reserved
D_FCCU_SAFE	Disable FCCU SAFE mode request 0 Reserved 1 An FCCU SAFE mode request event generates a SAFE mode request.
D_CMU0_FHL	Disable system clock freq. too high/low 0 A system clock freq. too high/low event triggers a reset sequence. 1 A system clock freq. too high/low event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CMU0_FHL].
D_CMU0_OLR	Disable oscillator freq. too low 0 An oscillator freq. too low event triggers a reset sequence. 1 An oscillator freq. too low event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CMU0_OLR].
D_PLL0	Disable PLL0 fail 0 A PLL0 fail event triggers a reset sequence. 1 A PLL0 fail event generates either a SAFE mode or an interrupt request, depending on the value of RGM_FEAR[AR_PLL0].
D_CWD	Disable core watchdog reset 0 A core watchdog reset event triggers a reset sequence. 1 A core watchdog reset event generates either a SAFE mode or an interrupt request, depending on the value of RGM_FEAR[AR_CWD].

Table 46-5. RGM_FERD field descriptions (continued)

Field	Description
D_SOFT_FUNC	Disable software (functional) reset 0 A software (functional) reset event triggers a reset sequence. 1 Reserved
D_CORE	Disable core debug reset 0 A core debug reset event triggers a reset sequence. 1 Reserved
D_JTAG	Disable JTAG-initiated reset 0 A JTAG-initiated reset event triggers a reset sequence. 1 Reserved

46.3.1.4 Destructive Event Reset Disable Register (RGM_DERD)

This register provides dedicated bits to disable particular destructive reset sources. It can be accessed in read-only in supervisor mode, test mode, and user mode.

Address 0xC3FE_4006 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	D_SOFT_DEST	0	0	0	0	0	0	D_LVD27_ADC	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	0	D_HVD12	D_LVD12
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 46-5. Destructive Event Reset Disable Register (RGM_DERD)

Table 46-6. RGM_DERD field descriptions

Field	Description
D_SOFT_DEST	Disable software (destructive) reset 0 A software (destructive) reset event triggers a reset sequence. 1 Reserved
D_LVD27_ADC	Disable 2.7 V low-voltage det. (ADC supply) 0 A 2.7 V low-voltage det. (ADC supply) event triggers a reset sequence. 1 Reserved
D_LVD27_IO	Disable 2.7 V low-voltage det. (I/O) 0 A 2.7 V low-voltage det. (I/O) event triggers a reset sequence. 1 Reserved
D_LVD27_FLASH	Disable 2.7 V low-voltage det. (flash) 0 A 2.7 V low-voltage det. (flash) event triggers a reset sequence. 1 Reserved

Table 46-6. RGM_DERD field descriptions (continued)

Field	Description
D_LVD27_VREG	Disable 2.7 V low-voltage det. (VREG) 0 A 2.7 V low-voltage det. (VREG) event triggers a reset sequence. 1 Reserved
D_HVD12	Disable 1.2 V high-voltage detected 0 A 1.2 V high-voltage detected event triggers a reset sequence. 1 Reserved
D_LVD12	Disable 1.2 V low-voltage detected 0 A 1.2 V low-voltage detected event triggers a reset sequence. 1 Reserved

46.3.1.5 Functional Event Alternate Request Register (RGM_FEAR)

This register defines an alternate request to be generated when a reset on a functional event has been disabled. The alternate request can be either a SAFE mode request to MC_ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read-only in user mode. The following reset sources can only generate a SAFE mode request:

- Flash memory
- ECC
- Lockstep error

Address 0xC3FE_4010 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	AR_CMU12_FHL	0	AR_PLL1	0	0	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CWD	0	0	0
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 46-6. Functional Event Alternate Request Register (RGM_FEAR)

Table 46-7. Functional Event Alternate Request Register (RGM_FEAR) field descriptions

Field	Description
AR_CMU12_FHL	Alternate Request for CMU1/2 clock freq. too high/low 0 Generates a SAFE mode request on a CMU1/2 clock freq. too high/low event if the reset is disabled. 1 Generates an interrupt request on a CMU1/2 clock freq. too high/low event if the reset is disabled.
AR_PLL1	Alternate Request for PLL1 fail 0 Generates a SAFE mode request on a PLL1 fail event if the reset is disabled. 1 Generates an interrupt request on a PLL1 fail event if the reset is disabled

Table 46-7. Functional Event Alternate Request Register (RGM_FEAR) field descriptions (continued)

Field	Description
AR_CMU0_FHL	Alternate Request for system clock freq. too high/low 0 Generates a SAFE mode request on a system clock freq. too high/low event if the reset is disabled. 1 Generates an interrupt request on a system clock freq. too high/low event if the reset is disabled.
AR_CMU0_OLR	Alternate Request for oscillator freq. too low 0 Generates a SAFE mode request on an oscillator freq. too low event if the reset is disabled. 1 Generates an interrupt request on an oscillator freq. too low event if the reset is disabled.
AR_PLL0	Alternate Request for PLL0 fail 0 Generates a SAFE mode request on a PLL0 fail event if the reset is disabled. 1 Generates an interrupt request on a PLL0 fail event if the reset is disabled.
AR_CWD	Alternate Request for core watchdog reset 0 Generates a SAFE mode request on a core watchdog reset event if the reset is disabled. 1 Generates an interrupt request on a core watchdog reset event if the reset is disabled.

46.3.1.6 Functional Event Short Sequence Register (RGM_FESS)

This register defines which reset sequence occurs when a functional reset sequence is triggered. The functional reset sequence can either start from PHASE1 or from PHASE3, skipping PHASE1 and PHASE2.

NOTE

This could be useful for a fast reset sequence, for example to skip flash memory reset.

However, short sequence resets should not be used when in a mode with the flash memory configured to a power-down state. In this case, a full PHASE1 reset is required in order to power up the flash memory. Therefore, the RGM_FESS register should be set to 0x2000.

This registercan be accessed read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Address 0xC3FE_4018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS_EXR	SS_FCCU_HARD	SS_FCCU_SOFT	SS_ST_DONE	SS_CMU12_FHL	SS_FL_ECC_RCC	SS_PLL1	SS_SWT	0	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CWD	SS_SOFT_FUNC	SS_CORE	SS_JTAG
W																
POR	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 46-7. Functional Event Short Sequence Register (RGM_FESS)

Table 46-8. RGM_FESS field descriptions

Field	Description
SS_EXR	Short Sequence for External Reset 0 The reset sequence triggered by an external reset event starts from PHASE1. 1 The reset sequence triggered by an external reset event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_FCCU_HARD	Short Sequence for FCCU hard reaction request 0 The reset sequence triggered by an FCCU hard reaction request event starts from PHASE1. 1 Reserved
SS_FCCU_SOFT	Short Sequence for FCCU soft reaction request 0 Reserved 1 The reset sequence triggered by an FCCU soft reaction request event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_ST_DONE	Short Sequence for self-test completed 0 The reset sequence triggered by a self-test completed event starts from PHASE1. 1 Reserved
SS_CMU12_FHL	Short Sequence for CMU1/2 clock freq. too high/low 0 The reset sequence triggered by a CMU1/2 clock freq. too high/low event starts from PHASE1. 1 The reset sequence triggered by a CMU1/2 clock freq. too high/low event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_FL_ECC_RCC	Short Sequence for flash, ECC, or lockstep error 0 The reset sequence triggered by a flash, ECC, or lockstep error event starts from PHASE1. 1 Reserved
SS_PLL1	Short Sequence for PLL1 fail 0 The reset sequence triggered by a PLL1 fail event starts from PHASE1. 1 The reset sequence triggered by a PLL1 fail event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_SWT	Short Sequence for software watchdog timer 0 The reset sequence triggered by a software watchdog timer event starts from PHASE1. 1 Reserved
SS_CMU0_FHL	Short Sequence for system clock freq. too high/low 0 The reset sequence triggered by a system clock freq. too high/low event starts from PHASE1. 1 The reset sequence triggered by a system clock freq. too high/low event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_CMU0_OLR	Short Sequence for oscillator freq. too low 0 The reset sequence triggered by an oscillator freq. too low event starts from PHASE1. 1 The reset sequence triggered by an oscillator freq. too low event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_PLL0	Short Sequence for PLL0 fail 0 The reset sequence triggered by a PLL0 fail event starts from PHASE1. 1 The reset sequence triggered by a PLL0 fail event starts from PHASE3, skipping PHASE1 and PHASE2.

Table 46-8. RGM_FESS field descriptions (continued)

Field	Description
SS_CWD	Short Sequence for core watchdog reset 0 The reset sequence triggered by a core watchdog reset event starts from PHASE1. 1 The reset sequence triggered by a core watchdog reset event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_SOFT_FUNC	Short Sequence for software (functional) reset 0 The reset sequence triggered by a software (functional) reset event starts from PHASE1. 1 The reset sequence triggered by a software (functional) reset event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_CORE	Short Sequence for core debug reset 0 The reset sequence triggered by a core debug reset event starts from PHASE1. 1 The reset sequence triggered by a core debug reset event starts from PHASE3, skipping PHASE1 and PHASE2.
SS_JTAG	Short Sequence for JTAG-initiated reset 0 The reset sequence triggered by a JTAG-initiated reset event starts from PHASE1. 1 The reset sequence triggered by a JTAG-initiated reset event starts from PHASE3, skipping PHASE1 and PHASE2.

NOTE

This register is reset on any enabled ‘destructive’ or ‘functional’ reset event.

46.3.1.7 Functional Bidirectional Reset Enable Register (RGM_FBRE)

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Address 0xC3FE_401C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE_EXR	BE_FCCU_HARD	BE_FCCU_SOFT	BE_ST_DONE	BE_CMU12_FHL	BE_FL_ECC_RCC	BE_PLL1	BE_SWT	0	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CWD	BE_SOFT_FUNC	BE_CORE	BE_JTAG
W																
POR	0	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1

Figure 46-8. Functional Bidirectional Reset Enable Register (RGM_FBRE)

Table 46-9. Functional Bidirectional Reset Enable Register (RGM_FBRE) field descriptions

Field	Description
BE_EXR	Bidirectional Reset Enable for External Reset 0 $\overline{\text{RESET}}$ is asserted on an external reset event if the reset is enabled. 1 $\overline{\text{RESET}}$ is not asserted on an external reset event.
BE_FCCU_HARD	Bidirectional Reset Enable for FCCU hard reaction request 0 $\overline{\text{RESET}}$ is asserted on an FCCU hard reaction request event if the reset is enabled. 1 $\overline{\text{RESET}}$ is not asserted on an FCCU hard reaction request event.

Table 46-9. Functional Bidirectional Reset Enable Register (RGM_FBRE) field descriptions (continued)

Field	Description
BE_FCCU_SOFT	Bidirectional Reset Enable for FCCU soft reaction request 0 RESET is asserted on an FCCU soft reaction request event if the reset is enabled. 1 RESET is not asserted on an FCCU soft reaction request event.
BE_ST_DONE	Bidirectional Reset Enable for self-test completed 0 RESET is asserted on a self-test completed event if the reset is enabled. 1 RESET is not asserted on a self-test completed event.
BE_CMU12_FHL	Bidirectional Reset Enable for CMU1/2 clock freq. too high/low 0 RESET is asserted on a CMU1/2 clock freq. too high/low event if the reset is enabled. 1 RESET is not asserted on a CMU1/2 clock freq. too high/low event.
BE_FL_ECC_RCC	Bidirectional Reset Enable for flash, ECC, or lockstep error 0 RESET is asserted on a flash, ECC, or lockstep error event if the reset is enabled. 1 RESET is not asserted on a flash, ECC, or lockstep error event.
BE_PLL1	Bidirectional Reset Enable for PLL1 fail 0 RESET is asserted on a PLL1 fail event if the reset is enabled. 1 RESET is not asserted on a PLL1 fail event.
BE_SWT	Bidirectional Reset Enable for software watchdog timer 0 RESET is asserted on a software watchdog timer event if the reset is enabled. 1 RESET is not asserted on a software watchdog timer event.
BE_CMU0_FHL	Bidirectional Reset Enable for system clock freq. too high/low 0 RESET is asserted on a system clock freq. too high/low event if the reset is enabled. 1 RESET is not asserted on a system clock freq. too high/low event.
BE_CMU0_OLR	Bidirectional Reset Enable for oscillator freq. too low 0 RESET is asserted on an oscillator freq. too low event if the reset is enabled. 1 RESET is not asserted on an oscillator freq. too low event.
BE_PLL0	Bidirectional Reset Enable for PLL0 fail 0 RESET is asserted on a PLL0 fail event if the reset is enabled. 1 RESET is not asserted on a PLL0 fail event.
BE_CWD	Bidirectional Reset Enable for core watchdog reset 0 RESET is asserted on a core watchdog reset event if the reset is enabled. 1 RESET is not asserted on a core watchdog reset event.
BE_SOFT_FUNC	Bidirectional Reset Enable for software (functional) reset 0 RESET is asserted on a software (functional) reset event if the reset is enabled. 1 RESET is not asserted on a software (functional) reset event.
BE_CORE	Bidirectional Reset Enable for core debug reset 0 RESET is asserted on a core debug reset event if the reset is enabled. 1 RESET is not asserted on a core debug reset event.
BE_JTAG	Bidirectional Reset Enable for JTAG-initiated reset 0 RESET is asserted on a JTAG-initiated reset event if the reset is enabled. 1 RESET is not asserted on a JTAG-initiated reset event.

46.4 Functional description

46.4.1 Reset state machine

The main role of MC_RGM is the generation of the reset sequence, which ensures that the correct parts of the device are reset based on the reset source event. This is summarized in Table 46-10.

Table 46-10. MC_RGM reset implications

Source	What gets reset	External reset assertion[1]	Boot mode capture
Power-on reset	All	Yes	Yes
'Destructive' resets	All except some clock/reset management	Yes	Yes
External reset	All except some clock/reset management and debug	Programmable[2]	Yes
'Functional' resets	All except some clock/reset management and debug	Programmable[2]	Programmable[3]
Shortened 'functional' resets[4]	Flip-flops except some clock/reset management	Programmable[2]	Programmable[3]

1 'External reset assertion' means that the $\overline{\text{RESET}}$ pin is asserted by the MC_RGM until the end of reset PHASE3.

2 The assertion of the external reset is controlled via the RGM_FBRE register.

3 The boot mode is captured if the external reset is asserted.

4 The short sequence is enabled via the RGM_FESS register.

NOTE

JTAG logic has its own independent reset control and is not controlled by the MC_RGM in any way.

The reset sequence comprises five phases managed by a state machine. This state machine ensures that all phases are correctly processed through waiting for a minimum duration and until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in Figure 46-9.

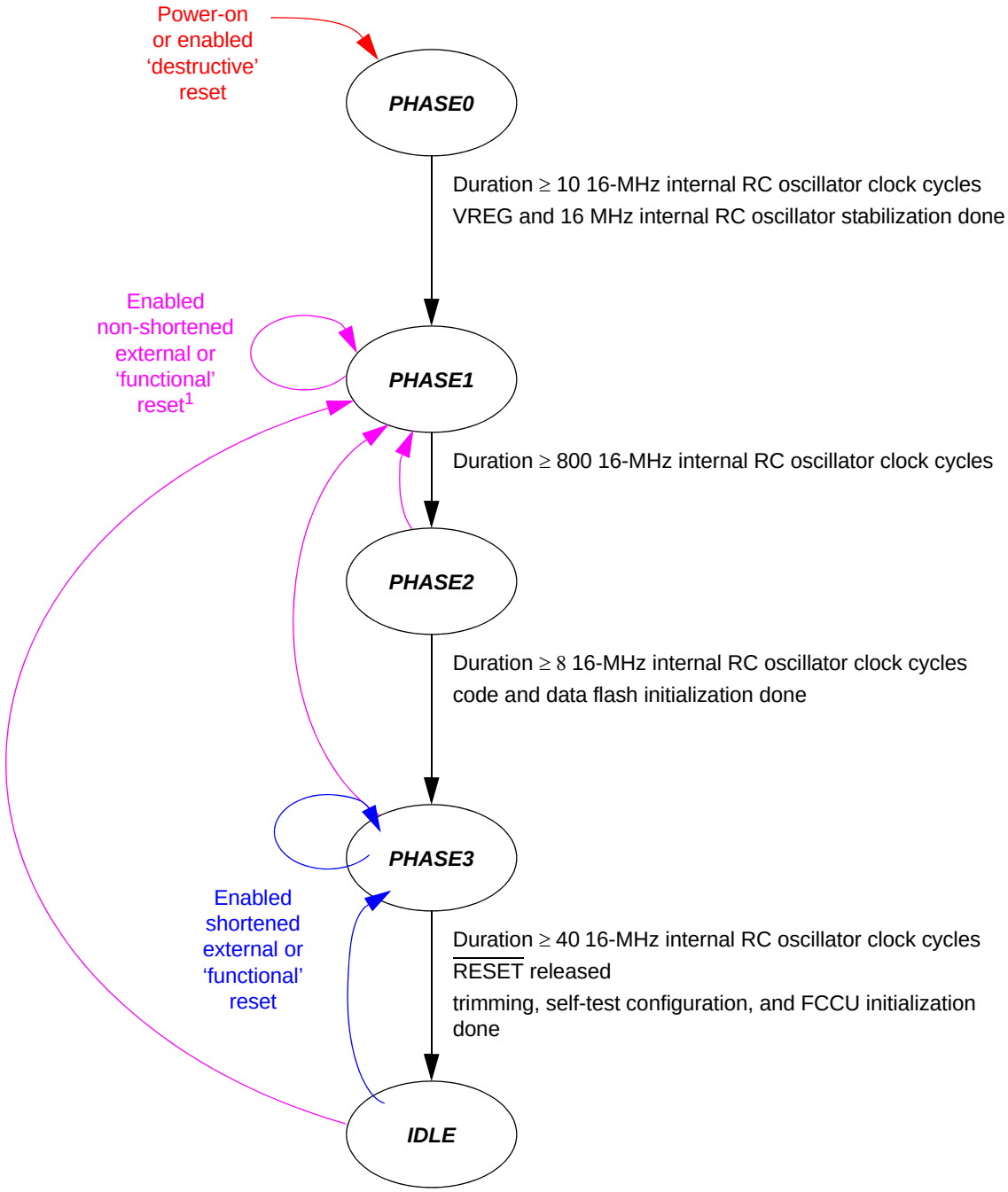


Figure 46-9. MC_RGM state machine

46.4.1.1 PHASE0 phase

This phase is entered immediately from any phase on a power-on or enabled 'destructive' reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- All enabled 'destructive' resets have been processed.
- All processes that need to be done in PHASE0 are completed.

— VREG and 16 MHz internal RC oscillator stabilization

- A minimum of 3 16-MHz internal RC oscillator clock cycles have elapsed since power-up completion and the last enabled ‘destructive’ reset event.

46.4.1.2 PHASE1 phase

This phase is entered either on exit from PHASE0 or immediately from PHASE2, PHASE3, or IDLE on a non-masked external or ‘functional’ reset event if it has not been configured to trigger a ‘short’ sequence. The reset state machine exits PHASE1 and enters PHASE2 on verification of the following:

- All enabled, non-shortened ‘functional’ resets have been processed.
- A minimum of 800 16-MHz internal RC oscillator clock cycles have elapsed since the last enabled external or non-shortened ‘functional’ reset event.

46.4.1.3 PHASE2 phase

This phase is entered on exit from PHASE1. The reset state machine exits PHASE2 and enters PHASE3 on verification of the following:

- All processes that need to be done in PHASE2 are completed.
  — code and data flash initialization
- A minimum of 8 16-MHz internal RC oscillator clock cycles have elapsed since entering PHASE2.

46.4.1.4 PHASE3 phase

This phase is entered either on exit from PHASE2 or immediately from IDLE on an enabled, shortened ‘functional’ reset event. The reset state machine exits PHASE3 and enters IDLE on verification of the following:

- All processes that need to be done in PHASE3 are completed.
  — trimming, self-test configuration, and FCCU initialization
- A minimum of 40 16-MHz internal RC oscillator clock cycles have elapsed since the last enabled, shortened ‘functional’ reset event.

46.4.1.5 IDLE phase

This is the final phase and is entered on exit from PHASE3. When this phase is reached, the MC_RGM releases control of the system to the platform and waits for new reset events that can trigger a reset sequence.

46.4.2 Destructive resets

A ‘destructive’ reset indicates that an event has occurred after which critical register or memory content can no longer be guaranteed.

The status flag associated with a given ‘destructive’ reset event (RGM_DES.F_<*destructive reset*> bit) is set when the ‘destructive’ reset is asserted and the power-on reset is not asserted. It is possible for multiple

status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The device's low-voltage detector threshold ensures that, when 1.2 V low-voltage detected is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given 'destructive' reset is enabled, the MC_RGM ensures that the associated reset event is correctly triggered to the full system. However, if the given 'destructive' reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled destructive reset triggers a reset sequence starting from the beginning of PHASE0.

46.4.3 External reset

The MC_RGM manages the external reset coming from $\overline{\text{RESET}}$. The detection of a falling edge on $\overline{\text{RESET}}$ starts the reset sequence from the beginning of PHASE1.

The status flag associated with the external reset falling edge event (RGM_FES.F_EXR bit) is set when the external reset is asserted and the power-on reset is not asserted.

The external reset can optionally be disabled by writing bit RGM_FERD.D_EXR.

NOTE

The RGM_FERD register can be written only once between two power-on reset events.

An enabled external reset normally triggers a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by the external reset. When RGM_FESS.SS_EXR is set, the external reset triggers a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially when an external reset should not reset the flash memory.

The MC_RGM may also assert the external reset if the reset sequence was triggered by one of the following:

- A power-on reset
- A 'destructive' reset event
- An external reset event
- A 'functional' reset event configured via the RGM_FBRE register to assert the external reset

In this case, the external reset is asserted until the end of PHASE3.

CAUTION

On an external reset that is configured to be 'long' the device may remain in reset if the system clock is configured to be sourced by a clock source other than the 16 MHz Internal RC Oscillator (IRCOSC). Recovery from the reset in this case can only be achieved via a powerdown and power-up cycle.

The failure condition can only be seen with the following Reset Generation Module (MC_RGM) settings for Functional Event Short Sequence register, External Reset field (RGM_FESS[SS_EXR]) and Functional Bidirectional Reset Enable register, External Reset field (RGM_FBRE[BE_EXR]):

- RGM_FESS[SS_EXR] = 0b0 (long external reset)

- RGM_FBRE[BE_EXR] = 0b0 (asserted on external reset event)

Note 1: This condition can only occur if the cause of the device reset was the external reset assertion. It does not occur if, for example, the device reset was due to a power-on.

Note 2: RGM_FESS[SS_EXR] = 0b0 and RGM_FBRE[BE_EXR] = 0b0 are the default settings out of power-on reset (POR).

There are two possible recommended sequences. In both, the recommended sequence takes effect only after software has reconfigured the MC_RGM. Therefore, in order to ensure that the issue cannot occur after

POR exit and before the software has executed the recommended sequence, the system clock must not be re-configured in the Mode Entry module (MC_ME) to be sourced by a clock source other than the IRCOSC until after the recommended sequence has been executed.

Recommended sequence #1:

Always configure the external reset event to prevent the external reset output to be driven by the MC_RGM by writing 0b1 to RGM_FBRE[BE_EXR].

If the external reset has been configured to be long (RGM_FESS[SS_EXR] = 0b0) and self testing has been enabled via the flash option, the external reset pin will still be asserted from the time of external assertion until reset sequence exit after start-up self test execution.

If the external reset has been configured to be long (RGM_FESS[SS_EXR] = 0b0) and self testing has been disabled via the flash option, the external reset pin will still be asserted from the time of external assertion until the chip configuration is loaded from the flash during reset

PHASE3.

If the external reset has been configured to be short (RGM_FESS[SS_EXR] = 0b1), the external reset pin will still be released as soon as it is no longer asserted from off-chip.

Recommended sequence #2:

Always configure the external reset as 'short' by writing 0b1 to RGM_FESS[SS_EXR]. In addition, use software to trigger a long 'functional' or 'destructive' reset via the Mode Entry module (MC_ME) if flash initialization or start-up self test is required.

The impact of this recommended sequence is the additional time that the device is in reset (due to the short reset sequence triggered by the external reset) and the overhead required for software to check the reset status and request a software reset.

46.4.4 Functional resets

A ‘functional’ reset indicates that an event has occurred after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given ‘functional’ reset event (RGM_FES.F_<*functional reset*> bit) is set when the ‘functional’ reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software’s responsibility to determine which reset source is the most critical for the application.

The ‘functional’ reset can be optionally disabled by software writing bit RGM_FERD.D_<*functional reset*>.

NOTE

The RGM_FERD register can be written only once between two power-on reset events.

An enabled ‘functional’ reset normally triggers a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by a functional reset. When RGM_FESS.SS_<*functional reset*> is set, the associated ‘functional’ reset triggers a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially in case a functional reset should not reset the flash module.

See Chapter 35, Mode Entry Module (MC_ME), for details on the STANDBY0 and DRUN modes.

46.4.5 Alternate event generation

The MC_RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the MC_RGM normally enters the reset sequence. Alternatively, it is possible for some reset source events to be converted from a reset to ei*ther* a SAFE mode request issued to the MC_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM_FERD and RGM_FEAR registers as shown in Table 46-11.

Table 46-11. MC_RGM alternate event selection

RGM_FERD bit value	RGM_FEAR bit value	Generated event
0	X	Reset
1	0	SAFE mode request
1	1	Interrupt request

The alternate event is cleared by deasserting the source of the request (that is, at the reset source that caused the alternate request) and also clearing the appropriate RGM_FES status bit.

NOTE

Alternate requests (SAFE mode as well as interrupt requests) are generated regardless of whether the system clock is running.

NOTE

If a masked ‘functional’ reset event that is configured to generate a SAFE mode/interrupt request occurs during PHASE1, it is ignored, and the MC_RGM does not send a safe mode/interrupt request to the MC_ME.

46.4.6 Boot mode capturing

The MC_RGM samples etimer0_ETC[4:3] and etimer1_ETC[3] whenever $\overline{\text{RESET}}$ is asserted until five 16-MHz internal RC oscillator clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after $\overline{\text{RESET}}$ has been deasserted for subsequent boots after reset sequences during which $\overline{\text{RESET}}$ is not asserted.

NOTE

In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that $\overline{\text{RESET}}$ is asserted.

NOTE

$\overline{\text{RESET}}$ can be asserted as a consequence of the internal reset generation. This forces re-sampling of the boot mode pins. (See Table 46-10 for details.)

Chapter 47
Semaphore Unit (SEMA4)

47.1 Introduction

In a dual-processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The SEMA4 provides the hardware support needed in multi-core systems for implementing semaphores and providing a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

The MPC5675K contains two SEMA4 units.

On the MPC5675K, the SEMA4 unit is intended for use when using the device in Decoupled Parallel Mode (DPM). When using the device in Lock Step Mode (LSM), the SEMA4 unit is disabled and its interrupt sources deasserted.

In LSM, an access to the SEMA4 module results in an error that the core takes as a machine-check exception.

In this chapter, the two instantiations of the e200z7 core on the MPC5675K are referred to as e200z7_0 and e200z7_1. The e200z7_0 instantiation runs from reset in DPM.

47.1.1 Block diagram

Figure 47-1 is a simplified block diagram of the SEMA4 unit that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

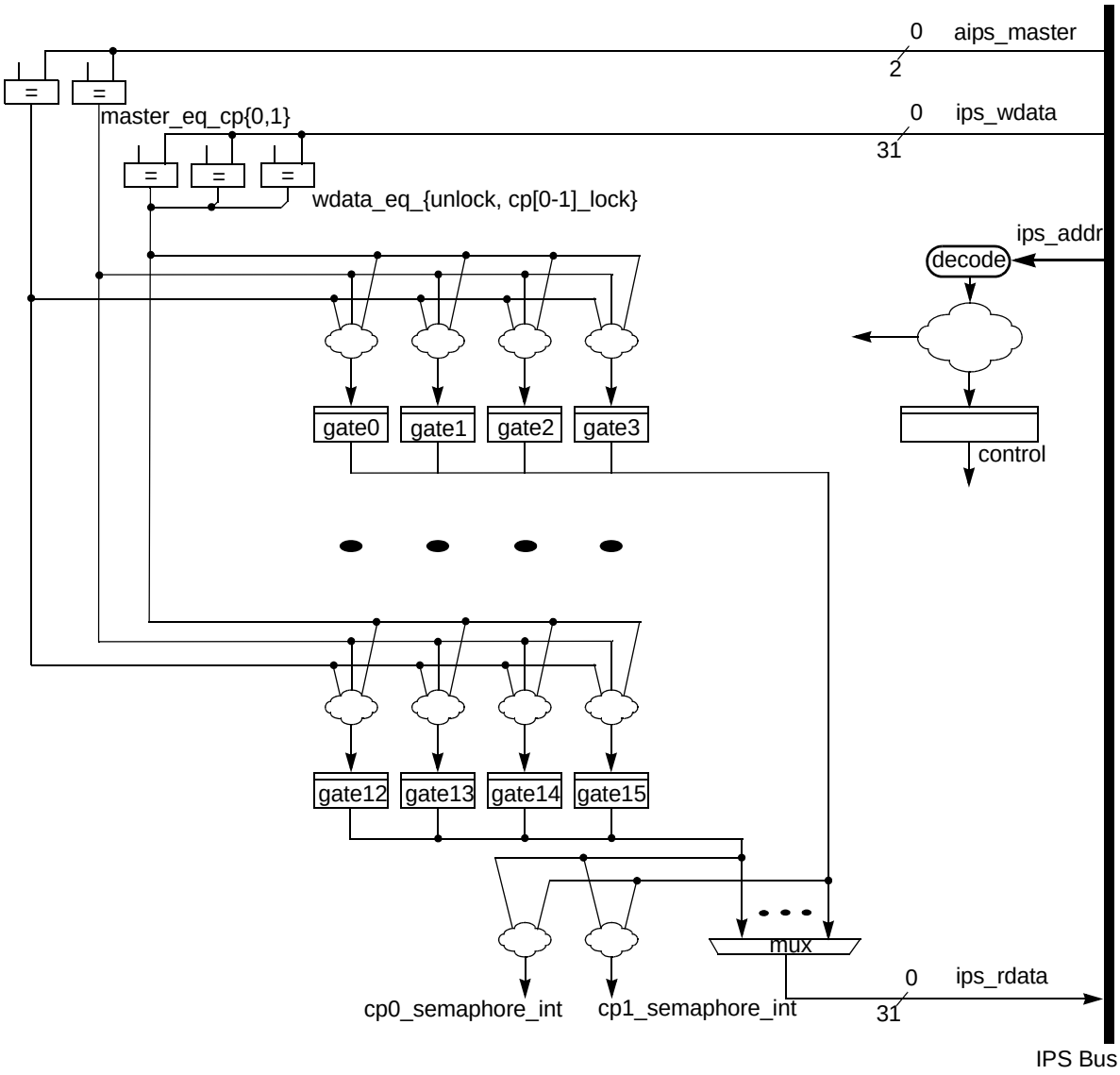


Figure 47-1. SEMA4 block diagram

47.1.2 Features

The SEMA4 unit implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration.
- Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes.
  - Three-state implementation
  - If gate = 0b00, then state = unlocked
  - If gate = 0b01, then state = locked by e200z7_0 (master ID = 0)

  - If gate = 0b10, then state = locked by e200z7_1 (master ID = 1)
  - Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
  - After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
- Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked.
- Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability.

47.1.3 Modes of operation

The SEMA4 unit does not support any special modes of operation.

47.2 Signal description

The SEMA4 unit does not include any external signals.

47.3 Memory map and register description

The SEMA4 programming model map is shown in Table 47-2. The address of each register is given as an offset to the SEMA4 base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 47-1. SEMA4 module base addresses

Mode	Module	Module base address
LSM	SEMA4_0	0xFFF2_4000
	SEMA4_1	0x8FF2_4000
DPM	SEMA4_0	0xFFF2_4000 (same as LSM)
	SEMA4_1	0x8FF2_4000

NOTE

The SEMA4 units operate only in DPM. In LSM, the SEMA4 units are disabled.

Table 47-2. SEMA4 memory map

Offset from SEMA4_BASE	Register	Access[1]	Reset Value[2]	Location
0x0000	Semaphores gate 0 (SEMA4_Gate00)	R/W	0x00	on page 1606
0x0001	Semaphores gate 1 (SEMA4_Gate01)	R/W	0x00	on page 1606
0x0002	Semaphores gate 2 (SEMA4_Gate02)	R/W	0x00	on page 1606
0x0003	Semaphores gate 3 (SEMA4_Gate03)	R/W	0x00	on page 1606

Table 47-2. SEMA4 memory map (continued)

Offset from SEMA4_BASE	Register	Access[1]	Reset Value[2]	Location
0x0004	Semaphores gate 4 (SEMA4_Gate04)	R/W	0x00	on page 1606
0x0005	Semaphores gate 5 (SEMA4_Gate05)	R/W	0x00	on page 1606
0x0006	Semaphores gate 6 (SEMA4_Gate06)	R/W	0x00	on page 1606
0x0007	Semaphores gate 7 (SEMA4_Gate07)	R/W	0x00	on page 1606
0x0008	Semaphores gate 8 (SEMA4_Gate08)	R/W	0x00	on page 1606
0x0009	Semaphores gate 9 (SEMA4_Gate09)	R/W	0x00	on page 1606
0x000A	Semaphores gate 10 (SEMA4_Gate10)	R/W	0x00	on page 1606
0x000B	Semaphores gate 11 (SEMA4_Gate11)	R/W	0x00	on page 1606
0x000C	Semaphores gate 12 (SEMA4_Gate12)	R/W	0x00	on page 1606
0x000D	Semaphores gate 13 (SEMA4_Gate13)	R/W	0x00	on page 1606
0x000E	Semaphores gate 14 (SEMA4_Gate14)	R/W	0x00	on page 1606
0x000F	Semaphores gate 15 (SEMA4_Gate15)	R/W	0x00	on page 1606
0x0010–0x003F	Reserved			
00x040	Semaphores CP0 IRQ notification enable (SEMA4_CP0INE)	R/W	0x0000	on page 1607
0x0042–0x0047	Reserved			
0x0048	Semaphores CP1 IRQ notification enable (SEMA4_CP1INE)	R/W	0x0000	on page 1607
0x004A–0x07F	Reserved			
0x0080	Semaphores CP0 IRQ notification (SEMA4_CP0NTF)	R	0x0000	on page 1608
0x008 2–00x087	Reserved			
0x0088	Semaphores CP1 IRQ notification (SEMA4_CP1NTF)	R	0x0000	on page 1607
0x008A–0x00FF	Reserved			
0x0100	Semaphores reset gate (SEMA4_RSTGT)	R/W	0x0000	on page 1609
0x0102	Reserved			
0x0104	Semaphores reset IRQ notification (SEMA4_RSTNTF)	R/W	0x0000	on page 1610
0x0106–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where 'H' is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

47.3.1 Semaphores gate *n* register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted

write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update only the state of a single gate. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: *n* (*n* = 0x0, 0x1,0x2,..., 0xF) Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	GTFSM	
W								
Reset	0	0	0	0	0	0	0	0

Figure 47-2. SEMA4 gate *n* register (SEMA4_GATE*n*)

Table 47-3. SEMA4_GATE*n* field descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by processor 0. 10 The gate has been locked by processor 1. 11 This state encoding is never used and is therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. **Note:** The state of the gate reflects the last processor that locked it, which can be useful during system debug.

47.3.2 Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits are read as zeroes and writes are ignored.

Offset: 0x0040 (SEMA4_CP0INE)
0x0048 (SEMA4_CP1INE)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INE0	INE1	INE2	INE3	INE4	INE5	INE6	INE7	INE8	INE9	INE10	INE11	INE12	INE13	INE14	INE15
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 47-3. Semaphores processor *n* IRQ notification enable (SEMA4_CP{0,1}INE)

Table 47-4. SEMA4_CP{0,1}NTF field descriptions

Field	Description
INE*n*	Interrupt Request Notification Enable *n*. This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate *n*. 0 Generation of the notification interrupt is disabled. 1 Generation of the notification interrupt is enabled.

47.3.3 Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

1. When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
2. After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
3. When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: 0x0080 (SEMA4_CP0NTF)
0x0088 (SEMA4_CP1NTF)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 47-4. Semaphores processor *n* IRQ notification (SEMA4_CP{0,1}NTF)

Table 47-5. SEMA4_CP{0,1}NTF field descriptions

Field	Description
GN*n*	Gate *n* Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate *n*. 0 No notification interrupt generated. 1 Notification interrupt generated.

47.3.4 Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the SEMA4 unit implements a secure reset mechanism that allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify that a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) that implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]), and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: 0x0100 (SEMA4_RSTGT) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTGSM		0	RSTGMS			RSTGTN							
W	RSTGDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 47-5. Semaphores (secure) reset gate *n* (SEMA4_RSTGT)

Table 47-6. SEMA4_RSTGT field descriptions

Field	Description
RSTGSM	Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and therefore reserved. Reads of the SEMA4_RSTGT register return the encoded state machine value. The RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.
RSTGMS	Reset Gate Bus Master. This 3-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. 000 Bus master is Core_0. 001 Bus master is Core_1. All other values are reserved.
RSTGTN	Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write. If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.
RSTGDP	Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xE2 while the second write requires RSTGDP = 0x1D.

47.3.5 Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the SEMA4 unit implements a secure reset mechanism that allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN]) specifies the notification(s) to be reset. This field can specify that a single notification be cleared or that all notifications are cleared.

3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTNSM		0	RSTNMS			RSTNTN							
W	RSTNDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 47-6. Semaphores (secure) Reset IRQ notification (SEMA4_RSTNTF)

Table 47-7. SEMA4_RSTNTF field descriptions

Field	Description
RSTNSM	Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and is therefore reserved. Reads of the SEMA4_RSTNTF register return the encoded state machine value. The RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.
RSTNMS	Reset Notification Bus Master. This 3-bit read-only field records the logical number of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. 000 Bus master is Core_0. 001 Bus master is Core_1. All other values are reserved.
RSTNTN	Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write. If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN; otherwise, reset all the notifications.
RSTNDP	Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires that RSTNDP = 0xb8.

47.4 Functional description

Multiprocessor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and so on. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate; that is, other processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

These three important rules must be followed for a correctly implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses by other processes/processors to the shared data or resources must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

Hennessy and Patterson's *Computer Architecture: A Quantitative Approach* offers this description of software gating:

> One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.
>
> The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can create locks with the proper semantics. The adjective *atomic* is key, for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory. (471–472)

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

> Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly “spin waiting” to see who can lock the variable first. All processes use a swap instruction that reads the old value and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the winner. (The losers will continue to set the variable to the locked value, but that doesn’t matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets. (472–473)

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with Power Architecture reservation instructions.

47.4.1 Semaphore usage

Example 1: Inter-processor communication done with software interrupts and semaphores.

- The e200z7_1 uses software interrupts to tell the e200z7_0 that new data is available, or the e200z7_0 does the same to tell the e200z7_1 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIUL to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM is not being updated, then the semaphore must first be locked, after which the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur:

    1. The e200z7_1 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the e200z7_0.

    2. Before the e200z7_0 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. The e200z7_0 software interrupt ISR reads the data sent to the e200z7_1, not the data sent from the e200z7_1, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores.

- The e200z7_0 wants to coherently read a section of shared memory.
- The e200z7_0 should check that the semaphore for the shared memory is not currently set.
- The e200z7_0 should set the semaphore for the shared memory to prevent the e200z7_1 from updating the shared memory.
- The e200z7_0 reads the required data, then unlocks the semaphore.

47.5 Initialization information

The reset state of the SEMA4 unit allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the unit to immediately begin operation.

47.6 Application information

In an operational multi-core system, most interactions involving the SEMA4 unit involve reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate:
  - The processor performs a byte write of logical_processor_number + 1 to gate[i].
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

    If the compare indicates the expected value

    then the gate is locked; proceed with the protected code segment

    else

    lock operation failed;

    repeat process beginning with byte write to gate[i] in spin-wait loop, or

    proceed with another execution path and wait for failed lock interrupt notification.

A simple C-language example of a gatelock function is shown in Example 47-1. This function follows the Hennessy and Patterson example.

Example 47-1. Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP2_LOCK    2

void gateLock (n)
int   n;                          /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;

  /* read the current value of the gate and wait until the state == UNLOCK */
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for this
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;   /* write gate with processor_number + 1 */
    current_value = gate[n];  /* read gate to verify ownership was obtained */
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate:
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate.

In this example, a reference to `processor_number()` retrieves this hardware configuration value. The logical processor numbers are defined by a hardwired input vector to the individual cores. For Power

Architecture cores, SPR 286 contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPR*n*.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt, because the request is automatically negated by the SEMA4 unit after the gate has been successfully locked by the failing processor.

Finally, in the event that a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset two-write sequence to avoid any race conditions involving spurious notification interrupt requests.

47.7 DMA requests

No DMA requests are associated with the SEMA4 unit.

This page is intentionally left blank.

Chapter 48
System Integration Unit Lite (SIUL)

48.1 Introduction

This chapter describes the SIUL, which is used for the management of the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads and is responsible for the management of the external interrupts to the device.

48.2 Overview

The SIUL controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals and external interrupts with trigger event configuration. Figure 48-1 shows a block diagram of the SIUL and its interfaces to other system components.

The module provides dedicated general-purpose pads that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the associated output pad. When configured as an input, the state of the associated pad can be detected by reading the value from an internal register. When configured as an input and output, the pad value can be read back, which can be used as a method of checking if the written value appeared on the pad.

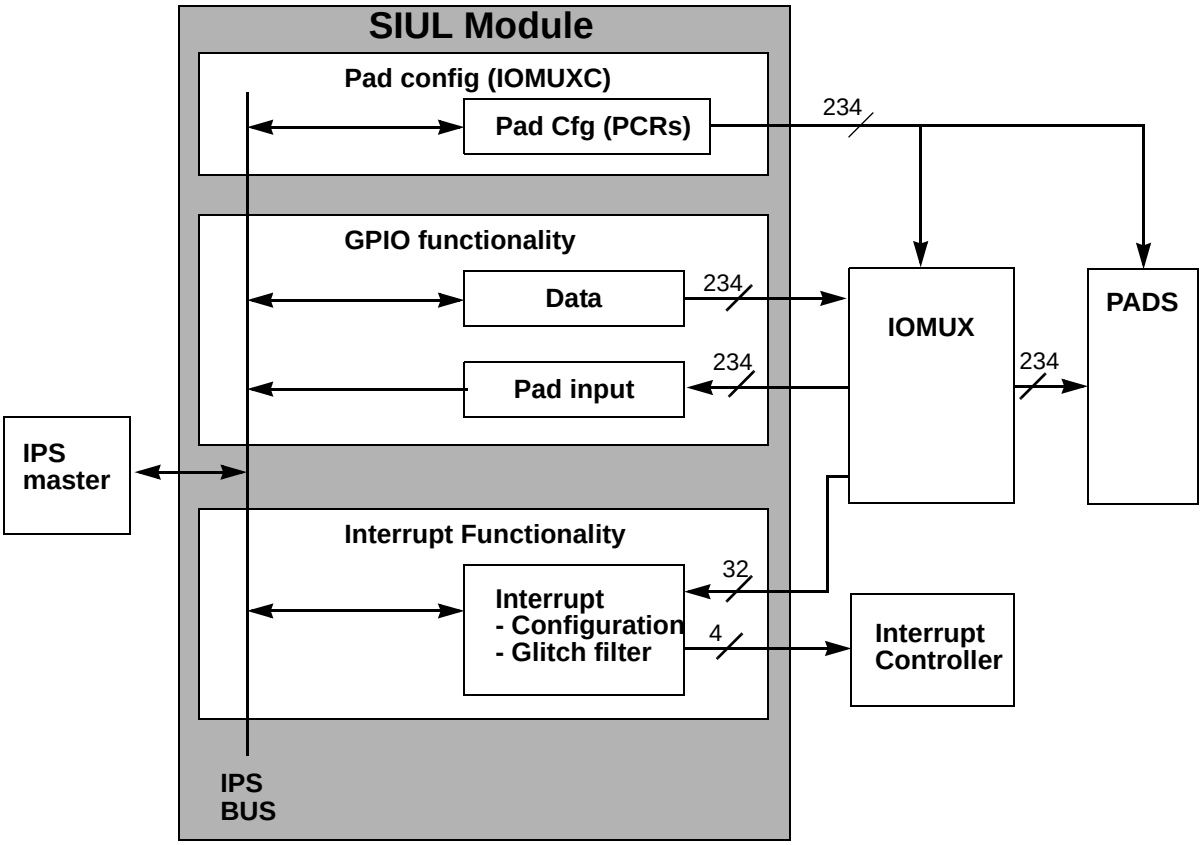


Figure 48-1. System Integration Unit Lite block diagram

48.3 Features

The SIUL supports these features:

- GPIO
  - GPIO function on 234 I/O pins
  - Dedicated input and output registers for each GPIO pin
- External interrupts
  - 4 system interrupt vectors for 32 interrupt sources
  - 32 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

48.3.1 Register protection

The individual registers of the SIUL are protected from accidental writes. See Chapter 45, Register Protection (REG_PROT).

48.4 External signal description

The pad configuration allows flexible, centralized control of the pin electrical characteristics of the MCU. The GPIO control provides centralized general purpose I/O for an MCU that multiplexes GPIO with other signals at the I/O pads. These other signals, or alternate functions, are normally the peripherals' functions. The internal multiplexing allows the selection of the input to chip-level signal multiplexers.

Table 48-1 lists the external pins used by the SIUL.

Table 48-1. SIUL signal properties

Name	I/O Type	Function
System Configuration		
GPIO[0:233][1]	I/O	General-Purpose I/O
External Interrupt		
EIRQ[0:8, 10:18, 22, 30:31, 35:38, 77:81, 96:97]	Input	External Interrupt Request Input

[1] GPIO [35], GPIO [40], GPIO [41], GPIO [61], GPIO [63], GPIO [65], GPIO [67], GPIO [72], GPIO [77], GPIO [79], GPIO [81], GPIO [82], GPIO [83], GPIO [96], GPIO [97], GPIO [104], GPIO [105], GPIO [106], and GPIO [107] are not implemented.

48.4.1 Detailed signal descriptions

48.4.1.1 General-purpose I/O pins (GPIO[0:233])

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDI*n_n*) or output (GPDO*n_n*) register. See Section 48.5.2.10, GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233), and Section 48.5.2.11, GPIO Pad Data Input Registers (GPDIn).

48.4.1.2 External interrupt request input pins (EIRQ[0:31])

The EIRQ[0:31] are connected to the SIUL inputs. Rising or falling edge events are enabled by setting the corresponding bits in the SIU_IREER or the SIU_IFEER register. See Section 48.5.2.5, Interrupt Rising-Edge Event Enable Register (IREER), and Section 48.5.2.6, Interrupt Falling-Edge Event Enable Register (IFEER).

48.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

48.5.1 SIUL memory map

Table 48-2 gives an overview on the SIUL registers implemented on the device.

Table 48-2. SIUL memory map

Offset from SIU_BASE (0xC3F9_0000)	Register	Access[1]	Reset Value[2]	Location
0x0000	Reserved			
0x0004	MCU ID Register #1 (MIDR1)	R	—[3]	on page 1621
0x0008	MCU ID Register #2 (MIDR2)	R	—[3]	on page 1621
0x000C–0x0013	Reserved			
0x0014	Interrupt Status Flag Register (ISR)	RW	0x0000_0000	on page 1622
0x0018	Interrupt Request Enable Register (IRER)	RW	0x0000_0000	on page 1623
0x001C–0x0027	Reserved			
0x0028	Interrupt Rising-Edge Event Enable (IREER)	RW	0x0000_0000	on page 1623
0x002C	Interrupt Falling-Edge Event Enable (IFEER)	RW	0x0000_0000	on page 1624
0x0030	Interrupt Filter Enable Register (IFER)	RW	0x0000_0000	on page 1625
0x0034–0x003F	Reserved			
0x0040–0x0210 See Table 48-11.	Pad Configuration Registers 0–234 (PCR0–PCR234[4])	RW	See Table 48-11.	on page 1625
0x0212–0x04FF	Reserved			

Table 48-2. SIUL memory map (continued)

Offset from SIU_BASE (0xC3F9_0000)	Register	Access[1]	Reset Value[2]	Location
0x0500– 0x053C See Table 48-13.	Pad Selection for Multiplexed Inputs (PSMI0–PSMI60)	RW	0x0000_0000	on page 1634
0x0540–0x05FF	Reserved			
0x0600–0x06E8 See Table 48-16.	GPIO Pad Data Output Register (GPDO0_3–GPDO232_233[4])	RW	0x0000_0000	on page 1640
0x06EC–0x07FF	Reserved			
0x0800–0x08E8 See Table 48-18.	GPIO Pad Data Input Register (GPDI0_3–GPDI232_233[4])	R	0x0000_0000	on page 1642
0x08EC–0x0BFF	Reserved			
0x0C00–0x0C1C See Table 48-20.	Parallel GPIO Pad Data Out Register (PGPDO0–PGPDO7)	RW	0x0000_0000	on page 1643
0x0C20–0x0C3F	Reserved			
0x0C40–0x0C5C See Table 48-22.	Parallel GPIO Pad Data In Register (PGPDI0–PGPDI7)	R	0x0000_0000	on page 1644
0x0C60–0x0C7F	Reserved			
0x0C80–0x0CBC See Table 48-24.	Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO15)	W	0x0000_0000	on page 1645
0x0CC0–0x0FFF	Reserved			
0x1000–0x107C See Table 48-26.	Interrupt Filter Maximum Counter Register (IFMC0–IFMC31)	RW	0x0000_0000	on page 1647
0x1080	Interrupt Filter Clock Prescaler Register (IFCP)	RW	0x0000_0000	on page 1648
0x1084–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 Device-specific. Please see register description.

4 GPIO [35], GPIO [40], GPIO [41], GPIO [61], GPIO [63], GPIO [65], GPIO [67], GPIO [72], GPIO [77], GPIO [79], GPIO [81], GPIO [82], GPIO [83], GPIO [96], GPIO [97], GPIO [104], GPIO [105], GPIO [106], and GPIO [107] are not implemented.

NOTE

A transfer error is issued when trying to access completely reserved register space.

48.5.2 Register description

This section describes the SIUL registers in address order.

48.5.2.1 MCU ID Register 1 (MIDR1)

This register contains information that identifies the device.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM[15:0]															
W																
Reset	0	1	0	1	0	1	1	0	0	1	1	1	0	x	x	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG					0	0	MAJOR_MASK				MINOR_MASK			
W																
Reset	0	x	x	x	x	x	0	0	0	0	0	0	0	0	0	0

Figure 48-2. MCU ID Register 1 (MIDR1)

Table 48-3. MIDR1 field descriptions

Field	Description
PARTNUM [15:0]	MCU Part Number Read-only, device part number of the MCU. 0101_0110_0111_0101 (5675): MPC5675K 0101_0110_0111_0011 (5673): MPC5673 0101_0110_0111_0001 (5671): MPC5671 For the full part number, this field needs to be combined with MIDR2[PARTNUM].
CSP	Always reads 0.
PKG	Package Settings Can by read by software to determine the package type that is used for the particular device. 257MapBGA = 01000 473MapBGA = 11101
MAJOR_MASK	Major Mask Revision. Read-only counter starting at 0x0. Incremented each time there is a resynthesis. (Cut 2.0) Major Mask = 0001
MINOR_MASK	Minor Mask Revision. Read-only counter starting at 0x0. Incremented each time a mask change is done. Minor Mask = 0000

48.5.2.2 MCU ID Register 2 (MIDR2)

This register contains information that identifies the device.

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SF	FLASH_SIZE_1[3:0]				FLASH_SIZE_2[3:0]			0	0	0	0	0	0	0	0
W																
Reset	0	0	x	x	x	0	x	x	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PARTNUM[23:16]								0	0	0	EE	0	0	0	FR
W																
Reset	0	1	0	0	1	0	1	1	0	0	0	x	0	0	0	x

Figure 48-3. MCU ID Register 2 (MIDR2)

Table 48-4. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0 Freescale 1 Reserved
FLASH_SIZE_1 [3:0]	Coarse granularity for flash memory size. Combine with FLASH_SIZE_2 to calculate the actual memory size. 0011 128 KB 0100 256 KB 0101 512 KB 0110 1 MB 0111 2 MB (MPC5675K) All other values are reserved.
FLASH_SIZE_2 [3:0]	Fine granularity for flash memory size. Combine with FLASH_SIZE_1 to calculate the actual memory size. 0000 0 × (FLASH_SIZE_1 ÷ 8) (MPC5675K) 0010 2 × (FLASH_SIZE_1 ÷ 8) 0100 4 × (FLASH_SIZE_1 ÷ 8) . . . 1110 14 × (FLASH_SIZE_1 ÷ 8) 1111 15 × (FLASH_SIZE_1 ÷ 8)
PARTNUM[23:16]	ASCII character in MCU part number. 0x4B: K family (MPC5675K)
EE	Data flash memory present. 0 No data flash memory is present. 1 Data flash memory is present. (MPC5675K)
FR	FlexRay present. 0 FlexRay is not present. 1 FlexRay is present. (MPC5675K)

48.5.2.3 Interrupt Status Flag Register (ISR)

The Interrupt Status Flag Register (ISR) holds the interrupt flags.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EIF[31:16]															
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF[15:0]															
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-4. Interrupt Status Flag Register (ISR)

Table 48-5. ISR field descriptions

Field	Description
EIF[*x*]	External Interrupt Status Flag *x* This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER[*x*]), EIF[*x*] causes an interrupt request. 0 No interrupt event has occurred on the pad. 1 An interrupt event as defined by IREER[*x*] and IFEER[*x*] has occurred.

48.5.2.4 Interrupt Request Enable Register (IRER)

The Interrupt Request Enable Register (IRER) enables the interrupt messaging to the interrupt controller.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EIRE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EIRE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-5. Interrupt Request Enable Register (IRER)

Table 48-6. IRER field descriptions

Field	Description
EIRE[*x*]	External Interrupt Request Enable *x* 0 Interrupt requests from the corresponding EIR[*x*] bit are disabled. 1 A set EIR[*x*] bit causes an interrupt request.

48.5.2.5 Interrupt Rising-Edge Event Enable Register (IREER)

The Interrupt Rising-Edge Event Enable Register (IREER) register enables rising-edge triggered events on the corresponding interrupt pads.

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IREE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IREE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-6. Interrupt Rising-Edge Event Enable Register (IREER)

Table 48-7. IREER field descriptions

Field	Description
IREE[x]	Enable rising-edge events to cause the EIF[x] bit to be set. 0 Rising-edge event is disabled. 1 Rising-edge event is enabled.

48.5.2.6 Interrupt Falling-Edge Event Enable Register (IFEER)

The Interrupt Falling-Edge Event Enable Register (IFEER) enables falling-edge triggered events on the corresponding interrupt pads.

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IFEE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFEE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-7. Interrupt Falling-Edge Event Enable Register (IFEER)

Table 48-8. IFEER field descriptions

Field	Description
IFEE[*x*]	Enable falling-edge events to cause the EIF[*x*] bit to be set. 0 Falling-edge event is disabled. 1 Falling-edge event is enabled.

NOTE

If both the IREE and IFEE bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt is never set.

48.5.2.7 Interrupt Filter Enable Register (IFER)

The Interrupt Filter Enable Register (IFER) register enables or disables a digital filter counter on the interrupt pads to filter out glitches on the inputs.

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IFE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-8. Interrupt Filter Enable Register (IFER)

Table 48-9. IFER field descriptions

Field	Description
IFE[x]	Enable digital glitch filter on the interrupt pad input. 0 Filter is disabled. 1 Filter is enabled.

48.5.2.8 Pad Configuration Registers (PCR*n*)

The Pad Configuration Registers (PCR*n*) allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR*n* controls the characteristics of a single pad.

Address: Base + 0x0040 (PCR0)
. . .
Base + 0x0214 (PCR234) See Table 48-11. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HYS	SMC	APC	0	PA		OBE	IBE	0	0	ODE	SRC			WPE	WPS
W																
Reset	—[1]	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 48-9. Pad Configuration Registers (PCR*n*)

[1] See Table 48-11.

NOTE

16- and 32-bit accesses are supported.

Table 48-10. PCR*n* field descriptions

Field	Description
HYS	Input Hysteresis. This bit controls whether the hysteresis output signal from the SIUL is asserted or negated. 0 Hysteresis enable signal is negated for the pad. 1 Hysteresis enable signal is asserted for the pad. **Note:** For PCRn where the HYS bitfield is listed as "X", the input hysteresis input for the pad is hardwired to '1' (enabled), and is not configurable. For PCRn where HYS is available, the HYS bit controls hysteresis as described above.
SMC	Safe Mode Control. This bit supports overriding automatic deactivation of the output buffer of the associated pad on entering Safe mode. 0 In Safe mode, the output buffer of the pad is disabled. 1 In Safe mode, the output buffer remains functional.
APC	Analog Pad Control. This bit enables the pad as analog input. 0 Analog input path from the pad is gated and cannot be used. 1 Analog input path switch can be enabled by the ADC.
PA[1:0]	Pad Output Assignment. This field selects the function that is allowed to drive the output of a multiplexed pad. The PA field size can vary from zero to two bits, depending on the number of output functions associated with this pad. 00 Alternative Mode 0: GPIO. 01 Alternative Mode 1. 10 Alternative Mode 2. 11 Alternative Mode 3.
OBE	Output Buffer Enable This bit enables the output buffer of the pad in case the pad is in GPIO mode. 0 Output Buffer of the pad is disabled when PA = 00. 1 Output Buffer of the pad is enabled when PA = 00.
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0 Input Buffer of the pad is disabled. 1 Input Buffer of the pad is enabled.
ODE	Open Drain Output Enable. This bit controls output driver configuration for the pads connected to this signal. Either open drain or push/pull driver configurations can be selected. This feature applies to output pads only. 0 Open drain enable signal is negated for the pad. 1 Open drain enable signal is asserted for the pad.
SRC[2:0]	Slew Rate Control 000 Slowest configuration. . . . 111 Fastest configuration. **Note:** For any given PCR*n* register, not all the SRC bits may be implemented. See Table 48-11 for more information. SRC bits shown as “x” are not implemented and should not be written.
WPE	Weak Pullup/pulldown Enable. This bit controls whether the weak pullup/pulldown devices are enabled/disabled for the pad connected to this signal. 0 Weak pull device enable signal is negated for the pad. 1 Weak pull device enable signal is asserted for the pad.

Table 48-10. PCR*n* field descriptions (continued)

Field	Description
WPS	Weak Pullup/pulldown Select. This bit controls whether weak pullup or weak pulldown devices are used for the pads connected to this signal when weak pullup/pulldown devices are enabled. 0 The pulldown is enabled. 1 The pullup is enabled.

Table 48-11. PCR*n* addresses

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x0040	PCR0	X0XX_X000_XX0X_X000	GPIO[0]	C6	D7
0x0042	PCR1	X0XX_X000_XX0X_X000	GPIO[1]	C7	C7
0x0044	PCR2	X0XX_X000_XX0X_X000	GPIO[2]	C8	C8
0x0046	PCR3	X0XX_X001_XX0X_X010	GPIO[3]	C9	C9
0x0048	PCR4	X0XX_X000_XX0X_X000	GPIO[4]	T15	AA20
0x004A	PCR5	X0XX_0000_XX0X_X000	GPIO[5]	H4	K2
0x004C	PCR6	X0XX_X000_XX0X_X000	GPIO[6]	K3	K3
0x004E	PCR7	X0XX_X000_XX0X_X000	GPIO[7]	K4	K4
0x0050	PCR8	X0XX_X000_XX0X_X000	GPIO[8]	F4	J4
0x0052	PCR9	X0XX_X000_XX0X_X000	GPIO[9]	B6	B6
0x0054	PCR10	X0XX_0000_XX0X_X000	GPIO[10]	J3	L3
0x0056	PCR11	X00X_X000_XX0X_X000	GPIO[11]	L3	U4
0x0058	PCR12	X00X_X000_XX0X_X000	GPIO[12]	T3	AB3
0x005A	PCR13	X0XX_X000_XX0X_X000	GPIO[13]	U3	AC3
0x005C	PCR14	X0XX_X000_XX0X_X000	GPIO[14]	B4	B4
0x005E	PCR15	X0XX_X000_XX0X_X000	GPIO[15]	D3	D3
0x0060	PCR16	X0XX_0000_XX0X_X000	GPIO[16]	B15	B21
0x0062	PCR17	X0XX_0000_XX0X_X000	GPIO[17]	C14	C20
0x0064	PCR18	X0XX_0000_XX0X_X000	GPIO[18]	R14	V20
0x0066	PCR19	X0XX_0000_XX0X_X000	GPIO[19]	W20	W20
0x0068	PCR20	X1XX_X110_XX0X_X100	GPIO[20]	L17	AB18
0x006A	PCR21	X1XX_X001_XX0X_X011	GPIO[21]	AA19	AA19
0x006C	PCR22	X0XX_0000_XX0X_X000	GPIO[22]	B3	B3
0x006E	PCR23	XX0X_X0X0_XXXX_XXXX	GPIO[23]	P7	AB10
0x0070	PCR24	XX0X_X0X0_XXXX_XXXX	GPIO[24]	T10	AC10
0x0072	PCR25	XX0X_X0X0_XXXX_XXXX	GPIO[25]	U11	Y14

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x0074	PCR26	XX0X_X0X0_XXXX_XXXX	GPIO[26]	T11	AA14
0x0076	PCR27	XX0X_X0X0_XXXX_XXXX	GPIO[27]	R11	AB14
0x0078	PCR28	XX0X_X0X0_XXXX_XXXX	GPIO[28]	P11	AC14
0x007A	PCR29	XX0X_X0X0_XXXX_XXXX	GPIO[29]	T12	AA15
0x007C	PCR30	XX0X_X0X0_XXXX_XXXX	GPIO[30]	R12	AB15
0x007E	PCR31	XX0X_X0X0_XXXX_XXXX	GPIO[31]	T13	AA16
0x0080	PCR32	XX0X_X0X0_XXXX_XXXX	GPIO[32]	—	AB16
0x0082	PCR33	XX0X_X0X0_XXXX_XXXX	GPIO[33]	R10	AA11
0x0084	PCR34	XX0X_X0X0_XXXX_XXXX	GPIO[34]	—	AC11
0x0086	reserved	—			
0x0088	PCR36	X0XX_0000_XX0X_X000	GPIO[36]	L1	L1
0x008A	PCR37	X0XX_0000_XX0X_X000	GPIO[37]	G3	K1
0x008C	PCR38	X0XX_0000_XX0X_X000	GPIO[38]	D4	D4
0x008E	PCR39	X0XX_0000_XX0X_X000	GPIO[39]	M3	M3
0x0090	reserved	—			
0x0092	reserved	—			
0x0094	PCR42	X0XX_0000_XX0X_X000	GPIO[42]	J4	L2
0x0096	PCR43	X0XX_X001_XX0X_X010	GPIO[43]	C6	C6
0x0098	PCR44	X0XX_X000_XX0X_X000	GPIO[44]	D6	D6
0x009A	PCR45	X0XX_X000_XX0X_X000	GPIO[45]	Y9	Y9
0x009C	PCR46	X0XX_0000_XX0X_X000	GPIO[46]	Y10	Y10
0x009E	PCR47	X0XX_X000_XX0X_X000	GPIO[47]	A8	A8
0x00A0	PCR48	X0XX_X000_XX0X_X000	GPIO[48]	B8	B8
0x00A2	PCR49	X0XX_0000_XX0X_X000	GPIO[49]	E3	E3
0x00A4	PCR50	X0XX_0000_XX0X_X000	GPIO[50]	C5	C5
0x00A6	PCR51	X0XX_X000_XX0X_X000	GPIO[51]	A7	A7
0x00A8	PCR52	X0XX_X000_XX0X_X000	GPIO[52]	B7	B7
0x00AA	PCR53	X0XX_0000_XX0X_X000	GPIO[53]	N3	W3
0x00AC	PCR54	X0XX_0000_XX0X_X000	GPIO[54]	P3	Y3
0x00AE	PCR55	X0XX_0000_XX0X_X000	GPIO[55]	R4	AA4
0x00B0	PCR56	X0XX_0000_XX0X_X000	GPIO[56]	V4	V4
0x00B2	PCR57	X0XX_0000_XX0X_X000	GPIO[57]	M1	M1

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x00B4	PCR58	X0XX_X000_XX0X_X000	GPIO[58]	N1	N1
0x00B6	PCR59	X0XX_X000_XX0X_X000	GPIO[59]	P1	P1
0x00B8	PCR60	X0XX_X000_XX0X_X000	GPIO[60]	N3	N3
0x00BA	reserved	—			
0x00BC	PCR62	X0XX_X000_XX0X_X000	GPIO[62]	—	P2
0x00BE	reserved	—			
0x00C0	PCR64	XX0X_X0X0_XXXX_XXXX	GPIO[64]	—	AA17
0x00C2	reserved	—			
0x00C4	PCR66	XX0X_X0X0_XXXX_XXXX	GPIO[66]	—	AA12
0x00C6	reserved	—			
0x00C8	PCR68	XX0X_X0X0_XXXX_XXXX	GPIO[68]	—	AB13
0x00CA	PCR69	XX0X_X0X0_XXXX_XXXX	GPIO[69]	—	AA13
0x00CC	PCR70	XX0X_X0X0_XXXX_XXXX	GPIO[70]	—	AB11
0x00CE	PCR71	XX0X_X0X0_XXXX_XXXX	GPIO[71]	—	AB12
0x00D0	reserved	—			
0x00D2	PCR73	XX0X_X0X0_XXXX_XXXX	GPIO[73]	—	AA18
0x00D4	PCR74	XX0X_X0X0_XXXX_XXXX	GPIO[74]	—	Y17
0x00D6	PCR75	XX0X_X0X0_XXXX_XXXX	GPIO[75]	—	AB17
0x00D8	PCR76	XX0X_X0X0_XXXX_XXXX	GPIO[76]	—	Y18
0x00DA	reserved	—			
0x00DC	PCR78	X0XX_X000_XX0X_X000	GPIO[78]	P13	Y15
0x00DE	reserved	—			
0x00E0	PCR80	X0XX_X000_XX0X_X000	GPIO[80]	—	R3
0x00E2	reserved	—			
0x00E4	reserved	—			
0x00E6	reserved	—			
0x00E8	PCR84	X0XX_0000_XX0X_X000	GPIO[84]	D2	D2
0x00EA	PCR85	X0XX_0000_XX0X_X000	GPIO[85]	D1	E2
0x00EC	PCR86	X0XX_0000_XX0X_X000	GPIO[86]	E2	D1
0x00EE	PCR87	X0XX_0000_XX0X_X000	GPIO[87]	J1	G1
0x00F0	PCR88	X0XX_0000_XX0X_X000	GPIO[88]	K2	G4
0x00F2	PCR89	X0XX_0000_XX0X_X000	GPIO[89]	K1	H3

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x00F4	PCR90	X0XX_0000_XX0X_X000	GPIO[90]	L1	H1
0x00F6	PCR91	X0XX_0000_XX0X_X000	GPIO[91]	L2	H4
0x00F8	PCR92	X0XX_X001_XX0X_X010	GPIO[92]	P8	Y11
0x00FA	PCR93	X0XX_0000_XX0X_X000	GPIO[93]	P12	Y16
0x00FC	PCR94	X0XX_0000_XX0X_X000	GPIO[94]	—	AA22
0x00FE	PCR95	X0XX_0000_XX0X_X000	GPIO[95]	—	AB21
0x0100	reserved	—			
0x0102	reserved	—			
0x0104	PCR98	X0XX_0000_XX0X_X000	GPIO[98]	—	R1
0x0106	PCR99	X0XX_X000_XX0X_X000	GPIO[99]	—	P3
0x0108	PCR100	X0XX_X000_XX0X_X000	GPIO[100]	—	N4
0x010A	PCR101	X0XX_X000_XX0X_X000	GPIO[101]	—	R2
0x010C	PCR102	X0XX_X000_XX0X_X000	GPIO[102]	—	P4
0x010E	PCR103	X0XX_X000_XX0X_X000	GPIO[103]	—	T1
0x0110	reserved	—			
0X0112	reserved	—			
0x0114	reserved	—			
0x0116	reserved	—			
0x0118	PCR108	X0XX_0000_XX0X_X000	GPIO[108]	F2	F2
0x011A	PCR109	X0XX_0000_XX0X_X000	GPIO[109]	H1	F1
0x011C	PCR110	X0XX_0000_XX0X_X000	GPIO[110]	A6	A6
0x011E	PCR111	X0XX_0000_XX0X_X000	GPIO[111]	J3	G3
0x0120	PCR112	X0XX_0000_XX0X_X000	GPIO[112]	A5	A5
0x0122	PCR113	X0XX_0000_XX0X_X000	GPIO[113]	F1	F3
0x0124	PCR114	X0XX_0000_XX0X_X000	GPIO[114]	A4	A4
0x0126	PCR115	X0XX_0000_XX0X_X000	GPIO[115]	G1	F4
0x0128	PCR116	X0XX_0000_XX0X_X000	GPIO[116]	—	Y5
0x012A	PCR117	X0XX_0000_XX0X_X000	GPIO[117]	—	T2
0x012C	PCR118	X0XX_0000_XX0X_X000	GPIO[118]	—	U1
0x012E	PCR119	X0XX_X000_XX0X_X000	GPIO[119]	—	AA5
0x0130	PCR120	X0XX_0000_XX0X_X000	GPIO[120]	—	T3
0x0132	PCR121	X0XX_0000_XX0X_X000	GPIO[121]	—	U2

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x0134	PCR122	X0XX_X000_XX0X_X000	GPIO[122]	—	AB4
0x0136	PCR123	X0XX_0000_XX0X_X000	GPIO[123]	—	U3
0x0138	PCR124	X0XX_X000_XX0X_X000	GPIO[124]	—	V3
0x013A	PCR125	X0XX_X000_XX0X_X000	GPIO[125]	—	AB5
0x013C	PCR126	X0XX_0000_XX0X_X000	GPIO[126]	—	AC4
0x013E	PCR127	X0XX_0000_XX0X_X000	GPIO[127]	—	AC5
0x0140	PCR128	X0XX_0000_XX0X_X000	GPIO[128]	C17	A17
0x0142	PCR129	00XX_0000_XX0X_0000	GPIO[129]	E14	B18
0x0144	PCR130	00XX_X000_XX0X_0000	GPIO[130]	G17	D19
0x0146	PCR131	00XX_0000_XX0X_0000	GPIO[131]	D16	B17
0x0148	PCR132	00XX_X000_XX0X_0000	GPIO[132]	D17	A16
0x014A	PCR133	00XX_0000_XX0X_0000	GPIO[133]	E15	C17
0x014C	PCR134	00XX_X000_XX0X_0000	GPIO[134]	E16	A15
0x014E	PCR135	00XX_0000_XX0X_0000	GPIO[135]	E17	B16
0x0150	PCR136	00XX_0000_XX0X_0000	GPIO[136]	C16	C16
0x0152	PCR137	00XX_0000_XX0X_0000	GPIO[137]	F15	B15
0x0154	PCR138	00XX_0000_XX0X_0000	GPIO[138]	F16	A18
0x0156	PCR139	00XX_X000_XX0X_0000	GPIO[139]	F17	C18
0x0158	PCR140	00XX_X000_XX0X_0000	GPIO[140]	G14	B19
0x015A	PCR141	00XX_X000_XX0X_0000	GPIO[141]	G15	A19
0x015C	PCR142	00XX_X000_XX0X_0000	GPIO[142]	G16	D18
0x015E	PCR143	00XX_X000_XX0X_0000	GPIO[143]	H14	C19
0x0160	PCR144	00XX_0000_XX0X_0000	GPIO[144]	H15	A20
0x0162	PCR145	00XX_0000_XX0X_0000	GPIO[145]	J14	B20
0x0164	PCR146	00XX_0000_XX0X_0000	GPIO[146]	J15	A21
0x0166	PCR147	X0XX_0000_XX00_00XX	GPIO[147]	P16	R22
0x0168	PCR148	X0XX_0000_XX00_00XX	GPIO[148]	K17	F22
0x016A	PCR149	X0XX_0000_XX00_00XX	GPIO[149]	K16	E21
0x016C	PCR150	X0XX_0000_XX00_00XX	GPIO[150]	L16	E22
0x016E	PCR151	X0XX_0000_XX00_00XX	GPIO[151]	N15	F20
0x0170	PCR152	X0XX_0000_XX00_00XX	GPIO[152]	M14	C23
0x0172	PCR153	X0XX_0000_XX00_00XX	GPIO[153]	P15	M21

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x0174	PCR154	X0XX_0000_XX00_00XX	GPIO[154]	N14	D23
0x0176	PCR155	X0XX_0000_XX00_00XX	GPIO[155]	N16	D21
0x0178	PCR156	X0XX_0000_XX00_00XX	GPIO[156]	N17	E23
0x017A	PCR157	X0XX_0000_XX00_00XX	GPIO[157]	M17	M20
0x017C	PCR158	X0XX_0000_XX00_00XX	GPIO[158]	—	U23
0x017E	PCR159	X0XX_0000_XX00_00XX	GPIO[159]	—	T22
0x0180	PCR160	X0XX_0000_XX00_00XX	GPIO[160]	—	V23
0x0182	PCR161	X0XX_0000_XX00_00XX	GPIO[161]	—	R21
0x0184	PCR162	X0XX_0000_XX00_00XX	GPIO[162]	—	W23
0x0186	PCR163	X0XX_0000_XX00_00XX	GPIO[163]	P17	Y23
0x0188	PCR164	X0XX_0000_XX00_00XX	GPIO[164]	R16	U20
0x018A	PCR165	X0XX_0000_XX00_00XX	GPIO[165]	R17	W22
0x018C	PCR166	X0XX_0000_XX00_00XX	GPIO[166]	—	T20
0x018E	PCR167	X0XX_0000_XX00_00XX	GPIO[167]	—	T21
0x0190	PCR168	X0XX_0000_XX00_00XX	GPIO[168]	—	AA23
0x0192	PCR169	X0XX_0000_XX00_00XX	GPIO[169]	—	Y22
0x0194	PCR170	X0XX_0000_XX00_00XX	GPIO[170]	—	U21
0x0196	PCR171	X0XX_0000_XX00_00XX	GPIO[171]	—	V21
0x0198	PCR172	X0XX_0000_XX00_00XX	GPIO[172]	—	W21
0x019A	PCR173	X0XX_0000_XX00_00XX	GPIO[173]	—	Y21
0x019C	PCR174	X0XX_0000_XX00_00XX	GPIO[174]	—	J20
0x019E	PCR175	X0XX_0000_XX00_00XX	GPIO[175]	—	J21
0x01A0	PCR176	X0XX_0000_XX00_00XX	GPIO[176]	—	H20
0x01A2	PCR177	X0XX_0000_XX00_00XX	GPIO[177]	—	J22
0x01A4	PCR178	X0XX_0000_XX00_00XX	GPIO[178]	—	K21
0x01A6	PCR179	X0XX_0000_XX00_00XX	GPIO[179]	—	F23
0x01A8	PCR180	X0XX_0000_XX00_00XX	GPIO[180]	—	J23
0x01AA	PCR181	X0XX_0000_XX00_00XX	GPIO[181]	—	G23
0x01AC	PCR182	X0XX_0000_XX00_00XX	GPIO[182]	—	K22
0x01AE	PCR183	X0XX_0000_XX00_00XX	GPIO[183]	—	K23
0x01B0	PCR184	X0XX_0000_XX00_00XX	GPIO[184]	—	M23
0x01B2	PCR185	X0XX_0000_XX00_00XX	GPIO[185]	—	M22

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x01B4	PCR186	X0XX_0000_XX00_00XX	GPIO[186]	—	N23
0x01B6	PCR187	X0XX_0000_XX00_00XX	GPIO[187]	—	N22
0x01B8	PCR188	X0XX_0000_XX00_00XX	GPIO[188]	—	P20
0x01BA	PCR189	X0XX_0000_XX00_00XX	GPIO[189]	—	P21
0x01BC	PCR190	X0XX_0000_XX00_00XX	GPIO[190]	—	G21
0x01BE	PCR191	X0XX_0000_XX00_00XX	GPIO[191]	—	N20
0x01C0	PCR192	X0XX_0000_XX00_00XX	GPIO[192]	—	G22
0x01C2	PCR193	X0XX_0000_XX00_00XX	GPIO[193]	—	N21
0x01C4	PCR194	X0XX_0000_XX00_00XX	GPIO[194]	H17	F21
0x01C6	PCR195	X0XX_0000_XX00_00XX	GPIO[195]	J17	D22
0x01C8	PCR196	X0XX_0000_XX00_00XX	GPIO[196]	K14	G20
0x01CA	PCR197	X0XX_0000_XX00_00XX	GPIO[197]	K15	C22
0x01CC	PCR198	X0XX_0000_XX0X_X000	GPIO[198]	A13	A10
0x01CE	PCR199	X0XX_X000_XX0X_X000	GPIO[199]	D13	D15
0x01D0	PCR200	X0XX_0000_XX0X_X000	GPIO[200]	A14	A12
0x01D2	PCR201	X0XX_0000_XX0X_X000	GPIO[201]	C12	B11
0x01D4	PCR202	X0XX_0000_XX0X_X000	GPIO[202]	D11	C11
0x01D6	PCR203	X0XX_X000_XX0X_X000	GPIO[203]	D10	C10
0x01D8	PCR204	X0XX_0000_XX0X_X000	GPIO[204]	A15	A13
0x01DA	PCR205	X0XX_0000_XX0X_X000	GPIO[205]	B13	B13
0x01DC	PCR206	X0XX_0000_XX0X_X000	GPIO[206]	C13	C15
0x01DE	PCR207	X0XX_0000_XX0X_X000	GPIO[207]	B14	A11
0x01E0	PCR208	X0XX_0000_XX0X_X000	GPIO[208]	C11	C12
0x01E2	PCR209	X0XX_0000_XX0X_X000	GPIO[209]	A11	C13
0x01E4	PCR210	X0XX_0000_XX0X_X000	GPIO[210]	D12	A9
0x01E6	PCR211	X0XX_X000_XX0X_X000	GPIO[211]	A12	B12
0x01E8	PCR212	X0XX_0000_XX0X_X000	GPIO[212]	B12	C14
0x01EA	PCR213	X0XX_0000_XX0X_X000	GPIO[213]	A10	D14
0x01EC	PCR214	X0XX_X000_XX0X_X000	GPIO[214]	B10	B9
0x01EE	PCR215	X0XX_0000_XX0X_X000	GPIO[215]	B11	B10
0x01F0	PCR216	X0XX_0000_XX0X_X000	GPIO[216]	K3	J1
0x01F2	PCR217	X0XX_0000_XX0X_X000	GPIO[217]	M4	J3

Table 48-11. PCR*n* addresses (continued)

Address	Register	Reset value (binary)[1]	GPIO	Pin (257 pkg)	Pin (473 pkg)
0x01F4	PCR218	X0XX_0000_XX0X_X000	GPIO[218]	L4	J2
0x01F6	PCR219	X0XX_0000_XX0X_X000	GPIO[219]	B5	B5
0x01F8	PCR220	X0XX_0000_XX0X_X000	GPIO[220]	C2	C2
0x01FA	PCR221	XX0X_X0X0_XXXX_XXXX	GPIO[221]	R5	AA8
0x01FC	PCR222	XX0X_X0X0_XXXX_XXXX	GPIO[222]	U6	AB8
0x01FE	PCR223	XX0X_X0X0_XXXX_XXXX	GPIO[223]	T6	AB9
0x0200	PCR224	XX0X_X0X0_XXXX_XXXX	GPIO[224]	R6	AC9
0x0202	PCR225	XX0X_X0X0_XXXX_XXXX	GPIO[225]	U7	Y7
0x0204	PCR226	XX0X_X0X0_XXXX_XXXX	GPIO[226]	U8	AA7
0x0206	PCR227	XX0X_X0X0_XXXX_XXXX	GPIO[227]	T8	AB7
0x0208	PCR228	XX0X_X0X0_XXXX_XXXX	GPIO[228]	R8	Y8
0x020A	PCR229	XX0X_X0X0_XXXX_XXXX	GPIO[229]	T4	Y6
0x020C	PCR230	XX0X_X0X0_XXXX_XXXX	GPIO[230]	U4	AA6
0x020E	PCR231	XX0X_X0X0_XXXX_XXXX	GPIO[231]	U5	AB6
0x0210	PCR232	XX0X_X0X0_XXXX_XXXX	GPIO[232]	T5	AC6
0x0212	PCR233	00XX_0000_XX0X_0000	GPIO[233]	F14	E20
0x0214	PCR234	X0XX_XXXX_XXX0_00XX	GPIO[234]	—	R23/T23

1 For a bit value of x, the bit is not implemented for that PCR*n* register and its reset value is 0. Do not attempt to write to this reserved bit.

48.5.2.9 Pad Selection for Multiplexed Inputs (PSMI0_3–PSMI60_63)

The Pad Selection for Multiplexed Inputs (PSMI0_3–PSMI60_63) registers define pads as inputs to peripheral functions. Table 48-13 lists the addresses for the PSMI0_3–PSMI60_63 registers, and Table 48-14 shows how the pad selection bits (PADSEL) are allocated to the pads.

Address: Base + 0x0500–0x053C. See Table 48-13 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PADSEL0				0	0	0	0	PADSEL1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PADSEL2				0	0	0	0	PADSEL3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-10. Pad Selection for Multiplexed Inputs Register (PSMI0_3–PSMI60_63)

Table 48-12. PSMI0_3–PSMI60_63 field descriptions

Field	Description
PADSEL*n*	Pad Selection Each PADSEL field selects the pad currently used for a certain input function. See Table 48-14.

Table 48-13. PSMI*n* addresses

Register	Address	Register	Address	Register	Address
PSMI0_3	0x0500	PSMI24_27	0x0518	PSMI48_51	0x0530
PSMI4_7	0x0504	PSMI_28_31	0x051C	PSMI52_55	0x0534
PSMI8_11	0x0508	PSMI32_35	0x0520	PSMI56_59	0x0538
PSMI12_15	0x050C	PSMI36_39	0x0524	PSMI60_63	0x053C
PSMI16_19	0x0510	PSMI40_43	0x0528		
PSMI20_23	0x0514	PSMI44_47	0x052C		

Table 48-14. PADSEL allocations

Register	PADSEL*n*	Module	Signal	GPIO[1]	Binary Value[2]	Pin (257 pkg)	Pin (473 pkg)
PSMI0_3	PADSEL0	CTU0	EXT_IN	GPIO[45] GPIO[47] GPIO[116]	0000 0001 0010	P5 A8 —	Y9 A8 Y5
	PADSEL1	DSPI2	SCK	GPIO[11] GPIO[202]	0000 0001	L3 D11	U4 C11
	PADSEL2	DSPI2	SIN	GPIO[0] GPIO[13]	0000 0001	C6 U3	D7 AC3
	PADSEL3	DSPI2	CS0	GPIO[10] GPIO[198]	0000 0001	J3 A13	L3 A10

Table 48-14. PADSEL allocations (continued)

Register	PADSEL*n*	Module	Signal	GPIO[1]	Binary Value[2]	Pin (257 pkg)	Pin (473 pkg)
PSMI4_7	PADSEL4[3]	—	—	—	—	—	—
	PADSEL5[3]	—	—	—	—	—	—
	PADSEL6[3]	—	—	—	—	—	—
	PADSEL7	eTimer0	ETC[4]	GPIO[43] GPIO[30] GPIO[99]	0000 0001 0010	D7 R12 —	C6 AB15 P3
PSMI8_11	PADSEL8	eTimer0	ETC[5]	GPIO[44] GPIO[24] GPIO[100]	0000 0001 0010	D6 T10 —	D6 AC10 N4
	PADSEL9	eTimer1	ETC[0]	GPIO[4] GPIO[136]	0000 0001	T15 C16	AA20 C16
	PADSEL10	eTimer1	ETC[1]	GPIO[45] GPIO[137]	0000 0001	P5 F15	Y9 B15
	PADSEL11	eTimer1	ETC[2]	GPIO[46] GPIO[133]	0000 0001	P6 E15	Y10 C17
PSMI12_15	PADSEL12	eTimer1	ETC[3]	GPIO[92] GPIO[128]	0000 0001	P8 C17	Y11 A17
	PADSEL13	eTimer1	ETC[4]	GPIO[93] GPIO[135]	0000 0001	P12 E17	Y16 B16
	PADSEL14	eTimer1	ETC[5]	GPIO[78] GPIO[138]	0000 0001	P13 F16	Y15 A18
	PADSEL15	FlexPWM0	EXT_SYNC	GPIO[45] GPIO[47]	0000 0001	P5 A8	Y9 A8
PSMI16_19	PADSEL16	FlexPWM0	FAULT[0]	GPIO[9] GPIO[13]	0000 0001	B6 U3	B6 AC3
	PADSEL17	FlexPWM0	FAULT[1]	GPIO[42] GPIO[54]	0000 0001	J4 P3	L2 Y3
	PADSEL18[3]	—	—	—	—	—	—
	PADSEL19	FlexPWM0	FAULT[3]	GPIO[37] GPIO[56]	0000 0001	G3 M3	K1 V4
PSMI20_23	PADSEL20	FlexPWM0	A[0]	GPIO[58] GPIO[147]	0000 0001	— P16	N1 R22
	PADSEL21	FlexPWM0	A[1]	GPIO[80] GPIO[149]	0000 0001	— K16	R3 E21
	PADSEL22	FlexPWM0	A[2]	GPIO[99] GPIO[151]	0000 0001	— N15	P3 F20
	PADSEL23	FlexPWM0	A[3]	GPIO[102] GPIO[153]	0000 0001	— P15	P4 M21

Table 48-14. PADSEL allocations (continued)

Register	PADSEL*n*	Module	Signal	GPIO[1]	Binary Value[2]	Pin (257 pkg)	Pin (473 pkg)
PSMI24_27	PADSEL24	FlexPWM0	B[0]	GPIO[59] GPIO[148]	0000 0001	— K17	P1 F22
	PADSEL25	FlexPWM0	B[1]	GPIO[62] GPIO[150]	0000 00001	— L16	P2 E22
	PADSEL26	FlexPWM0	B[2]	GPIO100] GPIO[152]	0000 0001	— M14	N4 C23
	PADSEL27	FlexPWM0	B[3]	GPIO[103] GPIO[154]	0000 0001	— N14	T1 D23
PSMI_28_31	PADSEL28	FlexPWM0	X[1]	GPIO[60] GPIO[195]	0000 0001	— J17	N3 D22
	PADSEL29	FlexPWM0	X[2]	GPIO[98] GPIO[196]	0000 0001	— K14	R1 G20
	PADSEL30	FlexPWM0	X[3]	GPIO[101] GPIO[197]	0000 0001	— K15	R2 C22
	PADSEL31	LINFlex0	RxD	GPIO[19] GPIO[23] GPIO[205]	0000 0001 0010	T14 P7 B13	W20 AB10 B13
PSMI32_35	PADSEL32	LINFlex1	RxD	GPIO[29] GPIO[95] GPIO[199]	0000 0001 0010	T12 — D13	AA15 AB21 D15
	PADSEL33	FlexCAN0	RxD RxD	GPIO[15] GPIO[17]	0000 0001	D3 C14	D3 C20
	PADSEL34	FlexCAN1	RxD RxD	GPIO[15] GPIO[17]	0000 0001	D3 C14	D3 C20
	PADSEL35	eTimer0	ETC[0]	GPIO[0] GPIO[58]	0000 0001	C6 —	D7 N1
PSMI36_39	PADSEL36	eTimer0	ETC[1]	GPIO[1] GPIO[59]	0000 0001	C7 —	C7 P1
	PADSEL37	eTimer0	ETC[2]	GPIO[2] GPIO[80]	0000 0001	C8 —	C8 R3
	PADSEL38	eTimer0	ETC[3]	GPIO[3] GPIO[62]	0000 0001	C9 —	C9 P2
	PADSEL39	eTimer2	ETC[0]	GPIO[116] GPIO[210]	0000 0001	— D12	Y5 A9

Table 48-14. PADSEL allocations (continued)

Register	PADSEL*n*	Module	Signal	GPIO[1]	Binary Value[2]	Pin (257 pkg)	Pin (473 pkg)
PSMI40_43	PADSEL40	eTimer2	ETC[1]	GPIO[119] GPIO[201]	0000 0001	— C12	AA5 B11
	PADSEL41	eTimer2	ETC[2]	GPIO[122] GPIO[209]	0000 0001	— A11	AB4 C13
	PADSEL42	eTimer2	ETC[3]	GPIO[125] GPIO[208]	0000 0001	— C11	AB5 C12
	PADSEL43	FlexCAN2	RxD	GPIO[9] GPIO[118] GPIO[121] GPIO[218] GPIO[220]	0000 0001 0010 0011 0100	B6 — — L4 C2	B6 U1 U2 J2 C2
PSMI44_47	PADSEL44	FlexCAN3	RxD	GPIO[11] GPIO[118] GPIO[121] GPIO[218] GPIO[220]	0000 0001 0010 0011 0100	L3 — — L4 C2	U4 U1 U2 J2 C2
	PADSEL45	I2C1	SCL	GPIO[94] GPIO[211]	0000 0001	— A12	AA22 B12
	PADSEL46	I2C1	SDA	GPIO[95] GPIO[214]	0000 0001	— B10	AB21 B9
	PADSEL47	I2C2	SCL	GPIO[53] GPIO[145]	0000 0001	N3 J14	W3 B20
PSMI48_51	PADSEL48	I2C2	SDA	GPIO[54] GPIO[146]	0000 0001	P3 J15	Y3 A21
	PADSEL49	LINFlex2	RxD	GPIO[56] GPIO[60] GPIO[130]	0000 0001 0010	M3 — G17	V4 N3 D19
	PADSEL50	LINFlex3	RxD	GPIO[9] GPIO[101] GPIO[143]	0000 0001 0010	B6 — H14	B6 R2 C19
	PADSEL51	eTimer2	ETC[4]	GPIO[126] GPIO[207]	0000 0001	— B14	AC4 A11
PSMI52_55	PADSEL52	eTimer2	ETC[5]	GPIO[22] GPIO[127] GPIO[212] GPIO[233]	0000 0001 0010 0011	B3 — B12 F14	B3 AC5 C14 E20
	PADSEL53	FlexPWM1	A[0]	GPIO[117] GPIO[155]	0000 0001	— N16	T2 D21
	PADSEL54	FlexPWM1	A[1]	GPIO[120] GPIO[157]	0000 0001	— M17	T3 M20
	PADSEL55	FlexPWM1	A[2]	GPIO[123] GPIO[164]	0000 0001	— R16	U3 U20

Table 48-14. PADSEL allocations (continued)

Register	PADSEL*n*	Module	Signal	GPIO[1]	Binary Value[2]	Pin (257 pkg)	Pin (473 pkg)
PSMI56_59	PADSEL56	FlexPWM1	B[0]	GPIO[118] GPIO[156]	0000 0001	— N17	U1 E23
	PADSEL57	FlexPWM1	B[1]	GPIO[121] GPIO[163]	0000 0001	— P17	U2 Y23
	PADSEL58	FlexPWM1	B[2]	GPIO[124] GPIO[165]	0000 0001	— R17	V3 W22
	PADSEL59	CTU1	EXT_IN	GPIO[45] GPIO[48] GPIO[92] GPIO[116] GPIO[146]	0000 0001 0010 0011 0100	P5 B8 P8 J15	Y9 B8 Y11 Y5 A21
PSMI60_63	PADSEL60[4]	FlexPWM0	X[0]	GPIO[57] GPIO[194]	0000 0001	— H17	M1 F21

1 GPIO number corresponds to register PCRn value, i.e., GPIO 45 corresponds to PCR45.

2 Values not listed are reserved.

3 Field is reserved.

4 PADSEL[61:63] are not implemented.

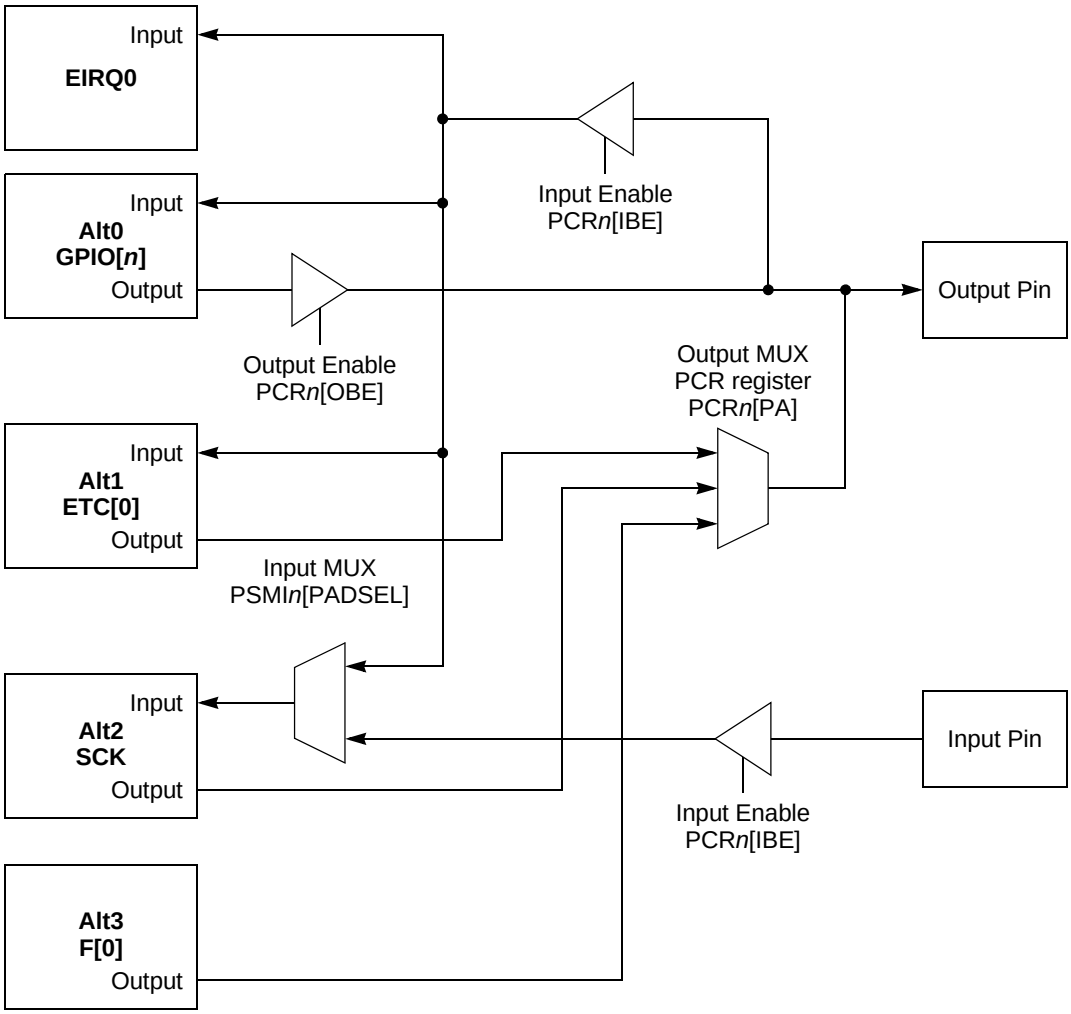


Figure 48-11. PADSEL muxing

48.5.2.10 GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233)

The GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233) can be used to set or clear a single GPIO pad with a byte access.

Address: Base + 0x0600–0x6E8. See Table 48-16. Access: User read/write

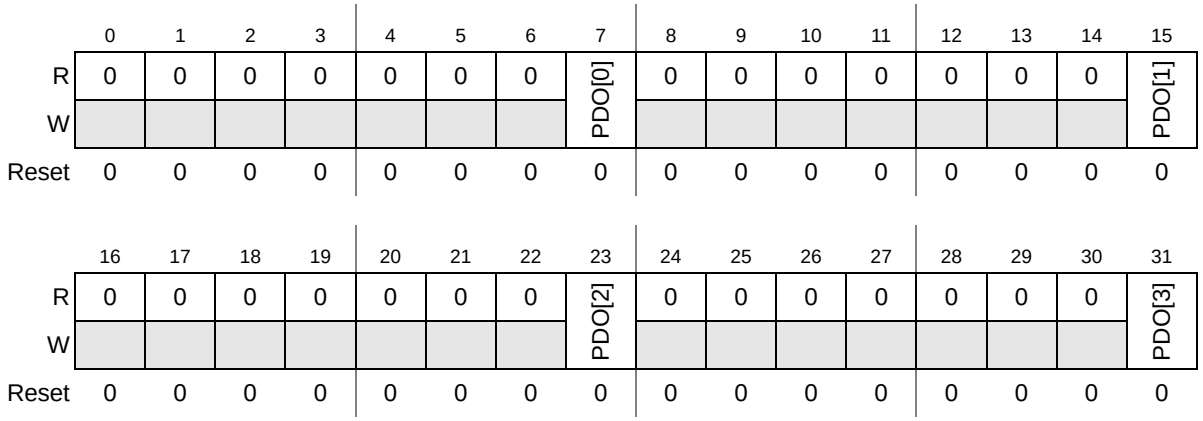


Figure 48-12. Port GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233)

Table 48-15. GPDO0_3–GPDO232_233 field descriptions

Field	Description
PDO[x]	Pad Data Out. This bit stores the data to be driven out on the external GPIO pad controlled by this register. 0 Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output. 1 Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output.

Table 48-16. GPDO*n* addresses

Register[1]	Address	Register	Address	Register	Address
GPDO0_3	0x0600	GPDO80_83	0x0650	GPDO160_163	0x06A0
GPDO4_7	0x0604	GPDO84_87	0x0654	GPDO164_167	0x06A4
GPDO8_11	0x0608	GPDO88_91	0x0658	GPDO168_171	0x06A8
GPDO12_15	0x060C	GPDO92_95	0x065C	GPDO172_175	0x06AC
GPDO16_19	0x0610	GPDO96_99	0x0660	GPDO176_179	0x06B0
GPDO20_23	0x0614	GPDO100_103	0x0664	GPDO180_183	0x06B4
GPDO24_27	0x0618	reserved	0x0668	GPDO184_187	0x06B8
GPDO_28_31	0x061C	GPDO108_111	0x066C	GPDO188_191	0x06BC
GPDO32_35	0x0620	GPDO112_115	0x0670	GPDO192_195	0x06C0
GPDO36_39	0x0624	GPDO116_119	0x0674	GPDO196_199	0x06C4
GPDO40_43	0x0628	GPDO120_123	0x0678	GPDO200_203	0x06C8
GPDO44_47	0x062C	GPDO124_127	0x067C	GPDO204_207	0x06CC
GPDO48_51	0x0630	GPDO128_131	0x0680	GPDO208_211	0x06D0
GPDO52_55	0x0634	GPDO132_135	0x0684	GPDO212_215	0x06D4

Table 48-16. GPDO*n* addresses (continued)

Register[1]	Address	Register	Address	Register	Address
GPDO56_59	0x0638	GPDO136_139	0x0688	GPDO216_219	0x06D8
GPDO60_63	0x063C	GPDO140_143	0x068C	GPDO220_223	0x06DC
GPDO64_67	0x0640	GPDO144_147	0x0690	GPDO224_227	0x06E0
GPDO68_71	0x0644	GPDO148_151	0x0694	GPDO228_231	0x06E4
GPDO72_75	0x0648	GPDO152_155	0x0698	GPDO232_233	0x06E8
GPDO76_79	0x064C	GPDO156_159	0x069C		

1 GPIO [35], GPIO [40], GPIO [41], GPIO [61], GPIO [63], GPIO [65], GPIO [67], GPIO [72], GPIO [77], GPIO [79], GPIO [81], GPIO [82], GPIO [83], GPIO [96], GPIO [97], GPIO [104], GPIO [105], GPIO [106], and GPIO [107] are not implemented on the MPC5675K. The associated PDO[*n*] address is reserved.

48.5.2.11 GPIO Pad Data Input Registers (GPDI*n*)

The GPIO Pad Data Input Registers (GPDI0_3–GPDI232_233) can be used to read the GPIO pad data with a byte access.

Address: Base + 0x0800–0x08E8. See Table 48-18. Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI[0]	0	0	0	0	0	0	0	PDI[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI[2]	0	0	0	0	0	0	0	PDI[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-13. Port GPIO Pad Data Input register (GPDI*n*)

Table 48-17. GPDI*n* field descriptions

Field	Description
PDI[*x*]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 0 The value of the data in signal for the corresponding GPIO pad is logic low. 1 The value of the data in signal for the corresponding GPIO pad is logic high.

Table 48-18. GPDI*n* addresses

Register[1]	Address	Register	Address	Register	Address
GPDI0_3	0x0800	GPDI80_83	0x0850	GPDI160_163	0x08A0
GPDI4_7	0x0804	GPDI84_87	0x0854	GPDI164_167	0x08A4

Table 48-18. GPDI*n* addresses (continued)

Register[1]	Address	Register	Address	Register	Address
GPDI8_11	0x0808	GPDI88_91	0x0858	GPDI168_171	0x08A8
GPDI12_15	0x080C	GPDI92_95	0x085C	GPDI172_175	0x08AC
GPDI16_19	0x0810	GPDI96_99	0x0860	GPDI176_179	0x08B0
GPDI20_23	0x0814	GPDI100_103	0x0864	GPDI180_183	0x08B4
GPDI24_27	0x0818	reserved	0x0868	GPDI184_187	0x08B8
GPDI_28_31	0x081C	GPDI108_111	0x086C	GPDI188_191	0x08BC
GPDI32_35	0x0820	GPDI112_115	0x0870	GPDI192_195	0x08C0
GPDI36_39	0x0824	GPDI116_119	0x0874	GPDI196_199	0x08C4
GPDI40_43	0x0828	GPDI120_123	0x0878	GPDI200_203	0x08C8
GPDI44_47	0x082C	GPDI124_127	0x087C	GPDI204_207	0x08CC
GPDI48_51	0x0830	GPDI128_131	0x0880	GPDI208_211	0x08D0
GPDI52_55	0x0834	GPDI132_135	0x0884	GPDI212_215	0x08D4
GPDI56_59	0x0838	GPDI136_139	0x0888	GPDI216_219	0x08D8
GPDI60_63	0x083C	GPDI140_143	0x088C	GPDI220_223	0x08DC
GPDI64_67	0x0840	GPDI144_147	0x0890	GPDI224_227	0x08E0
GPDI68_71	0x0844	GPDI148_151	0x0894	GPDI228_231	0x08E4
GPDI72_75	0x0848	GPDI152_155	0x0898	GPDI232_233	0x08E8
GPDI76_79	0x084C	GPDI156_159	0x089C		

1 GPIO [35], GPIO [40], GPIO [41], GPIO [61], GPIO [63], GPIO [65], GPIO [67], GPIO [72], GPIO [77], GPIO [79], GPIO [81], GPIO [82], GPIO [83], GPIO [96], GPIO [97], GPIO [104], GPIO [105], GPIO [106], and GPIO [107] are not implemented on the MPC5675K. The associated PDI[*n*] address is reserved.

48.5.2.12 Parallel GPIO Pad Data Out Register (PGPDO0–PGPDO7)

The Parallel GPIO Pad Data Out (PGPDO0–PGPDO7) registers are used to set or clear the respective pads of the device.

Address: Base + 0x0C00–0x0C1C. See Table 48-20. Access: User read/write

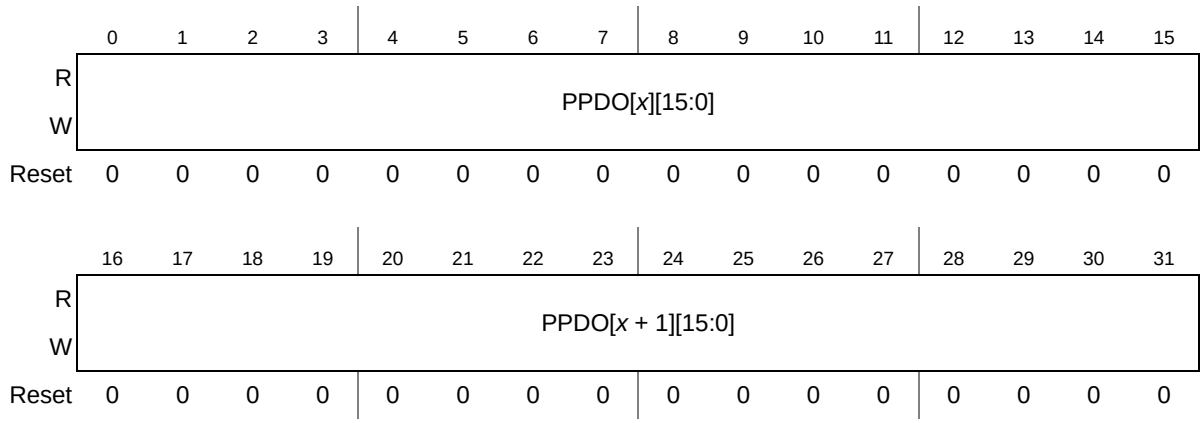


Figure 48-14. Parallel GPIO Pad Data Out Register (PGPDO*n*)

Table 48-19. PGPDO*n* field descriptions

Field	Description
PPDO[x]	Parallel Pad Data Out Write or read the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233). The x and bit index define which PPDO register bit is equivalent to which PDO register bit according to the following equation: PPDO[*x*][*y*] = PDO[(*x* × 16) + *y*]

Table 48-20. PGPDO*n* addresses

Register	Address	Register	Address
PGPDO0	0x0C00	PGPDO4	0x0C10
PGPDO1	0x0C04	PGPDO5	0x0C14
PGPDO2	0x0C08	PGPDO6	0x0C18
PGPDO3	0x0C0C	PGPDO7	0x0C1C

NOTE

The PGPDO registers access the same physical resource as the PDO and MPGPDO address locations. Some examples of the mapping:

PPDO[0][0] = PDO[0]

PPDO[2][0] = PDO[32]

48.5.2.13 Parallel GPIO Pad Data In Register (PGPDI0–PGPDI7)

The Parallel GPIO Pad Data In (PGPDI0–PGPDI7) registers hold the synchronized input value from the pads.

Address: Base + 0x0C40–0x0C5C. See Table 48-22. Access: User read-only

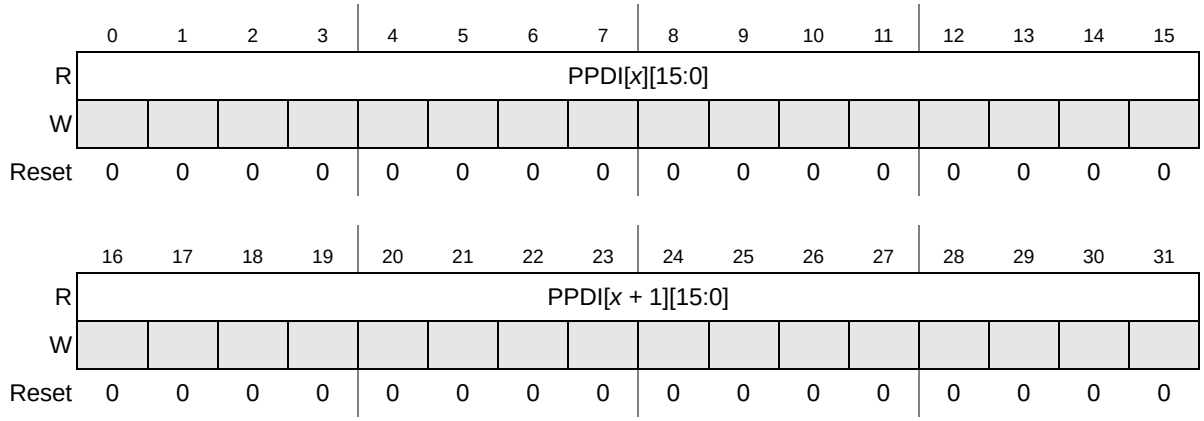


Figure 48-15. Parallel GPIO Pad Data In Register (PGPDI0–PGPDI7)

Table 48-21. PGPDI0_7 field descriptions

Field	Description
PPDI[*x*]	Parallel Pad Data In Read the current pad value. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Input Registers (GPDIn). The x and bit index define which PPDI register bit is equivalent to which PDI register bit according to the following equation: PPDI[*x*][*y*] = PDI[(*x* * 16) + *y*]

Table 48-22. PGPDI*n* addresses

Register	Address	Register	Address
PGPDI0	0x0C40	PGPDI4	0x0C50
PGPDI1	0x0C44	PGPDI5	0x0C54
PGPDI2	0x0C48	PGPDI6	0x0C58
PGPDI3	0x0C4C	PGPDI7	0x0C5C

48.5.2.14 Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO15)

The Masked Parallel GPIO Pad Data Out (MPGPDO0–MPGPDO15) registers can be used to selectively modify the pad values associated to PPDO[*x*][15:0]. The MPGPDO[*x*] register may only be accessed with 32-bit writes. 8-bit or 16-bit writes do not modify any bits in the register, and cause a transfer error response by the module. Read accesses return 0.

Address: Base + 0x0C80–0x0CBC. See Table 48-24. Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	MASK[*x*][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	MPPDO[*x*][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-16. Masked Parallel GPIO Pad Data Out Register (MPGPDO*n*)

Table 48-23. MPGPDO*n* field descriptions

Field	Description
MASK[*x*] [15:0]	Mask Field Each bit corresponds to one data bit in the MPPDO[*x*] register at the same bit location. 0 The associated bit value in the MPPDO[*x*] field is ignored. 1 The associated bit value in the MPPDO[*x*] field is written.
MPPDO[*x*] [15:0]	Masked Parallel Pad Data Out Write the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233). The x and bit index define which MPPDO register bit is equivalent to which PDO register bit according to the following equation: MPPDO[*x*][*y*] = PDO[(*x* ×- 16) + *y*]

Table 48-24. MPGPDO*n* addresses

Register	Address	Register	Address
MPGPDO0	0x0C80	MPGPDO8	0x0CA0
MPGPDO1	0x0C84	MPGPDO9	0x0CA4
MPGPDO2	0x0C88	MPGPDO10	0x0CA8
MPGPDO3	0x0C8C	MPGPDO11	0x0CAC
MPGPDO4	0x0C90	MPGPDO12	0x0CB0
MPGPDO5	0x0C94	MPGPDO13	0x0CB4
MPGPDO6	0x0C98	MPGPDO14	0x0CB8
MPGPDO7	0x0C9C	MPGPDO15[1]	0x0CBC

1 MPGPDO[15] has no corresponding GPDOs, because GPDOs are implemented only to GPDO233. MGPDO[15] corresponds to GPDO240–255.

48.5.2.15 Interrupt Filter Maximum Counter Register (IFMC0–IFMC31)

The Interrupt Filter Maximum Counter Registers (IFMC0–IFMC31) are used to configure the filter counter associated with each digital glitch filter.

Address: Base + 0x1000–0x107C. See Table 48-26. Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MAXCNT*x*			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-17. Interrupt Filter Maximum Counter Register (IFMCn)

Table 48-25. IFMCn field descriptions

Field	Description
MAXCNT*x*	Maximum Interrupt Filter Counter setting. Filter Period = T(CK) × MAXCNT*x* + *n* × T(CK) Where (n can be –1 to 3) MAXCNT*x* can be 0 to 15 T(CK): Prescaled Filter Clock Period, which is IRC clock prescaled to IFCP value T(IRC): Basic Filter Clock Period: 62.5 ns (F = 16 MHz)

Table 48-26. IFMC*n* addresses

Register	Address	Register	Address	Register	Address	Register	Address
IFMC0	0x1000	IFMC8	0x1020	IFMC16	0x1040	IFMC24	0x1060
IFMC1	0x1004	IFMC9	0x1024	IFMC17	0x1044	IFMC25	0x1064
IFMC2	0x1008	IFMC10	0x1028	IFMC18	0x1048	IFMC26	0x1068
IFMC3	0x100C	IFMC11	0x102C	IFMC19	0x104C	IFMC27	0x106C
IFMC4	0x1010	IFMC12	0x1030	IFMC20	0x1050	IFMC28	0x1070
IFMC5	0x1014	IFMC13	0x1034	IFMC21	0x1054	IFMC29	0x1074
IFMC6	0x1018	IFMC14	0x1038	IFMC22	0x1058	IFMC30	0x1078
IFMC7	0x101C	IFMC15	0x103C	IFMC23	0x105C	IFMC31	0x107C

48.5.2.16 Interrupt Filter Clock Prescaler Register (IFCPR)

The Interrupt Filter Clock Prescaler Register (IFCPR) configures a clock prescaler that selects the clock for all digital filter counters in the SIUL.

Address: Base + 0x1080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFCP[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 48-18. Interrupt Filter Clock Prescaler Register (IFCPR)

Table 48-27. IFCPR field descriptions

Field	Description
IFPC[3:0]	Interrupt Filter Clock Prescaler setting. Prescaled Filter Clock Period = T(IRC) × (IFCP + 1) T(IRC) is the IRC clock period. IFCP can be 0 to 15.

48.6 Functional description

48.6.1 Pad control

The SIUL controls the configuration and electrical characteristics of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The SIUL allows configuring each pad as either a GPIO or as one or more alternate functions (input or output). The pad configuration registers (PCR*n*, see Section 48.5.2.8, Pad Configuration Registers (PCRn)) allow software control of the static electrical characteristics of external pins with a single write. These PCRs are used to configure the following pad features:

- Open drain output enable
- Input hysteresis enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode configuration

48.6.2 Functional I/O multiplexing

The SIUL supports the selection of as many as four different output functions per functional pad. As many as 16 different input functions can be driven per pad.

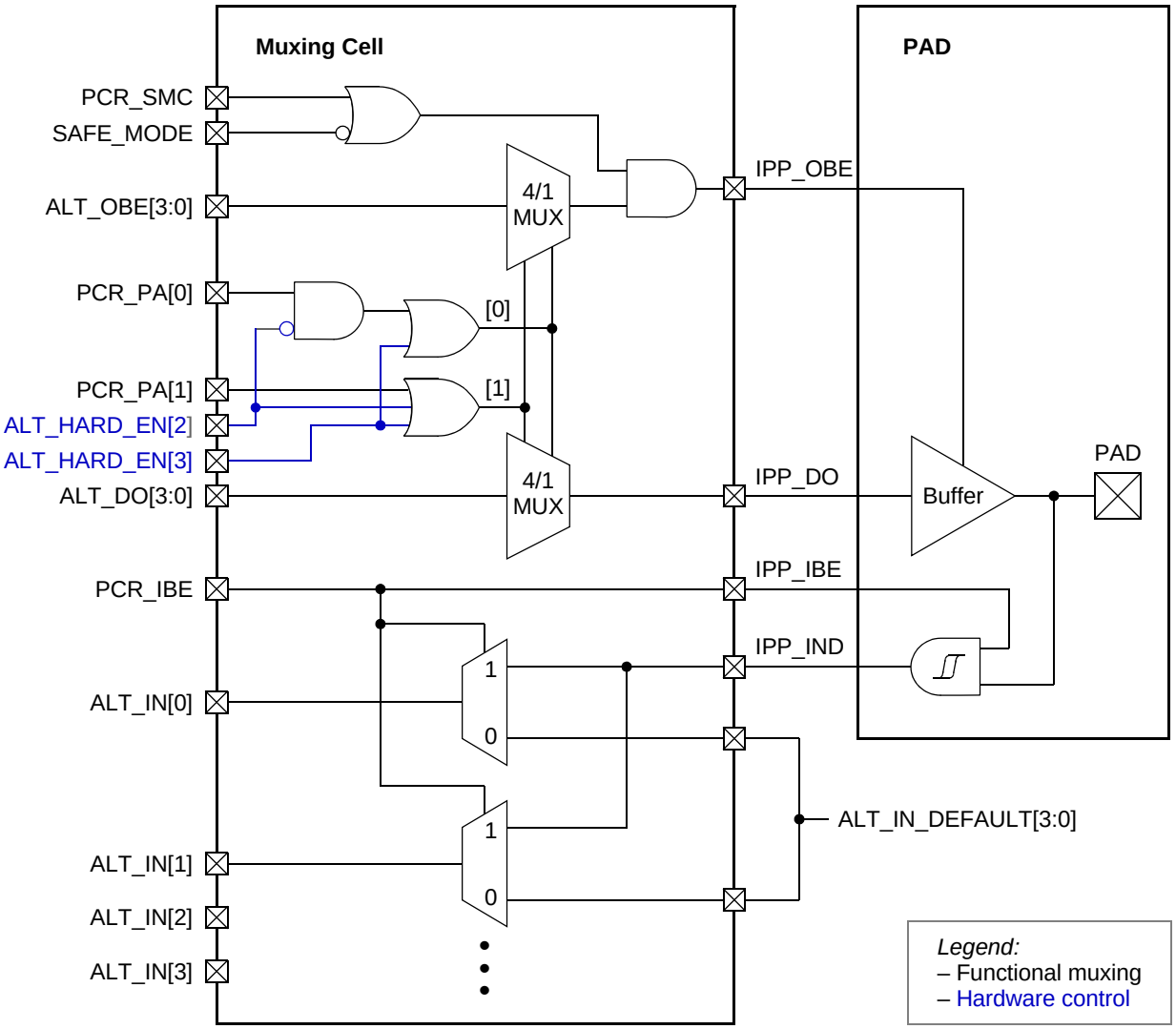


Figure 48-19. Functional I/O multiplexing diagram

48.6.3 GPIO pads

The SIUL allows each pad to be configured as either a GPIO pad or as one or more alternate functions (input or output), the function of which is normally determined by the peripheral that uses the pad.

The SIUL manages 108 GPIO pads organized as ports that can be accessed for data reads and writes as 32-bit, 16-bit, or 8-bit.

As shown in Figure 48-20, all port accesses are identical, with each read or write being performed only at a different location to access a different port width.

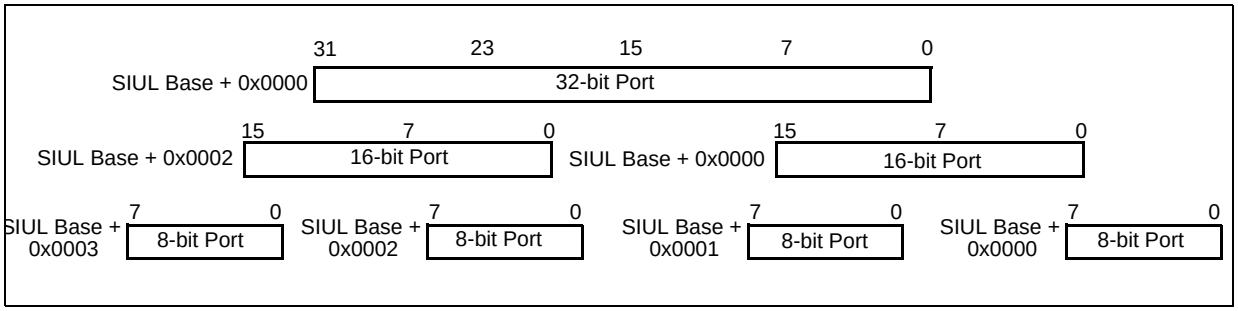


Figure 48-20. Data port width configurations for different port width accesses

This implementation requires that the registers are arranged to support this range of port widths without having to split reads or writes into multiple accesses.

The SIUL has separate data input (GPDI*n_n*, see Section 48.5.2.11, GPIO Pad Data Input Registers (GPDIn)) and data output (GPDO*n_n*, see Section 48.5.2.10, GPIO Pad Data Output Registers (GPDO0_3–GPDO232_233)) registers for all pads, allowing the possibility of reading back an input or output value of a pad directly. This allows validating the pad configuration rather than confirming the value that was written to the data register by accessing the data input registers.

The data output registers support both read and write operations. The data input registers support read access only.

When the pad is configured to use one of its alternate functions, the data input value reflects the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non-GPIO), this write is not reflected by the pad value until reconfigured to GPIO.

The allocation of a specific input function to a specific pin is defined by the PSMI registers (PCR*n*, see Section 48.5.2.8, Pad Configuration Registers (PCRn)).

48.6.4 External interrupts

The SIUL supports 32 external interrupts, EIRQ0–EIRQ31.

The SIUL supports four interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts, combined together with the presence of a flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

See Figure 48-21 for an overview of the external interrupt implementation.

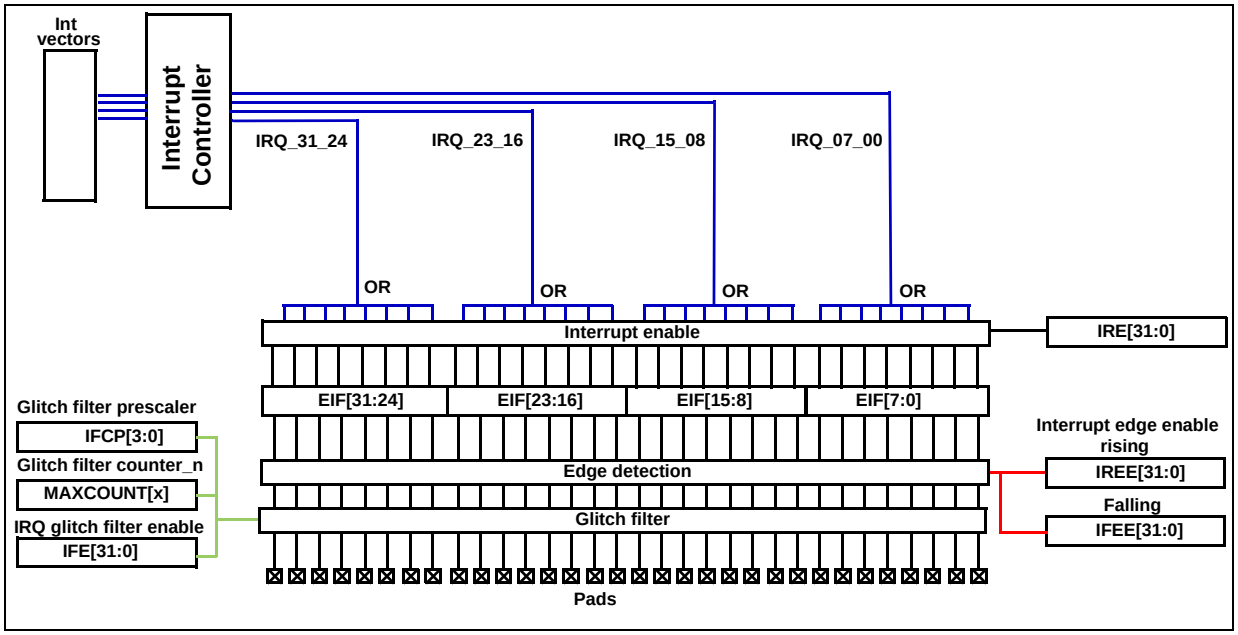


Figure 48-21. External interrupt pad diagram

48.6.4.1 External interrupt management

Each interrupt can be enabled or disabled independently. This can be performed using the Interrupt Request Enable Register (IRER, see Section 48.5.2.4, Interrupt Request Enable Register (IRER)). A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an active falling edge, or with both edges active. A setting with both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEER.

Each external interrupt supports an individual flag that is held in the flag register (see Section 48.5.2.3, Interrupt Status Flag Register (ISR)). This register is a write-1-to-clear register type, preventing inadvertent overwriting of other flags in the same register.

48.7 Pin muxing

For pin muxing, see Chapter 3, Signal Description.

This page is intentionally left blank.

Chapter 49
System Status and Configuration Module (SSCM)

49.1 Introduction

49.1.1 Overview

The SSCM, pictured in Figure 49-1, provides central SoC functionality. The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debugging the system.

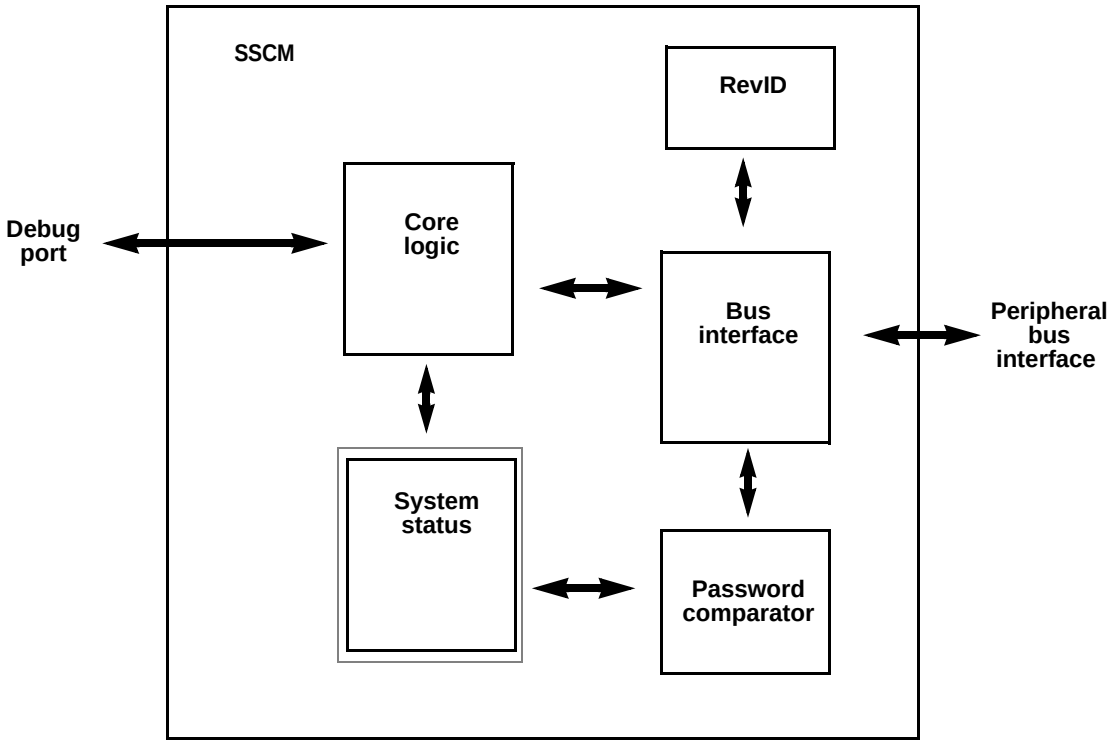


Figure 49-1. SSCM block diagram

49.1.2 Features

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search code flash memory for bootable sector
  - DMA status

- Device identification information (MCU ID registers)
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

49.1.3 Modes of operation

The SSCM operates identically in all system modes.

49.2 External signal description

The SSCM has no external pins.

49.3 Memory map and register description

This section provides a detailed description of all memory-mapped registers in the SSCM.

Table 49-1 shows the memory map for the SSCM.

Table 49-1. SSCM memory map

Offset from SSCM_BASE (0xC3FD_8000)	Register	Access[1]	Reset Value[2]	Location
0x0000	System Status Register (SSCM_STATUS)	R/W	0x U0U0[3]	on page 1655
0x0002	System Memory and ID Register (SSCM_MEMCONFIG)	R	0xB021	on page 1656
0x0004–0x0005	Reserved			
0x0006	Error Configuration Register (SSCM_ERROR)	R/W	0x0000	on page 1657
0x0008	Debug Status Port Register (SSCM_DEBUGPORT)	R/W	0x0000	on page 1658
0x000A–0x000B	Reserved			
0x000C	Password Comparison High Word Register (SSCM_PWCMPH)	W	0x0000_0000	on page 1659
0x0010	Password Comparison Low Word Register (SSCM_PWCMPL)	W	0x0000_0000	on page 1659
0x0014–0x0017	Reserved			
0x0018	DPM Boot Register (SSCM_DPMBOOT)	R/W	0x0000_0000	on page 1660
0x001C	DPM Boot Key Register (SSCM_DPMKEY)	W	0x0000_0000	on page 1661
0x0020	User Option Status Register (SSCM_UOPS)	R	0xUUUU_0000	on page 1661
0x0024–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

3 One or more bits (indicated by “U”) are of indeterminate value at reset. See register for more information.

All registers are accessible via 8-bit, 16-bit, or 32-bit accesses unless otherwise noted. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the SSCM_STATUS register is accessible by a 16-bit read/write to address Base + 0x0002, but performing a 16-bit access to Base + 0x0003 is illegal.

49.3.1 Register descriptions

The following registers are available in the SSCM. Those bits that are shaded out are reserved for future use. To optimize future compatibility, these bits should be masked out during any read/write operations to avoid conflict with future revisions.

49.3.1.1 System Status Register (SSCM_STATUS)

The System Status (SSCM_STATUS) register reflects the current state of the system.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LSM	CER	0	NXEN1	NXEN	PUB	SEC	0	BMODE			VLE	0	0	0	0
W		w1c														
Reset	0/1[1]	0	0	0	0	0	0	0	0/1	0/1	0/1	1	0	0	0	0

Figure 49-2. System Status Register (SSCM_STATUS)

1 Reset value depends on the associated option bit.

Table 49-2. SSCM_STATUS allowed register accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Allowed

Table 49-3. SSCM_STATUS field descriptions

Field	Description
LSM	Lock Step Mode. This field indicates how the two processor cores of the device are used. Lock Step Mode (LSM) is used to increase safety. Dual Processor Mode (DPM) is used to increase performance. 0 Device is in DPM. 1 Device is in LSM. **Note:** This field is used only to see what mode (LSM or DPM) the chip is in. To configure the chip to run in one of these modes, see Section 5.6, Selecting LSM or DPM.
CER	Configuration Error. This field indicates that the SSCM has detected a configuration error during startup. 0 No configuration problem detected by the SSCM. 1 Device configuration is not correct.
NXEN1	Processor 1 Nexus enabled.
NXEN	Processor 0 Nexus enabled.

Table 49-3. SSCM_STATUS field descriptions (continued)

Field	Description
PUB	Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed. 0 Serial boot mode with private flash memory password is allowed, provided the key hasn't been swallowed. 1 Serial boot mode with public password is allowed.
SEC	Security Status. This bit reflects the current security state of the flash memory. 0 The flash memory is not secured. 1 The flash memory is secured.
VLE	Variable Length Instruction Mode. When booting from flash memory, this field indicates that the code stored there is using the VLE instruction set. The value of this field is determined by the RCHW field of the flash memory boot sector. 0 Main flash memory contains standard PPC code. 1 Main flash memory contains VLE code.
BMODE	Device Boot Mode. 000 Reserved for FlexRay Boot Serial Boot Loader. 001 FlexCAN Serial Boot Loader. 010 LINFlexD Serial Boot Loader. 011 Single Chip. 100 Expanded Chip. All other values are reserved. This field is updated only during reset.

49.3.1.2 System Memory and ID (SSCM_MEMCONFIG) Register

The System Memory Configuration (SSCM_MEMCONFIG) register is a read-only register that reflects the memory configuration of the system. It also contains the JTAG ID.

Address: Base + 0x0002 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	JPIN										IVLD	MREV				DVLD
W																
Reset	1	0	1	1	0	0	0	0	0/1[1]	0/1[1]	1	0	0	0	0	1

Figure 49-3. System Memory and ID (SSCM_MEMCONFIG) Register

[1] Dependent on device version.

Table 49-4. SSCM_MEMCONFIG field descriptions

Field	Description
JPIN	Part Identification Number for MPC5675K. Contains the part number of the device. Cut1 0x2C0 (0b1011000000) Cut2 0x2C1 (0b1011000001)

Table 49-4. SSCM_MEMCONFIG field descriptions (continued)

Field	Description
IVLD	Instruction flash memory valid. This bit identifies whether or not the on-chip Code flash memory is accessible in the system memory map. The flash memory may not be accessible due to security limitations, or because there is no flash memory in the system. 0 Instruction flash memory is not accessible. 1 Instruction flash memory is accessible.
MREV	Minor Mask Revision.
DVLD	Data flash memory valid. This bit identifies whether or not the on-chip Data flash memory is visible in the system memory map. The flash memory may not be accessible due to security limitations, or because there is no flash memory in the system. 0 Data flash memory is not visible. 1 Data flash memory is visible.

Table 49-5. SSCM_MEMCONFIG Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed (also reads SSCM_STATUS register)
WRITE	Not Allowed	Not Allowed	Not Allowed

49.3.1.3 Error Configuration (SSCM_ERROR) Register

The Error Configuration (SSCM_ERROR) register is a read-write register that controls the error handling of the system.

Address: Base + 0x0006 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAE	RAE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-4. Error Configuration (SSCM_ERROR) Register

Table 49-6. SSCM_ERROR field descriptions

Field	Description
PAE	Peripheral Bus Abort Enable. This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception. 1 Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception.
RAE	Register Bus Abort Enable. This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception. 1 Illegal accesses to peripherals produce a Prefetch or Data Abort exception. **Note:** Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (that is, at the PBRIDGE level). In this case, the PER_ABORT and REG_ABORT register bits have no effect on the abort.

Table 49-7. SSCM_ERROR Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Not Allowed

49.3.1.4 Debug Status Port (SSCM_DEBUGPORT) Register

The Debug Status Port (SSCM_DEBUGPORT) register is used to (optionally) provide debug data on a set of pins.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DEBUG_MODE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-5. Debug Status Port (SSCM_DEBUGPORT) Register

Table 49-8. SSCM_DEBUGPORT field descriptions

Field	Description
DEBUG_MODE	Debug Status Port Mode. This field selects the alternate debug functionality for the Debug Status Port. 000 No alternate functionality selected. 001 Mode 1 selected. 010 Mode 2 selected. 011 Mode 3 selected. 100 Mode 4 selected. 101 Mode 5 selected. 110 Mode 6 selected. 111 Mode 7 selected. Table 49-9 describes the functionality of the Debug Status Port in each mode.

Table 49-9. Debug Status Port Modes

Pin[1]	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
0	STATUS[0]	STATUS[8]	MEMCONFIG[0]	MEMCONFIG[8]	Reserved	Reserved	Reserved
1	STATUS[1]	STATUS[9]	MEMCONFIG[1]	MEMCONFIG[9]	Reserved	Reserved	Reserved
2	STATUS[2]	STATUS[10]	MEMCONFIG[2]	MEMCONFIG[10]	Reserved	Reserved	Reserved
3	STATUS[3]	STATUS[11]	MEMCONFIG[3]	MEMCONFIG[11]	Reserved	Reserved	Reserved
4	STATUS[4]	STATUS[12]	MEMCONFIG[4]	MEMCONFIG[12]	Reserved	Reserved	Reserved
5	STATUS[5]	STATUS[13]	MEMCONFIG[5]	MEMCONFIG[13]	Reserved	Reserved	Reserved
6	STATUS[6]	STATUS[14]	MEMCONFIG[6]	MEMCONFIG[14]	Reserved	Reserved	Reserved
7	STATUS[7]	STATUS[15]	MEMCONFIG[7]	MEMCONFIG[15]	Reserved	Reserved	Reserved

1 All signals are active high, unless otherwise noted.

Table 49-10. SSCM_DEBUGPORT allowed register accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not allowed
WRITE	Allowed	Allowed	Not allowed

49.3.1.5 Password Comparison Registers (SSCM_PWCMPH and SSCM_PWCMPL)

The Password Comparison Registers (SSCM_PWCMPH and SSCM_PWCMPL) allow unsecuring the device if the correct password is known. They are intended for device-internal operation only.

Address: Base + 0x000C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-6. Password Comparison Register High Word (SSCM_PWCMPH)

Address: Base + 0x0010 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-7. Password Comparison Register Low Word (SSCM_PWCMPL)

Table 49-11. SSCM_PWCMPH/L field descriptions

Field	Description
PWD_HI	Upper 32 bits of the password
PWD_LO	Lower 32 bits of the password

Table 49-12. SSCM_PWCMPH/L Allowed Register Accesses

	8-bit	16-bit	32-bit[1]
READ	Allowed	Allowed	Allowed
WRITE	Not Allowed	Not Allowed	Allowed

1 All 32-bit accesses must be aligned to 32-bit addresses (for example, 0x0, 0x4, 0x8, or 0xC).

In order to unsecure the device, the password needs to be written first to the upper word (to the SSCM_PWCMPH register), then to the lower word (to the SSCM_PWCMPL register). The SSCM then inserts a delay, compares the password, and if the password is correct, unlocks the device.

49.3.1.6 DPM Boot Register (SSCM_DPMBOOT)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	P2BOOT[29:14]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	P2BOOT[13:0]														DVLE	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-8. DPM Boot (SSCM_DPMBOOT) Register

Table 49-13. SSCM_DPMBOOT field descriptions

Field	Description
P2BOOT	Determines the location from which the second processor boots once the main processor releases it from reset. This field is used only if the device is operating in DPM.
DVLE	Determines whether the second processor executes in VLE mode. 0 BookE mode. 1 VLE mode. **Note:** This field is used only if the device is operating in DPM.

49.3.1.7 DPM Boot Key Register (SSCM_DPMKEY)

Address: Base + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	KEY															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 49-9. DPM Boot Key Register (SSCM_DPMKEY)

Table 49-14. SSCM_DPMKEY field descriptions

Field	Description
KEY	Control key. This field activates the second core in DPM. The following sequence is required: 1. Write to the SSCM_DPMBOOT register. 2. Write the value 0101101011110000 (0x5AF0) to the key field. 3. Write the value 1010010100001111 (0xA50F) to the key field. After this, the second core starts executing from the address specified in the SSCM_DPMBOOT register.

49.3.1.8 User Option Status (SSCM_UOPS) Register

The User Option Status (SSCM_UOPS) Register mirrors bits written in the shadow flash memory block. For more information, see Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO).

Address: Base + 0x0020 Access: User read-only

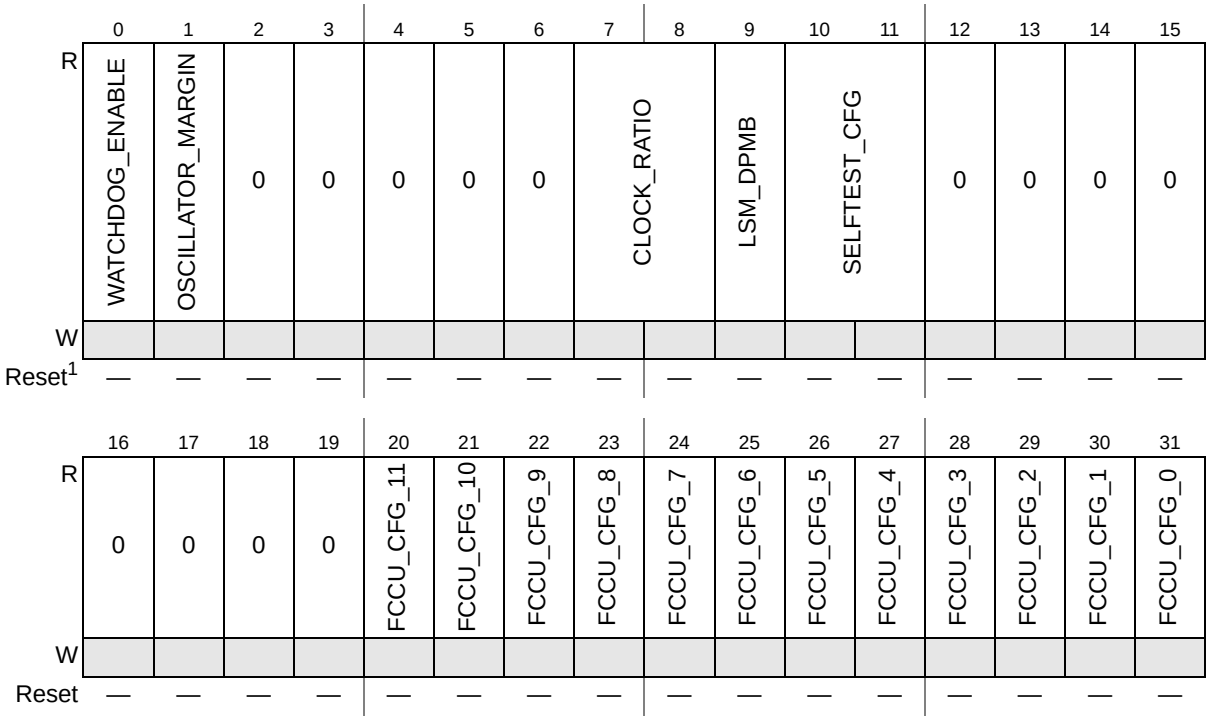


Figure 49-10. User Option Status (SSCM_UOPS) Register

[1] Varies depending on the contents of the Test flash memory block.

Table 49-15. SSCM_UOPS field descriptions

Field	Description
WATCHDOG_ENABLE	This bit shows whether the SWT is enabled. 0 SWT is disabled 1 SWT is enabled. **Note:** In DPM, applies only to SWT_0; SWT_1 is always off until enabled by software.
OSCILLATOR_MARGIN	XOSC oscillation margin select. 0 Selects lower XOSC consumption and lower oscillator margin. 1 Selects higher XOSC consumption and higher oscillator margin.
CLOCK_RATIO	This bit shows the clock ratio between core, platform, and Flash_IPG clock. 00 Configuration during reset 01 1:1 Mode 10 1:3 Mode 11 1:2 Mode (default after reset) See Section 13.3, System clock dividers, for more information.
LSM_DPMB	This bit shows whether the device is in LSM or DPM. 0 Device is in DPM. 1 Device is in LSM.
SELFTEST_CFG	These bits show the self-test configuration via SSCM or CPU selection. 00 CPU 01 Reserved 10 Reserved 11 SSCM

Table 49-15. SSCM_UOPS field descriptions (continued)

Field	Description
FCCU_CFG_11	FCCU Configuration bit 11. This bit mirrors the FCCU_CFG[CM] bit. Configuration mode 0 Configuration labelling: a specific FCCU_F configuration is assigned in CONFIG state. 1 Configuration transparency: the FCCU_F protocol is the same in CONFIG and NORMAL states.
FCCU_CFG_10	FCCU Configuration bit 10. This bit mirrors the FCCU_CFG[SM] bit. Switching mode 0 FCCU_F protocol (dual-rail, time-switching) slow switching mode. 1 FCCU_F protocol (dual-rail, time-switching) fast switching mode. SM has no effect on the bi-stable protocol.
FCCU_CFG_9	FCCU Configuration bit 9. This bit mirrors the FCCU_CFG[PS] bit. Polarity selection 0 FCCU_F[1] active high, FCCU_F[0] active high. 1 FCCU_F[1] active low, FCCU_F[0] active low.
FCCU_CFG_8 FCCU_CFG_7 FCCU_CFG_6	FCCU Configuration bits 8:6. These bits mirror the FCCU_CFG[FOM] field. Fault Output Mode selection 000 Dual-Rail (default state; FCCU_F[1:0]= outputs). 001 Time Switching (FCCU_F[1:0]= outputs). 010 Bi-Stable (FCCU_F[1:0]= outputs). 011 Reserved. 100 Reserved. 101 Test0 (FCCU_F[0] = input, FCCU_F[1] = output). 110 Test1 (FCCU_Ff[0] = output, FCCU_F[1] = output). 111 Test2 (FCCU_F[0] = output, FCCU_F[1] = input). The output level of the FCCU_F[0:1] pins in the test modes is controlled in the FCCU I/O Control Register (FCCU_EINOUT). **Note:** In Test*n* mode, a simple double-stage resynchronization stage is used to resynchronize the FCCU_F[0:1] input/outputs on the system/safe clock.
FCCU_CFG_5 FCCU_CFG_4 FCCU_CFG_3 FCCU_CFG_2 FCCU_CFG_1 FCCU_CFG_0	FCCU Configuration bits 5:0. These bits mirror the FCCU_CFG[FOP] field. Fault Output Prescaler FOP defines the prescaler setting used to generate the FCCU_F protocol frequency. 00 0000 Input clock frequency (RC_{16MHz} clock) is divided by 2048. 00 0001 Input clock frequency (RC_{16MHz} clock) is divided by 4096. 00 0010 Input clock frequency (RC_{16MHz} clock) is divided by 6144. 00 0011 Input clock frequency (RC_{16MHz} clock) is divided by 8192. 00 0100 Input clock frequency (RC_{16MHz} clock) is divided by 10240. 00 0101 Input clock frequency (RC_{16MHz} clock) is divided by 12288. 00 0110 Input clock frequency (RC_{16MHz} clock) is divided by 14336. 00 0111 Input clock frequency (RC_{16MHz} clock) is divided by 16384. 00 1000 Input clock frequency (RC_{16MHz} clock) is divided by 18432 (This is the reset value Id est ~ 868 Hz is the out-of- reset frequency of the FCCU_F[0:1] pins). The following equation gives the FCCU_F frequency: $$Fccu0F_{freq} = RC_{16MHz} / (1024 \times (FOP + 1) \times 2)$$

49.4 Functional description

The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debugging the system.

49.4.1 Reset Configuration Half Word Source (RCHW)

MPC5675K flash memory is partitioned into boot sectors shown in Table 49-17.

Each boot sector contains the Reset Configuration Half-Word (RCHW) at offset 0x0.

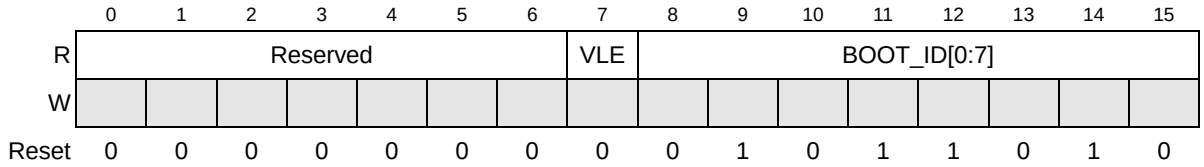


Figure 49-11. Reset Configuration Half Word (RCHW)

Table 49-16. RCHW field descriptions

Field	Description
0–6	Reserved
VLE	Selects the VLE or Book E instruction set. 0 Selects the classic PowerPC instruction set. 1 Selects the VLE instruction set (default).
BOOT_ID[0:7]	Valid boot identifier is 0x5A.

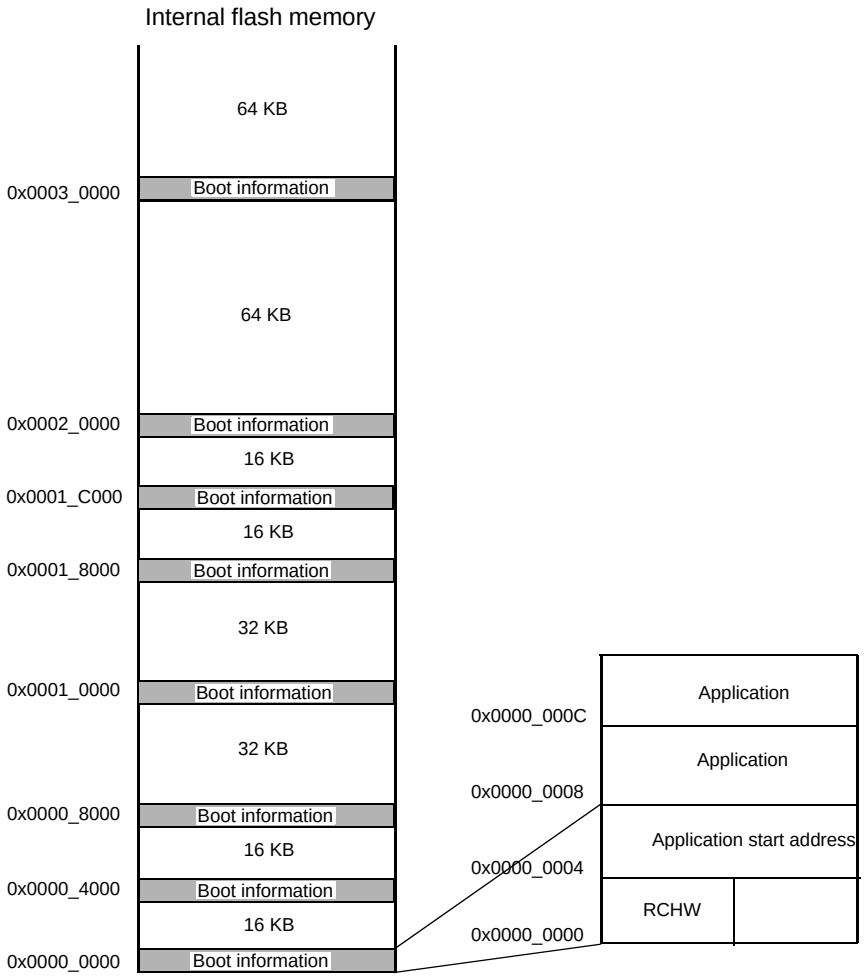


Figure 49-12. MPC5675K flash memory partitioning and RCHW search

Table 49-17. Flash boot sector locations

Block	Address
0	0x0000_0000
1	0x0000_4000
2	0x0000_8000
3	0x0001_0000
4	0x0001_8000
5	0x0001_C000
6	0x0002_0000
7	0x0003_0000

This page is intentionally left blank.

Chapter 50
Self-Test Control Unit (STCU)

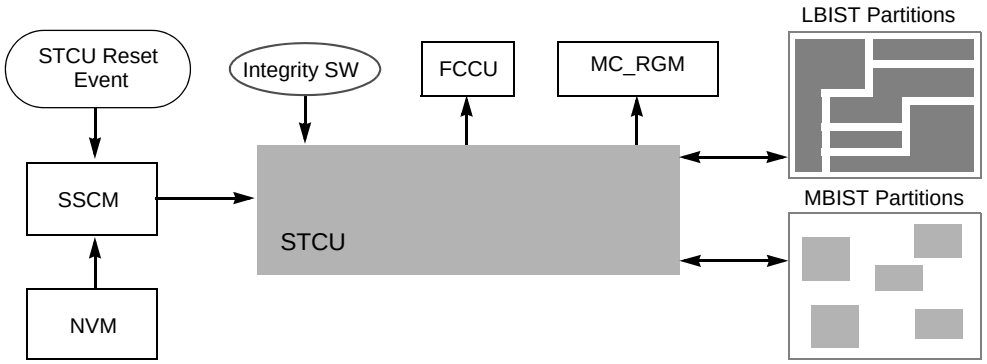
50.1 Introduction

The STCU is a component within the overall Safety Integrity Subsystem. The STCU controls the sequencing of the device's self-test before the primary user application starts running. The goal of the self-test is to detect physical defects in the digital logic and embedded memories with enough coverage to meet the required Safety Integrity Level (SIL, ASIL) of the system.

To the user, the purpose of the STCU is to display the results of the self-test.

50.1.1 Safety Integrity Subsystem

Figure 50-1 shows the STCU as a component of the Safety Integrity Subsystem on the device.



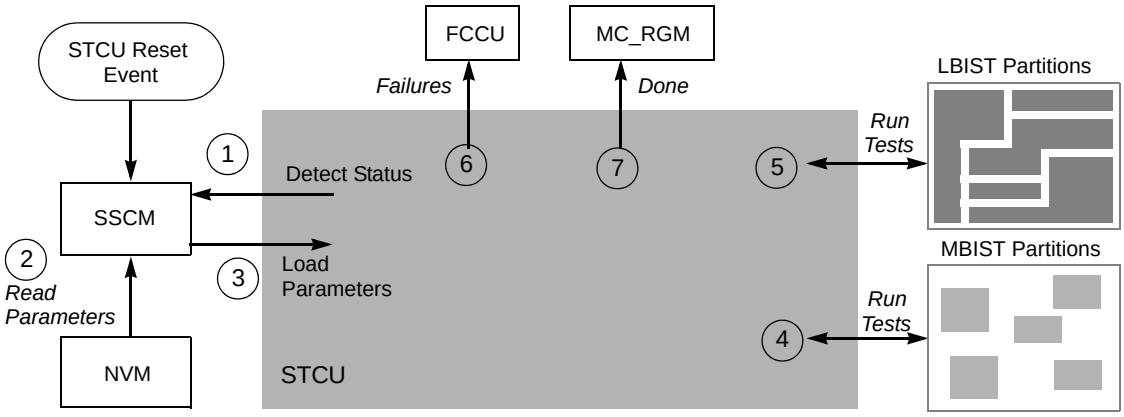
Component[1]	Description
FCCU	The Fault Collection Control Unit collects errors and controls the safety state of the device.
Integrity SW	The Safety Integrity Software checks the STCU status before passing control over to application software.
LBIST Partitions	The set of individual logic block partitions included in the self-test
MBIST Partitions	The set of individual embedded memory blocks included in the self-test
MC_RGM	Reset generation module
NVM	The flash nonvolatile memory contains the initial self-test parameters.
SSCM	The System Status and Configuration Module is the central control for device configuration after reset.
STCU	The Self-Test Control Unit manages the device self-test.
STCU Reset Event	The following reset events trigger the SSCM to activate the STCU: • Power-on reset • Destructive reset • External reset

[1] Components are the hardware and software that make up a subsystem. Events that affect subsystem behavior are

Figure 50-1. STCU within the Safety Integrity Subsystem

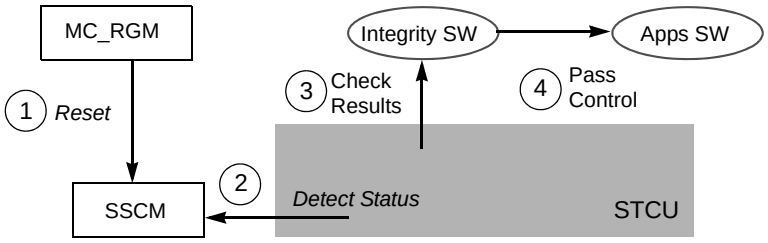
When an STCU Reset Event occurs, the device goes through a two-phase boot sequence:

1. Self-test phase (see Figure 50-2)
2. Functional-reset phase (see Figure 50-3)



1. After an STCU reset event, the SSCM detects that the device self-test has not been run yet.
2. The SSCM reads the self-test parameters from flash nonvolatile memory (NVM).
3. The SSCM loads the self-test parameters into the STCU and passes control over to the STCU.
4. The STCU manages the MBISTs and updates its internal status.
5. The STCU manages the LBISTs and updates its internal status.
6. If faults are detected, the STCU reports the test failures to the FCCU.
7. The STCU signals the MC_RGM that the tests are complete, and the boot sequence proceeds to the next phase (see Figure 50-3). However, if a SIR fault occurs, the STCU keeps the device in reset until an STCU Reset Event is applied.

Figure 50-2. Boot sequence phase 1: self-test



1. The MC_RGM triggers a functional reset.
2. The SSCM detects that the device self-test has been run and passes control over to the CPUs.
3. The Integrity software checks the results of the self-test. See Section 50.1.2, Integrity software operations.
4. If the Integrity software check passes, the device can be considered as OK and control can be passed to the application software.

Figure 50-3. Boot sequence phase 2: functional reset

50.1.2 Integrity software operations

The Integrity software should perform operations based on the STCU status conditions after the self-test. Even if no errors are reported, the Integrity software should confirm that the expected and actual values within the CRC and LBIST MISR registers do not indicate an error. This software confirmation prevents a fault within the STCU itself incorrectly indicating that the self-test passed.

50.1.2.1 Reported errors

In the case of reported errors, the Integrity software should:

- Read the STCU_LBS flag register to determine which LBISTs failed.
- Read the STCU_LBE flag register to determine which LBISTs did not finish.
- Read the STCU_MBSL flag register to determine which MBISTs failed.
- Read the STCU_MBSH flag register to determine which MBISTs failed.
- Read the STCU_MBEL flag register to determine which MBISTs did not finish.
- Read the STCU_MBEH flag register to determine which MBISTs did not finish.
- Read the STCU_ERR register to check whether there has been an internal STCU failure.

50.1.2.2 No reported errors

In the case of no reported errors, the Integrity software should confirm the following:

- The internal CRC computation result matches the expected value.
  Read the CRCE and CRCR registers to check the coherency with the STCU_ERR[CRCS] flag.
- The signature registers of each of the LBIST results match their corresponding expected values.
  For each LBIST:

— Read the STCU_LBMISREL/H and STCU_LBIST_NMISRRL/H registers to check the coherency with the STCU_LBS bits.
— Read the registers described in Section 50.1.2.1, Reported errors, and verify that their values are as expected.

50.2 STCU main features

The STCU features include the following:

- Performs a one-time self-test after a reset trigger event
- Provides register interfaces for both software and hardware:
  — Hardware: SSCM write-one-time register interface
  — Software: Internal Peripheral System (IPS) read-only register interface
- Manages 3 LBISTs
- Manages 58 MBISTs (embedded memory blocks)
- Performs self-checking: the Self Checker monitors critical internal signals during the self-test.
- Provides a rich set of status and error information:
  — Timeout flags if the self-test does not start or finish within a limited amount of time
  — Status flags for individual LBIST and MBIST operations
  — Flags for STCU internal errors
- Software can confirm the integrity of the CRC and LBIST status information by directly comparing the expected and actual results of the CRC and LBIST operations.

50.3 Block diagram and components

Figure 50-4 shows a block diagram of the STCU.

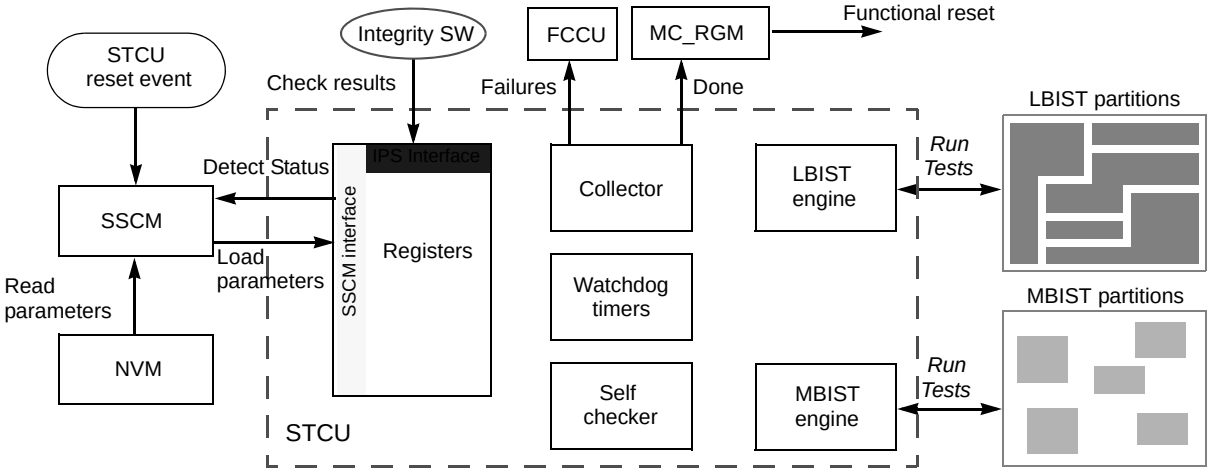


Figure 50-4. STCU block diagram

The main components of the STCU are:

- ***Registers***: Hold the self-test parameters and status flags (see Section 50.4, Memory map and register description):
  - — Scheduling activity
  - — LBIST setup
  - — Critical/Non-Critical/Stay-in-Reset fault mapping
  - — CRC and MISR expected values

  The IPS and SSCM interfaces provide access:
  - — *SSCM interface*—The SSCM uses this interface to program the STCU's self-test parameters without CPU intervention. A write-once mechanism disables the SSCM interface to prevent the STCU parameters from being reloaded after the self-test has been performed.
  - — *IPS interface*—Software running on the CPUs use this slave bus to access registers.
- ***Self checker***: Performs a CRC test on a sampled set of selected internal signals when the STCU is running
- ***Collector***: Collects and updates the status and error conditions related to the L/MBIST execution and STCU internal operation. The Collector sends the BIST results to the FCCU and signals the MC_RGM to begin a functional reset (unless a stay-in-reset fault is encountered).
- ***Watchdog timers:*** Provide timeout mechanisms to protect against the following:
  - — Dead-lock or runaway condition during the self-test
  - — STCU is activated but the self-test is not run
- ***LBIST engine***: Manages the testing of logic blocks on the device.
- ***MBIST engine***: Manages the testing of embedded memory blocks.

50.4 Memory map and register description

All registers shown in this section are defined as visible by the IPS interface.

50.4.1 Memory map

The STCU memory map is listed in Table 50-1.

For LBIST partitioning, see Section 50.5, LBIST partitioning.

Table 50-1. STCU register map

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x0000	STCU Run Register (STCU_RUN)	R/W	0x0000_0000	on page 1677
0x0004–0x000B	Reserved			
0x000C	STCU Configuration Register (STCU_CFG)	R/W	0x0000_0000	on page 1678

Table 50-1. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x0010	STCU Watchdog Register Granularity (STCU_WDGG)	R	0x0000_FFFF	on page 1678
0x0014	STCU CRC Expected Status Register (STCU_CRCE)	R	0xFFFF_FFFF	on page 1679
0x0018	STCU CRC Read Status Register (STCU_CRCR)	R	0x0000_0000	on page 1680
0x001C	STCU Error Register (STCU_ERR)	R	0x0000_0000	on page 1680
0x0020	Reserved			
0x0024	STCU LBIST Status Register (STCU_LBS)	R	0x0000_0000	on page 1682
0x0028	STCU LBIST End Flag Register (STCU_LBE)	R	0x0000_0000	on page 1683
0x002C	Reserved			
0x0030	STCU LBIST Critical FM Register (STCU_LBCFM)	R	0x0000_0000	on page 1683
0x0034	STCU LBIST Stay-In-Reset FM Register (STCU_LBSFM)	R	0x0000_0000	on page 1684
0x0038	Reserved			
0x003C	STCU MBIST Status Low Register (STCU_MBSL)	R	0x0000_0000	on page 1685
0x0040	STCU MBIST Status High Register (STCU_MBSH)	R	0x0000_0000	on page 1685
0x0044	STCU MBIST End Flag Low Register (STCU_MBEL)	R	0x0000_0000	on page 1686
0x0048	STCU MBIST End Flag High Register (STCU_MBEH)	R	0x0000_0000	on page 1689
0x004C	Reserved			
0x0050	STCU MBIST Critical FM Low Register (STCU_MBCFML)	R	0x0000_0000	on page 1689
0x0054	STCU MBIST Critical FM High Register (STCU_MBCFMH)	R	0x0000_0000	on page 1690
0x0058	STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)	R	0x0000_0000	on page 1691
0x005C	STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)	R	0x0000_0000	on page 1691
0x0060–0x007F	Reserved			

Table 50-1. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x0080	STCU LBIST Control Register 0 (STCU_LB_CTRL0)	R	0x0000_0000	on page 1692
0x0084	Reserved			
0x0088	STCU LBIST MISR Expected Low Register 0 (STCU_LB_MISREL0)	R	0xFFFF_FFFF	on page 1693
0x008C	STCU LBIST MISR Expected High Register 0 (STCU_LB_MISREH0)	R	0xFFFF_FFFF	on page 1694
0x0090	STCU LBIST MISR Read Low Register 0 (STCU_LB_MISRRL0)	R	0x0000_0000	on page 1694
0x0094	STCU LBIST MISR Read High Register 0 (STCU_LB_MISRRH0)	R	0x0000_0000	on page 1695
0x0098–0x009F	Reserved			
0x00A0	STCU LBIST Control Register 1 (STCU_LB_CTRL1)	R	0x0000_0000	on page 1692
0x00A4	Reserved			
0x00A8	STCU LBIST MISR Expected Low Register (STCU_LB_MISREL1)	R	0xFFFF_FFFF	on page 1693
0x00AC	STCU LBIST MISR Expected High Register (STCU_LB_MISREH1)	R	0xFFFF_FFFF	on page 1694
0x00B0	STCU LBIST MISR Read Low Register (STCU_LB_MISRRL1)	R	0x0000_0000	on page 1694
0x00B4	STCU LBIST MISR Read High Register (STCU_LB_MISRRH1)	R	0x0000_0000	on page 1695
0x00B8–0x00BF	Reserved			
0x00C0	STCU LBIST Control Register 2 (STCU_LB_CTRL2)	R	0x0000_0000	on page 1692
0x00C4	Reserved			
0x00C8	STCU LBIST MISR Expected Low Register 2 (STCU_LB_MISREL2)	R	0xFFFF_FFFF	on page 1693
0x00CC	STCU LBIST MISR Expected High Register 2 (STCU_LB_MISREH2)	R	0xFFFF_FFFF	on page 1694
0x00D0	STCU LBIST MISR Read Low Register 2 (STCU_LB_MISRRL2)	R	0x0000_0000	on page 1694
0x00D4	STCU LBIST MISR Read High Register 2 (STCU_LB_MISRRH2)	R	0x0000_0000	on page 1695
0x00D8–0x02FF	Reserved			
0x0300	STCU MBIST Control Register 0 (STCU_MB_CTRL0)	R	0x0000_0000	on page 1695

Table 50-1. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x0304	STCU MBIST Control Register 1 (STCU_MB_CTRL1)	R	0x0000_0000	on page 1695
0x0308	STCU MBIST Control Register 2 (STCU_MB_CTRL2)	R	0x0000_0000	on page 1695
0x030C	STCU MBIST Control Register 3 (STCU_MB_CTRL3)	R	0x0000_0000	on page 1695
0x0310	STCU MBIST Control Register 4 (STCU_MB_CTRL4)	R	0x0000_0000	on page 1695
0x0314	STCU MBIST Control Register 5 (STCU_MB_CTRL5)	R	0x0000_0000	on page 1695
0x0318	STCU MBIST Control Register 6 (STCU_MB_CTRL6)	R	0x0000_0000	on page 1695
0x031C	STCU MBIST Control Register 7 (STCU_MB_CTRL7)	R	0x0000_0000	on page 1695
0x0320	STCU MBIST Control Register 8 (STCU_MB_CTRL8)	R	0x0000_0000	on page 1695
0x0324	STCU MBIST Control Register 9 (STCU_MB_CTRL9)	R	0x0000_0000	on page 1695
0x0328	STCU MBIST Control Register 10 (STCU_MB_CTRL10)	R	0x0000_0000	on page 1695
0x032C	STCU MBIST Control Register 11 (STCU_MB_CTRL11)	R	0x0000_0000	on page 1695
0x0330	STCU MBIST Control Register 12 (STCU_MB_CTRL12)	R	0x0000_0000	on page 1695
0x0334	STCU MBIST Control Register 13 (STCU_MB_CTRL13)	R	0x0000_0000	on page 1695
0x0338	STCU MBIST Control Register 14 (STCU_MB_CTRL14)	R	0x0000_0000	on page 1695
0x033C	STCU MBIST Control Register 15 (STCU_MB_CTRL15)	R	0x0000_0000	on page 1695
0x0340	STCU MBIST Control Register 16 (STCU_MB_CTRL16)	R	0x0000_0000	on page 1695
0x0344	STCU MBIST Control Register 17 (STCU_MB_CTRL17)	R	0x0000_0000	on page 1695
0x0348	STCU MBIST Control Register 18 (STCU_MB_CTRL18)	R	0x0000_0000	on page 1695
0x034C	STCU MBIST Control Register 19 (STCU_MB_CTRL19)	R	0x0000_0000	on page 1695

Table 50-1. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x0350	STCU MBIST Control Register 20 (STCU_MB_CTRL20)	R	0x0000_0000	on page 1695
0x0354	STCU MBIST Control Register 21 (STCU_MB_CTRL21)	R	0x0000_0000	on page 1695
0x0358	STCU MBIST Control Register 22 (STCU_MB_CTRL22)	R	0x0000_0000	on page 1695
0x035C	STCU MBIST Control Register 23 (STCU_MB_CTRL23)	R	0x0000_0000	on page 1695
0x0360	STCU MBIST Control Register 24 (STCU_MB_CTRL24)	R	0x0000_0000	on page 1695
0x0364	STCU MBIST Control Register 25 (STCU_MB_CTRL25)	R	0x0000_0000	on page 1695
0x0368	STCU MBIST Control Register 26 (STCU_MB_CTRL26)	R	0x0000_0000	on page 1695
0x036C	STCU MBIST Control Register 27 (STCU_MB_CTRL27)	R	0x0000_0000	on page 1695
0x0370	STCU MBIST Control Register 28 (STCU_MB_CTRL28)	R	0x0000_0000	on page 1695
0x0374	STCU MBIST Control Register 29 (STCU_MB_CTRL29)	R	0x0000_0000	on page 1695
0x0378	STCU MBIST Control Register 30 (STCU_MB_CTRL30)	R	0x0000_0000	on page 1695
0x037C	STCU MBIST Control Register 31 (STCU_MB_CTRL31)	R	0x0000_0000	on page 1695
0x0380	STCU MBIST Control Register 32 (STCU_MB_CTRL32)	R	0x0000_0000	on page 1695
0x0384	STCU MBIST Control Register 33 (STCU_MB_CTRL33)	R	0x0000_0000	on page 1695
0x0388	STCU MBIST Control Register 34 (STCU_MB_CTRL34)	R	0x0000_0000	on page 1695
0x038C	STCU MBIST Control Register 35 (STCU_MB_CTRL35)	R	0x0000_0000	on page 1695
0x0390	STCU MBIST Control Register 36 (STCU_MB_CTRL36)	R	0x0000_0000	on page 1695
0x0394	STCU MBIST Control Register 37 (STCU_MB_CTRL37)	R	0x0000_0000	on page 1695
0x0398	STCU MBIST Control Register 38 (STCU_MB_CTRL38)	R	0x0000_0000	on page 1695

Table 50-1. STCU register map (continued)

Offset from STCU_BASE (0xC3FF_4000)	Register name	Access[1]	Reset value[2]	Location
0x039C	STCU MBIST Control Register 39 (STCU_MB_CTRL39)	R	0x0000_0000	on page 1695
0x03A0	STCU MBIST Control Register 40 (STCU_MB_CTRL40)	R	0x0000_0000	on page 1695
0x03A4	STCU MBIST Control Register 41 (STCU_MB_CTRL41)	R	0x0000_0000	on page 1695
0x03A8	STCU MBIST Control Register 42 (STCU_MB_CTRL42)	R	0x0000_0000	on page 1695
0x03AC	STCU MBIST Control Register 43 (STCU_MB_CTRL43)	R	0x0000_0000	on page 1695
0x03B0	STCU MBIST Control Register 44 (STCU_MB_CTRL44)	R	0x0000_0000	on page 1695
0x03B4	STCU MBIST Control Register 45 (STCU_MB_CTRL45)	R	0x0000_0000	on page 1695
0x03B8	STCU MBIST Control Register 46 (STCU_MB_CTRL46)	R	0x0000_0000	on page 1695
0x03BC	STCU MBIST Control Register 47 (STCU_MB_CTRL47)	R	0x0000_0000	on page 1695
0x03C0	STCU MBIST Control Register 48 (STCU_MB_CTRL48)	R	0x0000_0000	on page 1695
0x03C4	STCU MBIST Control Register 49 (STCU_MB_CTRL49)	R	0x0000_0000	on page 1695
0x03C8	STCU MBIST Control Register 50 (STCU_MB_CTRL50)	R	0x0000_0000	on page 1695
0x03CC	STCU MBIST Control Register 51 (STCU_MB_CTRL51)	R	0x0000_0000	on page 1695
0x03D0	STCU MBIST Control Register 52 (STCU_MB_CTRL52)	R	0x0000_0000	on page 1695
0x03D4	STCU MBIST Control Register 53 (STCU_MB_CTRL53)	R	0x0000_0000	on page 1695
0x03D8	STCU MBIST Control Register 54 (STCU_MB_CTRL54)	R	0x0000_0000	on page 1695
0x03DC	STCU MBIST Control Register 55 (STCU_MB_CTRL55)	R	0x0000_0000	on page 1695
0x03E0	STCU MBIST Control Register 56 (STCU_MB_CTRL56)	R	0x0000_0000	on page 1695
0x03E4	STCU MBIST Control Register 57 (STCU_MB_CTRL57)	R	0x0000_0000	on page 1695
0x03E8–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

50.4.2 Register conventions

The following bus operations (contiguous byte enables) are supported:

- Word (32-bit) data read operations
- Low and high halfwords (16-bit, data[31:16] or data[15:0]) data read operations
- Byte (8-bit, data[31:24] or data[23:16] or data[15:8] or data[7:0]) data read operations

The STCU generates a bus transfer error in the following cases:

- A read access to the register addresses not mapped on the peripheral but included in the address space of the peripheral
- A read operation different from byte/halfword/word (free byte enables or other operations) on each register

The registers of the STCU are accessible (read-only) in each access mode: user or supervisor.

50.4.3 Detailed register descriptions

50.4.3.1 STCU Run Register (STCU_RUN)

The STCU Run Register (STCU_RUN) defines the RUN bit to start the self-test.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RUN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-5. STCU Run Register (STCU_RUN)

Table 50-2. STCU_RUN field descriptions

Field	Description
RUN	RUN. The STCU automatically clears the RUN bit when the self-testing procedure has been completed. 0 Idle. 1 Self-testing procedure is running.

50.4.3.2 STCU Configuration Register (STCU_CFG)

The STCU Configuration Register (STCU_CFG) includes the global configuration of the STCU.

Address: Base + 0x000C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	PTR							0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	FM	BYP	0	0	0	0	CLK_CFG			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-6. STCU Configuration Register (STCU_CFG)

Table 50-3. STCU_CFG field descriptions

Field	Description
PTR	LBIST or MBIST pointer. PTR defines the logical pointer to the first LBIST or MBIST to be scheduled. 0 to 2: pointer to LBIST. 0x10 to 0x49: pointer to MBIST. Value is calculated as [0x10 + (0x3A – 1)]. 0x3A = total number of MBISTs (58). 0x7F: pointer to NIL. No BIST execution.
FM	FCCU masking. 0 The critical fault (CF) and non-critical fault (NCF) lines are not masked. 1 The CF and NCF lines are masked.
BYP	Bypass mode. 0 STCU is active. 1 STCU is in bypass mode. The self-test is bypassed.
CLK_CFG	Logic, Memory BIST and STCU core CLK Clock configuration. CLK_CFG defines the ratio between the SYS_CLK and the clock used to program the LBIST, MBIST, and STCU core clock. The allowed configurations are the following: 0000 SYS_CLK 0001 SYS_CLK/2 0010 SYS_CLK/3 . . . 1101 SYS_CLK/14 1110 SYS_CLK/15 1111 SYS_CLK/16

50.4.3.3 STCU Watchdog Register Granularity (STCU_WDGG)

The STCU Watchdog Register Granularity (STCU_WDGG) defines the granularity of the LBIST and MBIST watchdog timer that provides protection against dead-lock or runaway conditions during the self-test.

When the self-test is not run but the STCU is still activated (the STCU_CFG[BYP] bit is not set), the bits 16:31 define the timeout before the STCU_ERR[WDTO] bit is set and the STCU core clock is switched off.

Address: Base + 0x0010 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GLBIST													GMBIST		
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 50-7. STCU Watchdog Register Granularity (STCU_WDGG)

Table 50-4. STCU_WDGG field descriptions

Field	Description
GLBIST	Granularity of the LBIST. The value of this field has to be evaluated taking into account the following: • The maximum scan chain length of the LBIST • The related maximum shift-speed parameter defined in the STCU_LB_CTRL[SHS] field • The STCU clock frequency/prescaling factor defined in the STCU_CFG[CLK_CFG] field The following equation shows how to set this value: $GLBIST_{value} = MAX(LBIST_{Chain}) \times MAX(LBIST_{SHS})$
GMBIST	Granularity of the MBIST. The value of this field has to be evaluated in order to define the granularity of the MBIST run taking into account the following: • The watchdog timer operates at the STCU clock frequency prescaled depending on the parameter CLK_CFG defined in the STCU_CFG register • The minimum value is the length of a fixed prescaler (10 bits) The following equation gives the granularity of MBIST in system clock cycles: $GMBIST_{cycles} = 2^{GMBIST} \times 1024$ The following shows the granularity: 000 1 K STCU clock cycles 001 2 K STCU clock cycles . . . 111 128 K STCU clock cycles

50.4.3.4 STCU CRC Expected Status Register (STCU_CRCE)

The STCU CRC Expected Status Register (STCU_CRCE) holds the expected signature of the CRC-32 loaded by the SSCM. The Self Checker computes the CRC signature of the STCU's critical signals. If the CRC computation does not match the expected value, the STCU_ERR[CRCS] bit is set.

Address: Base + 0x0014 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CRCE[31:16]															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CRCE[15:0]															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 50-8. STCU CRC Expected Status Register (STCU_CRCE)

Table 50-5. STCU_CRCE field descriptions

Field	Description
CRCE	CRC expected signature

50.4.3.5 STCU CRC Read Status Register (STCU_CRCR)

The STCU CRC Read Status Register (STCU_CRCR) reports the value obtained by the Self Checker at the end of the self-test. It can be used for diagnostics and as an additional check with respect to the STCU_ERR[CRCS] bit.

Address: Base + 0x0018 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CRCR[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CRCR[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-9. STCU CRC Read Status Register (STCU_CRCR)

Table 50-6. STCU_CRCR field descriptions

Field	Description
CRCR	Read CRC signature

50.4.3.6 STCU Error Register (STCU_ERR)

The STCU Error Register (STCU_ERR) includes the status flags related to the STCU internal error conditions that occurred during the configuration or the self-testing execution and the related Fault Masking (Stay-In-Reset (SIR) and Critical) control bits.

The STCU_ERR bits can be cleared or set by software depending on the applied unlocked sequence:

1. Write Key*x* into the STCU_ERR.
2. Set/clear the STCU_ERR register.

where

- Key1 allows the clearing of the bits at 1 by programming those bits to 0.
- Key2 allows the setting of the bits at 0 by programming those bits to 1.

In case of invalid access (wrong or missing key), a transfer error on the IPS or SSCM bus is asserted (depending on the selected bus interface) and the write operation on STCU_ERR is ignored.

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	WDTOSFM	CRCSSFM	ENGESFM	INVPSFM	0	0	0	0	WDTOCFM	CRCSCFM	ENGECFM	INVPCFM
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	CFSF	NCFSF	SIRSF	0	0	0	0	WDTO	CRCS	ENGE	INVP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-10. STCU Error Register (STCU_ERR)

Table 50-7. STCU_ERR field descriptions

Field	Description
WDTOSFM	Watchdog timeout SIR fault mapping 0 Non SIR fault mapping. 1 SIR fault mapping.
CRCSSFM	CRC SIR fault napping 0 Non SIR fault mapping. 1 SIR fault mapping.
ENGESFM	Engine error SIR fault mapping 0 Non SIR fault mapping. 1 SIR fault mapping.
INVPSFM	Invalid pointer SIR fault mapping 0 Non SIR fault mapping. 1 SIR fault mapping.
WDTOCFM	Watchdog timeout critical fault mapping 0 Non critical fault mapping. 1 Critical fault mapping.
CRCSCFM	CRC status critical fault mapping 0 Non critical fault mapping. 1 Critical fault mapping.

Table 50-7. STCU_ERR field descriptions (continued)

Field	Description
ENGECFM	Engine error critical fault mapping 0 Non critical fault mapping. 1 Critical fault mapping.
INVPCFM	Invalid pointer critical fault mapping 0 Non critical fault mapping. 1 Critical fault mapping.
CFSF	Critical faults status flag This flag reports the global status of the CF. 0 No errors that trigger the CF condition occurred. 1 At least one error that triggers the CF condition occurred.
NCFSF	Non critical faults status flag 0 No errors that trigger the NCF condition occurred. 1 At least one error that triggers the NCF condition occurred.
SIRSF	Stay-In-Reset faults status flag 0 No errors have triggered the SIR condition. 1 One or more errors have triggered the SIR condition. In the typical condition, it should not be possible to read the content of this register when this bit is set because the system should be permanently under reset. However, for diagnostic purposes, the system could exit from reset to allow software to check the flag and attempt to trace the failure.
WDTO	Watchdog timeout 0 The self-test completed within the assigned watchdog time. 1 The self-test did not complete within the assigned watchdog time. This bit is also set when the STCU is activated but the self-test is not run.
CRCS	CRC status 0 Successful CRC comparison. 1 Failed CRC comparison.
ENGE	Engine error 0 Valid engine execution. 1 Invalid engine execution.
INVP	Invalid pointer. The following conditions set this bit: • Initial LBIST or MBIST pointer is out of range. • LBIST is selected when MBIST is concurrently running or vice versa. • Error exists in the LBIST/MBIST linking (execution generates an infinite loop). 0 Valid linked pointer list. 1 Invalid linked pointer list.

50.4.3.7 STCU LBIST Status Register (STCU_LBS)

The STCU LBIST Status Register (STCU_LBS) includes the results corresponding to the execution of each LBIST. The STCU_LBS register is automatically set following the completion of the LBIST run. The flags are cleared by hardware upon next reset which is able to restart BIST.

Address: Base + 0x0024 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBS2	LBS1	LBS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-11. STCU LBIST Status Register (STCU_LBS)

Table 50-8. STCU_LBS field descriptions

Field	Description
LBS*x*	LBIST status 0 Failed LBIST execution. 1 Successful LBIST execution.

50.4.3.8 STCU LBIST End Flag Register (STCU_LBE)

The STCU LBIST End Flag Register (STCU_LBE) includes the End Flag related to the execution of each LBIST. The STCU_LBE register is automatically updated following the completion of the LBIST run.

Address: Base + 0x0028 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBE2	LBE1	LBE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-12. STCU LBIST End Flag Register (STCU_LBE)

Table 50-9. STCU_LBE field descriptions

Field	Description
LBE*x*	LBIST End status 0 LBIST execution is not finished. 1 LBIST execution is finished.

50.4.3.9 STCU LBIST Critical FM Register (STCU_LBCFM)

The STCU LBIST Critical FM Register (STCU_LBCFM) defines the fault mapping of each LBIST in terms of critical or non-critical faults.

Address: Base + 0x0030 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBCFM2	LBCFM1	LBCFM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-13. STCU LBIST Critical FM Register (STCU_LBCFM)

Table 50-10. STCU_LBCFM field descriptions

Field	Description
LBCFM*x*	LBIST Critical Fault Mapping 0 This LBIST is a non-critical fault (NCF). 1 This LBIST is a critical fault (CF).

50.4.3.10 STCU LBIST Stay-In-Reset FM Register (STCU_LBSFM)

The STCU LBIST Stay-In-Reset FM Register (STCU_LBSFM) defines the fault mapping of each LBIST in terms of the SIR condition.

Address: Base + 0x0034 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	LBSFM2	LBSFM1	LBSFM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-14. STCU LBIST Stay-In-Reset FM Register (STCU_LBSFM)

Table 50-11. STCU_LBSFM field descriptions

Field	Description
LBSFM*x*	LBIST SIR Fault Mapping 0 This LBIST is not a Stay-In-Reset (SIR) fault. 1 This LBIST is a Stay-In-Reset (SIR) fault.

50.4.3.11 STCU MBIST Status Low Register (STCU_MBSL)

The STCU MBIST Status Low Register (STCU_MBSL) includes the results corresponding to the execution of each MBIST. The STCU_MBSL register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags. The flags are cleared by hardware upon next reset which is able to restart BIST.

Address: Base + 0x003C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBS31	MBS30	MBS29	MBS28	MBS27	MBS26	MBS25	MBS24	MBS23	MBS22	MBS21	MBS20	MBS19	MBS18	MBS17	MBS16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBS15	MBS14	MBS13	MBS12	MBS11	MBS10	MBS9	MBS8	MBS7	MBS6	MBS5	MBS4	MBS3	MBS2	MBS1	MBS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-15. STCU MBIST Status Low Register (STCU_MBSL)

Table 50-12. STCU_MBSL field descriptions

Field	Description
MBS*x*	MBIST status 0 Failed MBIST execution. 1 Successful MBIST execution.

50.4.3.12 STCU MBIST Status High Register (STCU_MBSH)

The STCU MBIST Status High Register (STCU_MBSH) includes the results corresponding to the execution of each MBIST. The STCU_MBSH register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags. The flags are cleared by hardware upon next reset which is able to restart BIST.

Address: Base + 0x0040 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	MBS57	MBS56	MBS55	MBS54	MBS53	MBS52	MBS51	MBS50	MBS49	MBS48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBS47	MBS46	MBS45	MBS44	MBS43	MBS42	MBS41	MBS40	MBS39	MBS38	MBS37	MBS36	MBS35	MBS34	MBS33	MBS32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-16. STCU MBIST Status High Register (STCU_MBSH)

Table 50-13. STCU_MBSH field descriptions

Field	Description
MBS*x*	MBIST status 0 Failed MBIST execution. 1 Successful MBIST execution.

50.4.3.13 STCU MBIST End Flag Low Register (STCU_MBEL)

The STCU MBIST End Flag Low Register (STCU_MBEL) includes the End Flag related to the execution of each MBIST. The STCU_MBEL register is automatically updated following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags.

Address: Base + 0x0044 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBE31	MBE30	MBE29	MBE28	MBE27	MBE26	MBE25	MBE24	MBE23	MBE22	MBE21	MBE20	MBE19	MBE18	MBE17	MBE16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBE15	MBE14	MBE13	MBE12	MBE11	MBE10	MBE9	MBE8	MBE7	MBE6	MBE5	MBE4	MBE3	MBE2	MBE1	MBE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-17. STCU MBIST End Flag Low Register (STCU_MBEL)

Table 50-14. STCU_MBEL field descriptions

Field	Description
MBE*x*	MBIST End status 0 MBIST execution is not finished. 1 MBIST execution is finished.

Table 50-15. MBIST fault flags

Function	MBIST	Address (LSM)	Address (DPM)
Cache (Core_0)—Instruction Tag	0 (a,b,c,d)	—	—
Cache (Core_0)—Data Tag	1 (a,b,c,d)	—	—
Cache (Core_0)—Instruction	2, 3	—	—
	4, 5	—	—
Cache (Core_0)—Data	6, 7	—	—
	8, 9	—	—
Cache (Core_0)—Dirty	10	—	—
SRAM (Core_0)	11, 12	11 – 0x4000_0000 to 0x4000_7FFF 12 – 0x4000_8000 to 0x4000_FFFF	11 – 0x4000_0000 to 0x4000_7FFF 12 – 0x4000_8000 to 0x4000_FFFF
	13, 14	13 – 0x4001_0000 to 0x4001_7FFF 14 – 0x4001_8000 to 0x4001_FFFF	13 – 0x4001_0000 to 0x4001_7FFF 14 – 0x4001_8000 to 0x4001_FFFF
	15, 16	15 – 0x4002_0000 to 0x4002_7FFF 16 – 0x4002_8000 to 0x4002_FFFF	15 – 0x4002_0000 to 0x4002_7FFF 16 – 0x4002_8000 to 0x4002_FFFF
	17, 18	17 – 0x4003_0000 to 0x4003_7FFF 18 – 0x4003_8000 to 0x4003_FFFF	17 – 0x4003_0000 to 0x4003_7FFF 18 – 0x4003_8000 to 0x4003_FFFF

Table 50-15. MBIST fault flags (continued)

Function	MBIST	Address (LSM)	Address (DPM)
SRAM (Core_1)	19, 20	19 – 0x4004_0000 to 0x4004_7FFF 20 – 0x4004_8000 to 0x4004_FFFF	19 – 0x5000_0000 to 0x5000_7FFF 20 – 0x5000_8000 to 0x5000_FFFF
	21, 22	21 – 0x4005_0000 to 0x4005_7FFF 22 – 0x4005_8000 to 0x4005_FFFF	21 – 0x5001_0000 to 0x5001_7FFF 22 – 0x5001_8000 to 0x5001_FFFF
	23, 24	23 – 0x4006_0000 to 0x4006_7FFF 24 – 0x4006_8000 to 0x4006_FFFF	23 – 0x5002_0000 to 0x5002_7FFF 24 – 0x5002_8000 to 0x5002_FFFF
	25, 26	25 – 0x4007_0000 to 0x4007_7FFF 26 – 0x4007_8000 to 0x4007_FFFF	25 – 0x5003_0000 to 0x5003_7FFF 26 – 0x5003_8000 to 0x5003_FFFF
Cache (Core_1)—Instruction Tag	27 (a,b,c,d)	—	—
Cache (Core_1)—Data Tag	28 (a,b,c,d)	—	—
Cache (Core_1)—Instruction	29, 30	—	—
	31, 32	—	—
Cache (Core_1)—Data	33, 34	—	—
	35, 36	—	—
Cache (Core_1)—Dirty	37	—	—
DMA (0,1)	38, 39	—	—
CAN_MB (0)	40	—	—
CAN_MB (1,2,3)	41, 42, 43	—	—
CAN_RXIM (0)	44	—	—
CAN_RXIM (1,2,3)	45, 46, 47	—	—
FLEXRAY_LRAM (LUT)	48, 49	—	—
FLEXRAY_DRAM (DATA)	50	—	—
FEC (MIB)	51	—	—
FEC (RIF)	52	—	—
Flash memory	53	—	—
	54	—	—
	55	—	—
FlexRay ROM	56	—	—
BAM ROM	57	—	—

50.4.3.14 STCU MBIST End Flag High Register (STCU_MBEH)

The STCU MBIST End Flag High Register (STCU_MBEH) includes the End Flag related to the execution of each MBIST. The STCU_MBEH register is automatically updated following the completion of the MBIST run.

Address: Base + 0x0048 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	MBE57	MBE56	MBE55	MBE54	MBE53	MBE52	MBE51	MBE50	MBE49	MBE48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBE47	MBE46	MBE45	MBE44	MBE43	MBE42	MBE41	MBE40	MBE39	MBE38	MBE37	MBE36	MBE35	MBE34	MBE33	MBE32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-18. STCU MBIST End Flag High Register (STCU_MBEH)

Table 50-16. STCU_MBEH field descriptions

Field	Description
MBE*x*	MBIST End status 0 MBIST execution is not finished. 1 MBIST execution is finished.

50.4.3.15 STCU MBIST Critical FM Low Register (STCU_MBCFML)

The STCU MBIST Critical FM Low Register (STCU_MBCFML) defines the MBIST fault mapping in terms of critical or non-critical faults. The STCU_MBCFML register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags.

Address: Base + 0x0050 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBCFM31	MBCFM30	MBCFM29	MBCFM28	MBCFM27	MBCFM26	MBCFM25	MBCFM24	MBCFM23	MBCFM22	MBCFM21	MBCFM20	MBCFM19	MBCFM18	MBCFM17	MBCFM16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBCFM15	MBCFM14	MBCFM13	MBCFM12	MBCFM11	MBCFM10	MBCFM9	MBCFM8	MBCFM7	MBCFM6	MBCFM5	MBCFM4	MBCFM3	MBCFM2	MBCFM1	MBCFM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-19. STCU MBIST Critical FM Low Register (STCU_MBCFML)

Table 50-17. STCU_MBCFML field descriptions

Field	Description
MBCFM*x*	MBIST Critical Fault Mapping 0 This MBIST is a non-critical fault (NCF). 1 This MBIST is a critical fault (CF).

50.4.3.16 STCU MBIST Critical FM High Register (STCU_MBCFMH)

The STCU MBIST Critical FM High Register (STCU_MBCFMH) defines the MBIST fault mapping in terms of critical or non-critical faults. The STCU_MBCFMH register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags.

Address: Base + 0x0054 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	MBCFM57	MBCFM56	MBCFM55	MBCFM54	MBCFM53	MBCFM52	MBCFM51	MBCFM50	MBCFM49	MBCFM48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBCFM47	MBCFM46	MBCFM45	MBCFM44	MBCFM43	MBCFM42	MBCFM41	MBCFM40	MBCFM39	MBCFM83	MBCFM37	MBCFM36	MBCFM35	MBCFM34	MBCFM33	MBCFM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-20. STCU MBIST Critical FM High Register (STCU_MBCFMH)

Table 50-18. STCU_MBCFMH field descriptions

Field	Description
MBCFM*x*	MBIST Critical Fault Mapping 0 This MBIST is a non-critical fault (NCF). 1 This MBIST is a critical fault (CF).

50.4.3.17 STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)

The STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML) defines the MBIST fault mapping in terms of the SIR condition. The STCU_MBSFML register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags.

Address: Base + 0x0058 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MBSFM31	MBSFM30	MBSFM29	MBSFM28	MBSFM27	MBSFM26	MBSFM25	MBSFM24	MBSFM23	MBSFM22	MBSFM21	MBSFM20	MBSFM19	MBSFM18	MBSFM17	MBSFM16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBSFM15	MBSFM14	MBSFM13	MBSFM12	MBSFM11	MBSFM10	MBSFM9	MBSFM8	MBSFM7	MBSFM6	MBSFM5	MBSFM4	MBSFM3	MBSFM2	MBSFM1	MBSFM0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-21. STCU MBIST Stay-In-Reset FM Low Register (STCU_MBSFML)

Table 50-19. STCU_MBSFML field descriptions

Field	Description
MBSFM*x*	MBIST SIR Fault Mapping 0 This MBIST is not a Stay-In-Reset (SIR) fault. 1 This MBIST is a Stay-In-Reset (SIR) fault.

50.4.3.18 STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)

The STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH) defines the MBIST fault mapping in terms of the SIR condition. The STCU_MBSFMH register is automatically set following the completion of the MBIST run. Table 50-15 lists the MBIST fault flags.

Address: Base + 0x005C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	MBSFM57	MBSFM56	MBSFM55	MBSFM54	MBSFM53	MBSFM52	MBSFM51	MBSFM50	MBSFM49	MBSFM48
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MBSFM47	MBSFM46	MBSFM45	MBSFM44	MBSFM43	MBSFM42	MBSFM41	MBSFM40	MBSFM39	MBSFM83	MBSFM37	MBSFM36	MBSFM35	MBSFM34	MBSFM33	MBSFM32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-22. STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)

Table 50-20. STCU_MBSFMH field descriptions

Field	Description
MBSFM*x*	MBIST SIR Fault Mapping 0 This MBIST is not a Stay-In-Reset (SIR) fault. 1 This MBIST is a Stay-In-Reset (SIR) fault.

50.4.3.19 STCU LBIST Control Register *n* (STCU_LB_CTRL*n*)

The STCU_LB_CTRL*n* registers define the control fields of each LBIST.

Address: Base + 0x0080 (STCU_LB_CTRL0)
Base + 0x00A0 (STCU_LB_CTRL1)
Base + 0x00C0 (STCU_LB_CTRL2) Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSM	PTR							0	0	0	0	0	SHS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SCEN_OFF				SCEN_ON				0	0	0	0	0	CWS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-23. STCU LBIST Control Register *n* (STCU_LB_CTRL*n*)

Table 50-21. STCU_LB_CTRL*n* field descriptions

Field	Description
CSM	Concurrent/sequential mode. 0 Sequential mode. 1 Concurrent mode.
PTR	Next LBIST or MBIST pointer. PTR defines the logical pointer to the first LBIST or MBIST to be scheduled. 0 to 2: pointer to LBIST. 0x10 to 0x49: pointer to MBIST. Value is calculated as [0x10 + (0x3A – 1)]. 0x3A = total number of MBISTs (58). 0x7F: pointer to NIL. No BIST execution. Other values: invalid pointer → an error is set into the STCU_ERR register.
SHS	Shift speed. SHS defines the shift speed. 000 Shift at full rate (bist_clk) 001 Shift at 1/2 rate (bist_clk) 010 Shift at 1/3 rate (bist_clk) 011 Shift at 1/4 rate (bist_clk) 100 Shift at 1/5 rate (bist_clk) 101 Shift at 1/6 rate (bist_clk) 110 Shift at 1/7 rate (bist_clk) 111 Shift at 1/8 rate (bist_clk) **Note:** This field has no effect on this device. Software can write and read the specified values, but no functional changes will occur.
SCEN_OFF	Scan enable off. SCEN_OFF defines the number of clock cycles OFF following the falling transition on the SCEN. 0000 0 delay cycles 0001 1 delay cycle . . . 1111 15 delay cycles
SCEN_ON	Scan enable on. SCEN_ON defines the number of clock cycles OFF following the rising transition on the SCEN. 0000 0 delay cycles 0001 1 delay cycle . . . 1111 15 delay cycles
CWS	Capture window size. CWS defines the capture window size. 000 Illegal 001 Wait 1 shift cycle for capture to finish. 010 Wait 2 shift cycles for capture to finish. . . . 111 Wait 7 shift cycles for capture to finish.

50.4.3.20 STCU LBIST MISR Expected Low Register *n* (STCU_LB_MISREL*n*)

The STCU_LB_MISREL*n* registers define the LSB part of the expected MISR of each LBIST.

Address: Base + 0x0088 (STCU_LB_MISREL0)
Base + 0x00A8 (STCU_LB_MISREL1)
Base + 0x00C8 (STCU_LB_MISREL2)
Access: User read-only

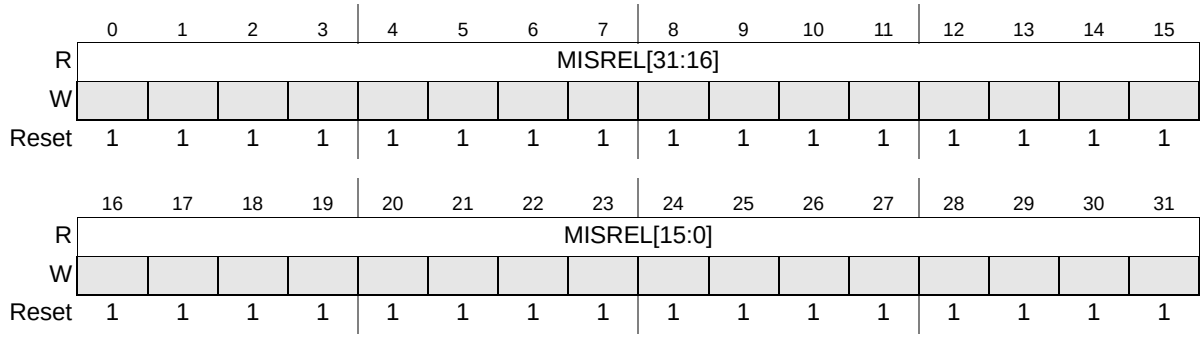


Figure 50-24. STCU LBIST MISR Expected Low Register *n* (STCU_LB_MISREL*n*)

Table 50-22. STCU_LB_MISREL*n* field descriptions

Field	Description
MISREL	MISR Expected low part. Defines the low word of the Expected MISR.

50.4.3.21 STCU LBIST MISR Expected High Register *n* (STCU_LB_MISREH*n*)

The STCU_LB_MISREH*n* registers define the MSB part of the expected MISR of each LBIST.

Address: Base + 0x008C (STCU_LB_MISREH0)
Base + 0x00AC (STCU_LB_MISREH1)
Base + 0x00CC (STCU_LB_MISREH2)
Access: User read-only

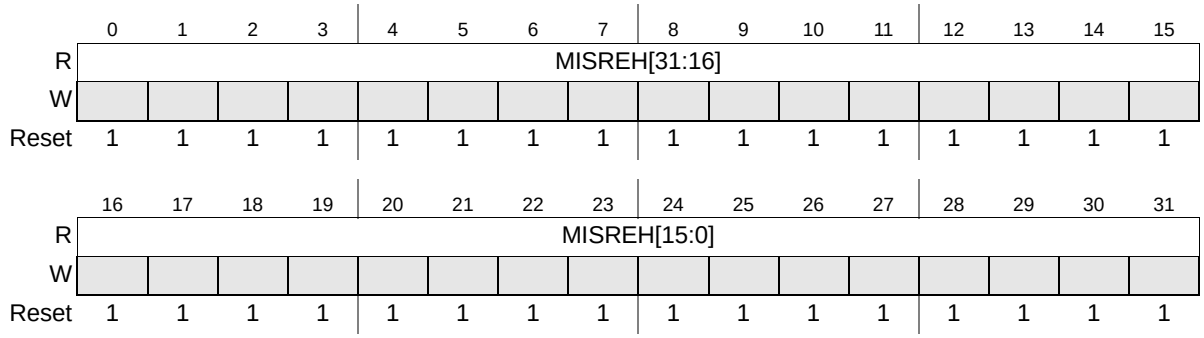


Figure 50-25. STCU LBIST MISR Expected High Register *n* (STCU_LB_MISREH*n*)

Table 50-23. STCU_LB_MISREH*n* field descriptions

Field	Description
MISREH	MISR Expected high part. Defines the high word of the Expected MISR.

50.4.3.22 STCU LBIST MISR Read Low Register *n* (STCU_LB_MISRRL*n*)

The STCU_LB_MISRRL*n* registers report the LSB part of the MISR obtained at the end of each LBIST.

Address: Base + 0x0090 (STCU_LB_MISRRL0)
Base + 0x00B0 (STCU_LB_MISRRL1)
Base + 0x00D0 (STCU_LB_MISRRL2)
Access: User read-only

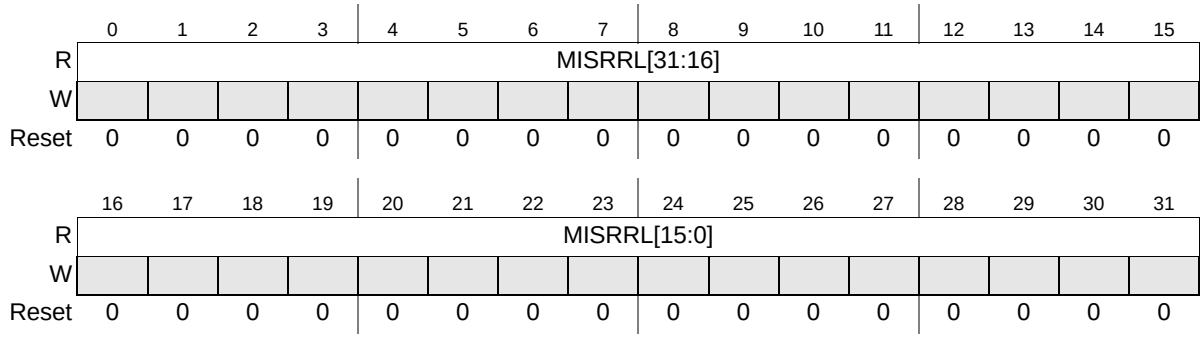


Figure 50-26. STCU LBIST MISR Read Low Register *n* (STCU_LB_MISRRL*n*)

Table 50-24. STCU_LB_MISRRL*n* field descriptions

Field	Description
MISRRL	MISR Read low part. Contains the low word of the MISR obtained at the end of the LBIST.

50.4.3.23 STCU LBIST MISR Read High Register *n* (STCU_LB_MISRRH*n*)

The STCU_LB_MISRRH*n* registers report the MSB part of the MISR obtained at the end of each LBIST.

Address: Base + 0x0094 (STCU_LB_MISRRH0)
Base + 0x00B4 (STCU_LB_MISRRH1)
Base + 0x00D4 (STCU_LB_MISRRH2)
Access: User read-only

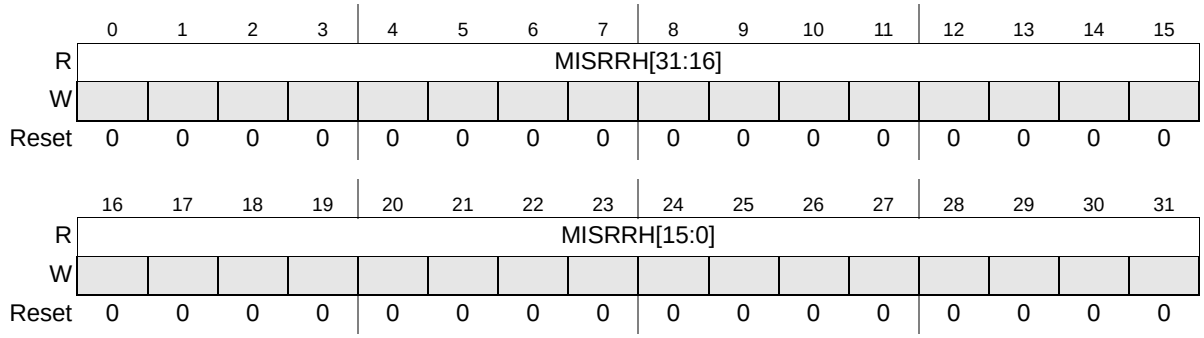


Figure 50-27. STCU LBIST MISR Read High Register *n* (STCU_LB_MISRRH*n*)

Table 50-25. STCU_LB_MISRRH field descriptions

Field	Description
MISRRH	MISR Read high part. Contains the high word of the MISR obtained at the end of the LBIST.

50.4.3.24 STCU MBIST Control Register *n* (STCU_MB_CTRL*n*)

The STCU_MB_CTRL*n* registers define the control fields of each MBIST.

Address: Base + 0x0300 + (*k* × 0x04) [1] Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSM	PTR							0	0	MB_TIME					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 50-28. STCU MBIST Control Register *n* (STCU_MB_CTRL*n*)

[1] The *k* variable represents the repeated register blocks of the multiple MBISTs: *k* ranges from 0 up to 57. See Table 50-1 for a complete list.

Table 50-26. STCU_MB_CTRL*n* field descriptions

Field	Description
CSM	Concurrent/sequential mode. 0 Sequential mode. 1 Concurrent mode.
PTR	Next LBIST or MBIST pointer. PTR defines the logical pointer to the next LBIST or MBIST to be scheduled. When the NIL pointer is encountered, the self-testing procedure is stopped when the current MBIST has been completed. 0 to 2: pointer to LBIST. 0x10 to 0x49: pointer to MBIST. Value is calculated as [0x10 + (0x3A – 1)]. 0x3A = total number of MBISTs (58). 0x7F: pointer to NIL. No BIST execution. Other valuers: invalid pointer → an error is set into the STCU_ERR register.
MB_TIME	Memory BIST Runtime. The time budget of the MBIST is evaluated applying the following equation: $MB_{cycles} = MBTIME \times GMBIST_{cycles}$ If the MBIST is not completed within the MB_TIME, the STCU_ERR[WDTO] bit is set.

50.5 LBIST partitioning

The LBIST partitioning scheme on this device is shown in Table 50-27.

Table 50-27. IP to LBIST partition and lake allocation

Assigned to lake #	LBIST partition	IP module	Remark
0	0	Core_0	Includes Nexus, MMU, Cache Controller, Cache memory
0	0	XBAR_0	—
0	0	MPU_0	—
0	0	PBRIDGE_0	—

Table 50-27. IP to LBIST partition and lake allocation (continued)

Assigned to lake #	LBIST partition	IP module	Remark
0	0	PBRIDGE_0 On-Platform Read Mux	—
0	0	PFLASHC_0	—
0	0	SRAMC_0	—
0	0	STM_0	—
0	0	SWT_0	—
0	0	INTC_0	—
0	0	ECSM_0	—
0	0	SEMA4_0	—
0	0	Cache Parity Propagation_0	—
0	0	SRAM Array (lower half)	—
0	0	SRAMC_0 Read Mux	—
0	0	NXSS_0	—
0	No LBIST partition	LBIST control [0]	—
0		MC_CGM [0]	—
1	1	Core_1	Includes Nexus, MMU, Cache Controller, Cache memory
1	1	XBAR_1	—
1	1	MPU_1	—
1	1	PBRIDGE_1	—
1	1	PBRIDGE_1 on-platform read mux	—
1	1	PFLASHC_1	—
1	1	SRAMC_1	—
1	1	STM_1	—
1	1	SWT_1	—
1	1	INTC_1	—
1	1	ECSM_1	—
1	1	SEMA4_1	—
1	1	Cache Parity Propagation_1	—
1	1	SRAM Array (upper half)	—
1	1	SRAMC_1 Read Mux	—
1	1	NXSS_1	—

Table 50-27. IP to LBIST partition and lake allocation (continued)

Assigned to lake #	LBIST partition	IP module	Remark
1	No LBIST Partition	LBIST Control [1]	—
1		MC_CGM [1]	—
SoG	2	SSCM	Parts of the module not LBISTed as they are needed during LBIST execution.
SoG	2	PMU	Parts of the module not LBISTed as they are needed during LBIST execution.
SoG	2	MC_PCU	Parts of the module not LBISTed as they are needed during LBIST execution.
SoG	2	MC_CGM	Parts of the module not LBISTed as they are needed during LBIST execution.
SoG	2	SIUL	—
SoG	2	WakeUp	—
SoG	2	IOMUX	—
SoG	2	PTI	—
SoG	2	MC_ME	—
SoG	2	NPC	—
SoG	2	PIT	—
SoG	2	BAM IPI	IPS Interface (not the ROM itself)
SoG	2	LINFlex_*x*	—
SoG	2	FlexCAN_*x*	—
SoG	2	DSPI_*x*	—
SoG	2	FlexRay	—
SoG	2	eTimer_*x*	—
SoG	2	CTU_x	—
SoG	2	FlexPWM_*x*	—
SoG	2	ADC_*x*	—
SoG	2	IIC_*x*	—
SoG	2	EBI	—
SoG	2	PDI	—
SoG	2	Ethernet (FEC)	—
SoG	2	DRAMC	—
SoG	2	FCCU	—
SoG	2	MACC	—
SoG	2	RCCU[*x*:0]_0[1]	—

Table 50-27. IP to LBIST partition and lake allocation (continued)

Assigned to lake #	LBIST partition	IP module	Remark
SoG	2	RCCU[*x*:0]_1[1]	—
SoG	2	Flash Module(s)	—
SoG	2	PBRIDGE_0/1 Read Mux Off-Platform	—
SoG	2	TSENS	—
SoG	2	DMA_*x*	—
SoG	2	DMACHMUX_*x*	—
SoG	No LBIST Partition	MC_RGM	—
		MC_RSL	—
		JTAGC	—
		STCU	—
		TCU	—
		PADS	—
		LBIST Control [4]	—
		MC_CGM [4]	—

1 Both sets of RCCUs (RCCU[6:0]_0 and RCCU[6:0]_1) are located in different physical regions of the die layout.

Chapter 51
System Timer Module (STM)

51.1 Introduction

51.1.1 Overview

The STM is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

51.1.2 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in Debug mode

51.1.3 Modes of operation

The STM supports two device modes of operation: Normal and Debug. When the STM is enabled in Normal mode, its counter runs continuously. In Debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR register. If the FRZ bit is set, the counter is stopped in Debug mode; otherwise it continues to run.

51.2 External signal description

The STM does not have any external interface signals.

51.3 Memory map and register description

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generate a bus error termination.

51.3.1 Memory map

Table 51-1. STM base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	STM_0	0xFFF3_C000
	STM_1	
Decoupled Parallel Mode (DPM)	STM_0	0xFFF3_C000 (same as LSM)
	STM_1	0x8FF3_C000

The STM memory map is shown in Table 51-2.

Table 51-2. STM memory map

Offset from STM_BASE	Register	Access [1]	Reset Value [2]	Location
0x0000	STM Control Register (STM_CR)	R/W	0x0000_0000	on page 1703
0x0004	STM Counter Value (STM_CNT)	R/W	0x0000_0000	on page 1703
0x0008–0x000C	Reserved			
0x0010	STM Channel 0 Control Register (STM_CCR0)	R/W	0x0000_0000	on page 1704
0x0014	STM Channel 0 Interrupt Register (STM_CIR0)	R/W	0x0000_0000	on page 1704
0x0018	STM Channel 0 Compare Register (STM_CMP0)	R/W	0x0000_0000	on page 1705
0x001C	Reserved			
0x0020	STM Channel 1 Control Register (STM_CCR1)	R/W	0x0000_0000	on page 1704
0x0024	STM Channel 1 Interrupt Register (STM_CIR1)	R/W	0x0000_0000	on page 1704
0x0028	STM Channel 1 Compare Register (STM_CMP1)	R/W	0x0000_0000	on page 1705
0x002C	Reserved			
0x0030	STM Channel 2 Control Register (STM_CCR2)	R/W	0x0000_0000	on page 1704
0x0034	STM Channel 2 Interrupt Register (STM_CIR2)	R/W	0x0000_0000	on page 1704
0x0038	STM Channel 2 Compare Register (STM_CMP2)	R/W	0x0000_0000	on page 1705
0x003C	Reserved			
0x0040	STM Channel 3 Control Register (STM_CCR3)	R/W	0x0000_0000	on page 1704
0x0044	STM Channel 3 Interrupt Register (STM_CIR3)	R/W	0x0000_0000	on page 1704
0x0048	STM Channel 3 Compare Register (STM_CMP3)	R/W	0x0000_0000	on page 1705
0x004C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

51.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

51.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control, and timer enable bits.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51-1. STM Control Register (STM_CR)

Table 51-3. STM_CR field descriptions

Field	Description
CPS	Counter Prescaler. Selects the clock divide value for the prescaler (1–256). 00 Divide system clock by 1 01 Divide system clock by 2 . . . FF Divide system clock by 256
FRZ	Freeze. Allows the timer counter to be stopped when the device enters Debug mode. 0 STM counter continues to run in Debug mode. 1 STM counter is stopped in Debug mode.
TEN	Timer Counter Enabled. 0 Counter is disabled. 1 Counter is enabled.

51.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51-2. STM Count Register (STM_CNT)

Table 51-4. STM_CNT field descriptions

Field	Description
CNT	Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value.

51.3.2.3 STM Channel Control Register *n* (STM_CCR*n*)

The STM Channel Control Register *n* (STM_CCR*n*) contains the enable bit for channel *n* of the timer.

Address: Base + 0x0010 (STM_CCR0) Base + 0x0030 (STM_CCR2)
Base + 0x0020 (STM_CCR1) Base + 0x0040 (STM_CCR3) Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51-3. STM Channel Control Register *n* (STM_CCR*n*)

Table 51-5. STM_CCR*n* field descriptions

Field	Description
CEN	Channel Enable. 0 The channel is disabled. 1 The channel is enabled.

51.3.2.4 STM Channel Interrupt Register *n* (STM_CIR*n*)

The STM Channel Interrupt Register *n* (STM_CIR*n*) has the interrupt flag for channel *n* of the timer.

Address: Base + 0x0014 (STM_CIR0) Base + 0x0034 (STM_CIR2)
Base + 0x0024 (STM_CIR1) Base + 0x0044 (STM_CIR3) Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51-4. STM Channel Interrupt Register *n* (STM_CIR*n*)

Table 51-6. STM_CIR*n* field descriptions

Field	Description
CIF	Channel Interrupt Flag 0 No interrupt request. 1 Interrupt request due to a match on the channel.

51.3.2.5 STM Channel Compare Register *n* (STM_CMP*n*)

The STM channel compare register *n* (STM_CMP*n*) holds the compare value for channel *n*.

Address: Base + 0x0014 (STM_CMP0) Base + 0x0034 (STM_CMP2)
Base + 0x0024 (STM_CMP1) Base + 0x0044 (STM_CMP3) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMP[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMP[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51-5. STM Channel Compare Register *n* (STM_CMP*n*)

Table 51-7. STM_CMP*n* field descriptions

Field	Description
CMP	Compare value for channel *n*. If the STM_CCR*n*[CEN] bit is set and the STM_CMP*n* register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIR*n*[CIF] bit is set.

51.4 Functional description

The STM is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in Normal mode, the counter continuously increments. When enabled in Debug mode, the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in Debug mode, otherwise it continues to run in Debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCR*n*), a channel interrupt register (STM_CIR*n*), and a channel compare register (STM_CMP*n*). The channel is enabled by setting the STM_CCR*n*[CEN] bit. When enabled, the channel sets the STM_CIR*n*[CIF] bit and generates an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM_CIR*n*[CIF] bit. A write of 0 to the STM_CIR*n*[CIF] bit has no effect.

NOTE

The STM counter does not advance when the system clock is stopped.

Chapter 52
Software Watchdog Timer (SWT)

52.1 Introduction

52.1.1 Overview

The SWT is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or a bus transaction failing to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing operation. The servicing operation resets the timer to a specified timeout period. If this servicing action does not occur before the timer expires, the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial timeout. A reset is always generated on a second consecutive timeout.

52.1.2 Features

The SWT has the following features:

- 32-bit timeout register to set the timeout period
- Programmable selection of system or oscillator clock for timer operation
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial timeout
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

52.1.3 Modes of operation

The SWT supports three device modes of operation: Normal, Debug, and Stop. When the SWT is enabled in Normal mode, its counter runs continuously. In Debug mode, operation of the counter is controlled by the FRZ bit in the SWT_CR. If the FRZ bit is set, the counter is stopped in Debug mode—otherwise, it continues to run. In Stop mode, operation of the counter is controlled by the STP bit in the SWT_CR. If the STP bit is set, the counter is stopped in Stop mode—otherwise, it continues to run.

52.2 External signal description

The SWT module does not have any external interface signals.

52.3 Memory map and register description

The SWT programming model has seven 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include:

- Writes to read-only registers
- Incorrect values written to the service register when enabled
- Accesses to reserved addresses
- Accesses by masters without permission.

If the RIA bit in the SWT_CR is set, the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. If either the HLK or SLK bits in the SWT_CR are set, then the SWT_CR, SWT_TO, SWT_WN, and SWT_SK registers are read-only.

52.3.1 Memory map

The SWT memory map is shown in Table 52-2.

Table 52-1. SWT module base addresses

Mode	Module	Module base address
Lock Step Mode (LSM)	SWT_0	0xFFF3_8000
	SWT_1	
Decoupled Parallel Mode (DPM)	SWT_0	0xFFF3_8000 (same as LSM)
	SWT_1	0x8FF3_8000

Table 52-2. SWT memory map

Offset from SWT_BASE	Register	Access[1]	Reset Value	Location
0x0000	SWT Control Register (SWT_CR)	R/W	0xFF00_011B	on page 1709
0x0004	SWT Interrupt Register (SWT_IR)	R/W	0x0000_0000	on page 1710
0x0008	SWT Timeout Register (SWT_TO)	R/W	0x0003_FDE0	on page 1710
0x000C	SWT Window Register (SWT_WN)	R/W	0x0000_0000	on page 1711
0x0010	SWT Service Register (SWT_SR)	W	0x0000_0000	on page 1711
0x0014	SWT Counter Output Register (SWT_CO)	R	0x0000_0000	on page 1712
0x0018	SWT Service Key Register (SWT_SK)	W	0x0000_0000	on page 1713
0x001C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

52.3.2 Register descriptions

The following sections detail the individual registers within the SWT programming model.

52.3.2.1 SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT. SWT_CR[WEN] remains set during single-chip boot. For serial boot, the BAM disables the SWT. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP0	MAP1	MAP2	MAP3	MAP4	MAP5	MAP6	MAP7	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1

Figure 52-1. SWT Control Register (SWT_CR)

Table 52-3. SWT_CR field descriptions

Field	Description
MAP*n*	Master Access Protection for XBAR Master *n*. 0 Access for the master is not enabled. 1 Access for the master is enabled.
KEY	Keyed Service Mode. 0 Fixed Service Sequence. The fixed sequence 0xC520 0xD928 is used to service the watchdog. 1 Keyed Service Mode. Two pseudorandom key values are used to service the watchdog.
RIA	Reset on Invalid Access. If the RIA bit is set, the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. 0 Invalid access to the SWT generates a bus error. 1 Invalid access to the SWT causes a system reset if WEN = 1.
WND	Window Mode. 0 Regular mode. Service sequence can be done at any time. 1 Windowed mode. The service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset. 0 Generate a reset on a timeout. 1 Generate an interrupt on an initial timeout, reset on a second consecutive timeout.
HLK	Hard Lock. This bit is only cleared at reset. 0 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read/write registers if SLK = 0. 1 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read-only registers.
SLK	Soft Lock. This bit is cleared by writing the unlock sequence to the service register. 0 SWT_CR, SWT_TO SWT_WN, and SWT_SK are read/write registers if HLK = 0. 1 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read-only registers.
CSL	Clock Selection. Selects the clock that drives the internal timer. 0 System clock. 1 Oscillator clock.

Table 52-3. SWT_CR field descriptions (continued)

Field	Description
STP	Stop Mode Control. Allows the watchdog timer to be stopped when the device enters Stop mode. 0 SWT counter continues to run in Stop mode. 1 SWT counter is stopped in Stop mode.
FRZ	Debug Mode Control. Allows the watchdog timer to be stopped when the device enters Debug mode. 0 SWT counter continues to run in Debug mode. 1 SWT counter is stopped in Debug mode.
WEN	Watchdog Enabled. 0 SWT is disabled. 1 SWT is enabled.

52.3.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the timeout interrupt flag.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52-2. SWT Interrupt Register (SWT_IR)

Table 52-4. SWT_IR field descriptions

Field	Description
TIF	Timeout Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to an initial timeout.

52.3.2.3 SWT Timeout Register (SWT_TO)

The SWT_TO register contains the 32-bit timeout period. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WTO[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WTO[15:0]															
Reset	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	0

Figure 52-3. SWT Timeout Register (SWT_TO)

Table 52-5. SWT_TO field descriptions

Field	Description
WTO	Watchdog timeout period in clock cycles. When the SWT is enabled or the service sequence is written, an internal 32-bit down counter is loaded with the value of WTO or 0x100, whichever is larger.

52.3.2.4 SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WST[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	WST[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52-4. SWT Window Register (SWT_WN)

Table 52-6. SWT_WN field descriptions

Field	Description
WST	Window start value. When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

52.3.2.5 SWT Service Register (SWT_SR)

The SWT timeout (SWT_SR) service register is the target for service operation writes used to reset the watchdog timer.

Address: Base + 0x0010 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	WSC															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52-5. SWT Service Register (SWT_SR)

Table 52-7. SWT_SR field descriptions

Field	Description
WSC	Watchdog Service Code.This field is used to service the watchdog and to clear the soft lock bit (SWT_CR[SLK]). If the SWT_CR[KEY] bit is set, two pseudorandom key values are written to service the watchdog (see Section 52.3.2.7, SWT Service Key Register (SWT_SK), for details). Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

52.3.2.6 SWT Counter Output Register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read-only register that shows the value of the internal down counter when the SWT is disabled.

Address: Base + 0x0014 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52-6. SWT Counter Output Register (SWT_CO)

Table 52-8. SWT_CO field descriptions

Field	Description
CNT	Watchdog Count. When the watchdog is disabled (SWT_CR[WEN] = 0), this field shows the value of the internal down counter. When the watchdog is enabled, the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for up to 6 system plus 8 counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

52.3.2.7 SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52-7. SWT Service Key Register (SWT_SK)

Table 52-9. SWT_SK field descriptions

Field	Description
SK	Service Key.This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17 × SK + 3) mod 2^{16}.

52.4 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or a bus transaction failing to terminate. It includes:

- A control register (SWT_CR)
- An interrupt register (SWT_IR)
- A timeout register (SWT_TO)
- A window register (SWT_WN)
- A service register (SWT_SR)
- A counter output register (SWT_CO)
- A service key register (SWT_SK)

The SWT_CR includes bits to enable the timer, set configuration options, and lock configuration of the module. The watchdog is enabled by setting the SWT_CR[WEN] bit. The reset value of the SWT_CR[WEN] bit is device-specific. If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT_TO register holds the watchdog timeout period in clock cycles unless the value is less than 0x100, in which case the timeout period is set to 0x100. This timeout period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_CR[CSL] bit selects which clock (system or oscillator) drives the down counter. The reset value of the SWT_TO register is 0x0003_FDE0.

The configuration of the SWT can be locked with either a soft or hard lock. In either case, when they are locked the SWT_CR, SWT_TO, SWT_WN, and SWT_SK registers are read-only. The hard lock is enabled by setting the SWT_CR[HLK] bit, which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation that consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the timeout period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_CR[KEY] bit is 0, the fixed sequence 0xC520 0xD928 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_CR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in Equation 52-1. This algorithm generates a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103 and 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499 and 0x7E2C.

Eqn. 52-1

$$SK_{n+1} = (17 \times SK_n + 3) \bmod 2^{16}$$

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If Window mode is enabled (SWT_CR[WND] = 1), the service sequence must be performed in the last part of the timeout period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset, depending on the value of the SWT_CR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000, then the service sequence must be performed in the last 20% of the timeout period. There is a short lag in the time it takes for the window to open, due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT_CR[ITR]) controls the action taken when a timeout occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a timeout. If the SWT_CR[ITR] bit is set, an initial timeout causes the SWT to generate an interrupt and load the down counter with the timeout period. If the service sequence is not written before the second consecutive timeout, the SWT generates a system reset. The interrupt is indicated by the timeout interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled, this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT_CO can be used during a software self-test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the timeout value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO can be read to determine if the internal down counter is working properly.

This page is intentionally left blank.

Chapter 53
Temperature Sensor (TSENS)

53.1 Introduction

The TSENS provides an analog voltage that is proportional to the internal temperature of the MPC5675K. This voltage can be sampled by the ADC and converted to an actual temperature by using factory-programmed calibration constants.

Figure 53-1 shows a block diagram of the TSENS. The module operates on the principles of a basic silicon bandgap sensor that measures the difference in base-emitter voltages of multiple transistors, which can be related to the absolute temperature of the device.

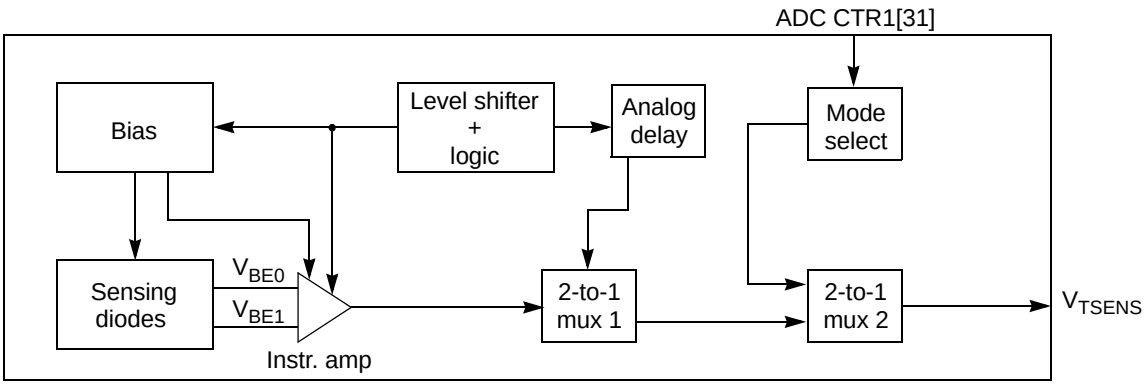


Figure 53-1. TSENS block diagram

53.2 Features

The TSENS has the following features:

- Temperature monitoring range: –40 to 150 °C
- Sensitivity: approximately 5.14 mV/°C

53.3 Signals

The TSENS does not use any input signals external to the device.

The TSENS outputs one analog signal, V_{TSENS}. Depending on the TSENS mode of operation, V_{TSENS} is proportional or inversely proportional to the device temperature. The V_{TSENS} signal is connected to ADC_0 channel 15. See Chapter 11, Analog-to-Digital Converter (ADC), for more information on how ADC channels are connected to V_{TSENS} and to other modules.

53.4 Memory map and register description

This section provides a detailed description of all registers accessible in the TSENS module.

53.4.1 Memory map

The TSENS module has no memory-mapped configuration or status registers. However, data can be read from flash memory. See Chapter 8, Flash Memory.

53.5 Modes of operation

The TSENS has two modes of operation:

- Proportional to absolute temperature (PTAT)—V_{TSENS} increases linearly with increasing temperature
- Complementary to absolute temperature (CTAT)—V_{TSENS} decreases linearly with increasing temperature

The mode of operation is controlled by the CTR1[31] field in the ADC, as described in Section 11.3.1.15, Conversion Timing Register 1 (CTR1).

53.6 Obtaining the device temperature using TSENS

To obtain the device temperature using TSENS, do the following:

- Extract the necessary TSENS calibration constants from the MPC5675K test flash memory (see Section 53.6.1, TSENS calibration constants).
- Measure V_{TSENS} in PTAT and CTAT modes.
- Calculate the device temperature using the equations in Section 53.6.2, Equations for converting TSENS voltage to device temperature.

The need for the measurements in two different modes is driven by the fact that although the TSENS output is linear in either mode, the intercept of the V_{TSENS}-T plot varies significantly based on the ADC reference voltage. Performing the two measurements allows the resulting equations to be independent of this reference voltage.

53.6.1 TSENS calibration constants

The equations needed to convert VTSENS to a device temperature depend on four calibration constants, as described in Table 53-1. These constants are determined during factory testing of the MPC5675K and stored in the MPC5675K test flash memory.

Table 53-1. TSENS calibration constants

Constant	Description
P_1	Code from the ADC converting V_{TSENS} in PTAT mode at 145 °C
P_2	Code from the ADC converting V_{TSENS} in PTAT mode at -40 °C
C_1	Code from the ADC converting V_{TSENS} in CTAT mode at 145 °C
C_2	Code from the ADC converting V_{TSENS} in CTAT mode at -40 °C

53.6.2 Equations for converting TSENS voltage to device temperature

In the equations below:

- P_n and C_n are the calibration constants described in Section 53.6.1, TSENS calibration constants
- T is the device temperature in °C
- P_x is the code from the ADC converting the VTSENS output in PTAT mode at a generic temperature T
- C_x is the code from the ADC converting the VTSENS output in CTAT mode at a generic temperature T
- $T_2 = -40$
- $T_1 = 145$

Define

$$A = P_x C_2 - P_2 C_x$$

$$B = C_x P_1 - P_x C_1$$

The temperature is calculated as

$$T = T_2 + \frac{(T_1 - T_2) \cdot A}{A + B}$$

This page is intentionally left blank.

Chapter 54
Wakeup Unit (WKPU)

54.1 Introduction

54.1.1 Overview

The WKPU supports one external source that can cause non-maskable interrupt requests or wakeup events. Figure 54-1 is a block diagram of the WKPU and its interfaces to other system components.

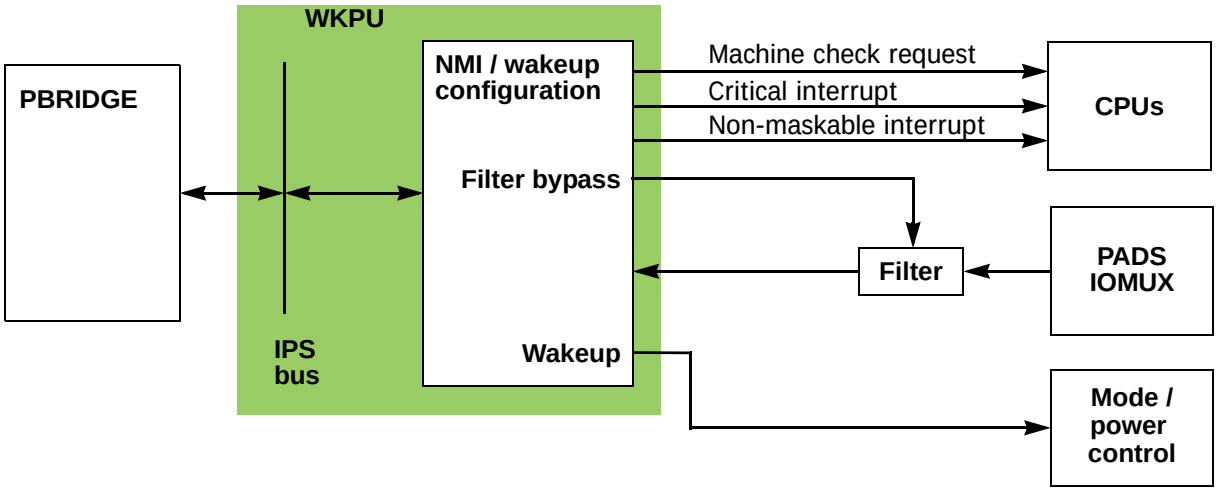


Figure 54-1. WKPU block diagram

54.1.2 Features

The WKPU supports these features:

Non-maskable interrupt (NMI) support with:

- 1 NMI source
- 1 analog glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
- Edge detection
- Configurable system wakeup triggering from NMI source

54.2 External signal description

The NMI input pin can be used as an external interrupt source in normal run mode or as a chip wakeup source during STOP0 mode.

NOTE

Be aware that the Wakeup pin is enabled in ALL modes. Therefore, the Wakeup pin should be correctly terminated to ensure minimal current consumption. If unused, the Wakeup signal input should be terminated by using an external pullup or pulldown.

54.3 Memory map and register description

This section provides a detailed description of all registers accessible in the WKPU module.

54.3.1 Memory map

Table 54-1 gives an overview of the WKPU registers implemented.

Table 54-1. WKPU memory map

Offset from WKPU_BASE (0xC3F9_4000)	Register	Access[1]	Reset Value[2]	Location
0x0000	NMI Status Flag Register (NSR)	R/W	0x0000_0000	on page 1722
0x0004–0x0007	Reserved			
0x0008	NMI Configuration Register (NCR)	R/W	0x0000_0000	on page 1723
0x000C–0x3FFF	Reserved			

1 In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

2 In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH_HHHH, where “H” is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

NOTE

Reserved registers are read as 0. Writes have no effect. If bus aborts are enabled by software in the SSCM_ERROR register, accessing reserved registers causes bus aborts. A transfer error is issued for any attempted access of a reserved register space.

54.3.1.1 NMI Status Flag Register (NSR)

The NMI Status Flag Register (NSR) holds the non-maskable interrupt status flags. This register supports 32-, 16-, and 8-bit accesses.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF	NOVF	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 54-2. NMI Status Flag Register (NSR)

Table 54-2. NSR field descriptions

Field	Description
NIF	NMI Status Flag. This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NREE or NFEE set), NIF causes an interrupt request. 0 No event has occurred on the pad. 1 An event as defined by NREE and NFEE has occurred.
NOVF	NMI Overrun Status Flag. This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF value whenever an NMI event occurs, thereby indicating to the software that an NMI occurred while the last one was not yet serviced. If enabled (NREE or NFEE set), NOVF causes an interrupt request. 0 No overrun has occurred on the NMI pin. 1 An overrun has occurred on the NMI pin.

54.3.1.2 NMI Configuration Register (NCR)

The NMI Configuration Register (NCR) holds the configuration bits for the non-maskable interrupt settings. This register supports 32-, 16-, and 8-bit accesses.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NLOCK	NDSS		NWRE	0	NREE	NFEE	NFE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 54-3. NMI Configuration Register (NCR)

Table 54-3. NCR field descriptions

Field	Description
NLOCK	NMI Configuration Lock Register. Writing a 1 to this bit locks the configuration for the NMI until it is unlocked by a system reset. Writing a 0 has no effect.
NDSS	NMI Destination Source Select. 00 Non-maskable interrupt. 01 Critical interrupt. 10 Machine check request. 11 Reserved. No NMI, critical interrupt, or machine check request generated.
NWRE	NMI Wakeup Request Enable. 0 System wakeup requests from the corresponding NIF bit are disabled. 1 A set NIF bit or set NOVF bit causes a system wakeup request.
NREE	NMI Rising-edge Events Enable. 0 Rising-edge event is disabled. 1 Rising-edge event is enabled.
NFEE	NMI Falling-edge Events Enable. 0 Falling-edge event is disabled. 1 Falling-edge event is enabled.
NFE	NMI Filter Enable. Enable analog glitch filter on the NMI pad input. 0 Filter is disabled. 1 Filter is enabled.

NOTE

Writing a 0 to both NREE and NFEE disables the NMI functionality completely. No system wakeup or interrupt is generated on any pad activity.

54.4 Functional description

54.4.1 General

This section provides a complete functional description of the WKPU.

54.4.2 Non-Maskable Interrupts (NMI)

The WKPU supports one non-maskable interrupt (NMI).

The WKPU supports the generation of three types of interrupts for the NMI input pin to the MPC5675K device. The WKPU supports capturing a second event before the interrupt is cleared, thus reducing the chance of losing an NMI event.

NMI passes through a bypassable analog glitch filter.

NOTE

Glitch filter control and pad configuration should be done while the NMI is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

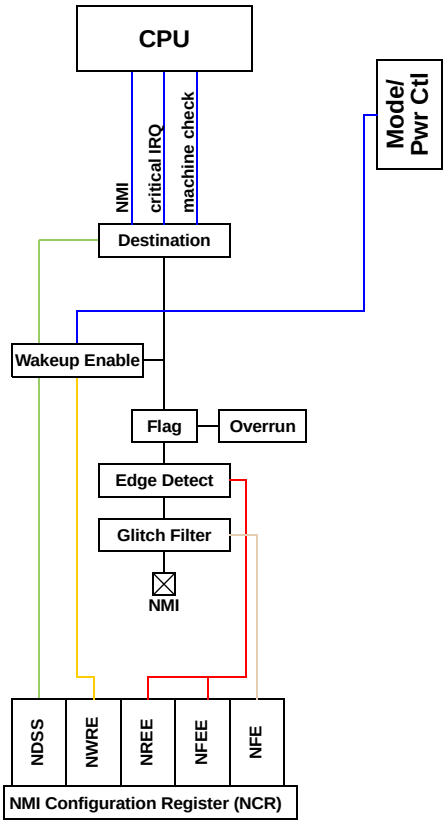


Figure 54-4. NMI pad diagram

54.4.2.1 NMI management

The NMI can be enabled or disabled using the single NCR register laid out to contain all configuration bits for an NMI in a single byte (see Figure 54-3). A pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge, or with both edges active. A setting with both edge events disabled results in no interrupt detection and should not be configured.

The active NMI edge is controlled through the configuration of the NREE and NFEE bits.

NOTE

After reset, NREE and NFEE are set to 0. Therefore the NMI functionality is disabled after reset and must be enabled explicitly by software.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See Table 54-3 for details.

The NMI supports a status flag and an overrun flag located in the NSR register (see Figure 54-2). This register is a write-1-to-clear register type, preventing inadvertent overwriting of other flags in the same register. The status flag is set whenever an NMI event is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, it has not yet been cleared).

NOTE

The overrun flag is cleared by writing a 1 to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt is not cleared.

Appendix A
Revision History

This appendix describes corrections to the *MPC5675K Microcontroller Reference Manual*. For convenience, the corrections are grouped by revision.

Beginning with Rev. 7, this revision history uses clickable cross-references for ease of navigation. The numbers and titles in these cross-references are relative to the latest published release.

NOTE

Section, table, and figure references shown in the "Description" column are relative to the newly-published revision. Thus, for example, if a table describes changes between revisions 3 and 4, the references are relative to revision 4.

A.1 Changes between revisions 9 and 10

Revision 9 uses change bars (vertical lines in the page margins) to indicate where technical (non-editorial) changes occurred.

Table A-1. Changes between revisions 9 and 10

Chapter	Description
1 Introduction	In Section 1.2, MPC5675K microcontroller comparison, updated table notes 2 and 6 of Table 1-1 (MPC5675K family device comparison).
2 Memory Map	In Section 2.1, Introduction: • In Table 2-1 (MPC5675K system memory map overview), added a table footnote. • In Table 2-2 (MPC5675K memory map), updated the entries for Static RAM (SRAM) for Lock Step and Decoupled Parallel modes.
4 Functional Safety	Updated the 3rd sentence in Section 4.4.1, BIST during boot.
7 General-purpose Static RAM (SRAM)	In Section 7.2.3.1, Lock Step Mode (LSM), updated "Start address", "Stop address" and "Size" entries for Table 7-2 (Address decoder details for MPC5674K)
8 Flash memory	In Section 8.2.1, Module memory map, memory size for the data flash test flash region changed from 4088 KB to 8 KB in Table 8-2 (Flash memory multi module sectorization). In Section 8.2.3.8, Platform Flash Configuration Register 0 (PFCR0), added a note in B02_RWSC field in Table 8-12 (PFCR0 field descriptions). In Section 8.2.3.9, Platform Flash Configuration Register 1 (PFCR1), added a note in B1_RWSC field in Table 8-13 (PFCR1 field descriptions).
11 Analog-to-Digital Converter (ADC)	Updated the equations in Section 11.4.4, ADC sampling and conversion timing. In Section 11.3.1.1, Main Configuration Register (MCR), added a note for CTU Trigger mode after Table 11-4 (MCR field descriptions). Updated Table 11-19 (xMinimum AD_ck frequency).
13 Clock Architecture	In Section 13.1, System clock generation, updated minimum and maximum values of FR_CLK in Table 13-1 (MPC5675K clock signals).

Table A-1. Changes between revisions 9 and 10 (continued)

Chapter	Description
14 Clock Generation Module (MC_CGM)	Made the following changes in Table 14-5 (CGM_OCDS_SC field descriptions), Table 14-6 (CGM_SC_SS field descriptions), Table 14-9 (CGM_AC0_SC field descriptions), Table 14-11 (CGM_AC1_SC field descriptions), and Table 14-13 (CGM_AC2_SC field descriptions): • System PLL replaced with FMPLL0_CLK • Secondary (80 MHz) PLL replaced with FMPLL1_1D1_CLK • Secondary (120 MHz) PLL replaced with FMPLL1_1D0_CLK Created EBI_CR register (Section 14.3.1.1, EBI Control Register (EBI_CR)) and updated Section Table 14-2., MC_CGM memory map with EBI_CR entry.
18 Cross-triggerering Unit (CTU)	Added a note in Section 18.6.2, ADC commands list format and Section 18.4.1.14, FIFO Status Register (FST).
22 External Bus Interface (EBI)	Updated Table 22-1 (Signal properties) - changed DATA [0:15] to DATA [0:31], made TA, TEA, and TS active low by creating a overbar over their signal names. Updated the title and text of Section 22.2.2.6, DATA [0:31] — Data Lines 0-31. Created a new section - Section 22.2.2.14, ALE - Output. In Section 22.4.2.4.1, Single beat read flow, updated Figure 22-12 (Single beat 32-bit read cycle, CS access, SETA = 1, zero wait states). In Section 22.4.2.4.2, Single beat write flow, updated Figure 22-19 (Single beat 32-bit write cycle, CS access, zero wait states, LWRN = 1). In Section 22.4.2.4.3, Back-to-back accesses, updated Figure 22-25 (Read after write to the same CS bank (EBI_BR[GCSN] = 1)) for CS signal.
24 Enhanced Motor Control Timer (eTimer)	In Section 24.3.3.11, Status Register (STS), updated description of ICF2 field bit in Table 24-13 (STS field descriptions). In Section 24.3.3.6, Hold Register (HOLD), updated the description of HOLD register. Added 2nd paragraph in Section 24.6, DMA.
26 Fast Ethernet Controller (FEC)	In Section 26.3.4.8, MII Speed Control Register (MSCR),: • In Table 26-11 (MSCR field descriptions), changed “MII_SPEED × 4” to “MII_SPEED × 2” for MII_SPEED bitfield. • In Table 26-12 (Programming Examples for MSCR), changed MII_SPEED bitfield entry of “0xA” to “0x10” for 80 MHz System clock frequency.
27 FlexCAN	Added a new section for programming steps - Section 27.6, Programming Considerations.
33 JTAG Controller (JTAGC)	In Section 33.2.1, Overview, added a note on LBIST in Table 33-1 (JTAG signal properties).
34 LIN Controller (LINFlexD)	In Section 34.10.3, LIN Status Register (LINSR), added a note in LINS field of Table 34-16 (LINSR field descriptions). In Section 34.13.2, Slave node, added programming steps (just below Figure 34-70 (Programming consideration: slave node, transmit filter, receive filter, BF is set)).

Table A-1. Changes between revisions 9 and 10 (continued)

Chapter	Description
36 Multi-Port DDR DRAM Controller (MDDRC)	In Section 36.3.2.1, DDR System Configuration Register (DDR_SYS_CONFIG), in Table 36-2 (DDR_SYS_CONFIG field descriptions), added a note in WDLY bit of DDR_SYS_CONFIG register in . In Section 36.3.2.2.3, DDR Time Configuration Register 2 (DDR_TIME_CONFIG2)in Table 36-8 (Timing parameters), added CCD and RTP timing parameters. In Section 36.3.2.6, DQS Config Offset Count (DQS_CONFIG_OFFSET_COUNT) Register changed “DQS_SLAVE_[3:0]_OFFSET_COUNT” to “DQS_SLAVE_[6:0]_OFFSET_COUNT” in Table 36-12 (DQS_CONFIG_OFFSET_COUNT field descriptions). In Section 36.3.2.7, DQS Config Offset Time (DQS_CONFIG_OFFSET_TIME) Register, changed the width of all field bits from 5 to 6 and changed “DQS_SLAVE_[3:0]_ OFFSET_TIME” to “DQS_SLAVE_[5:0]_OFFSET_TIME” in Table 36-13 (DQS_CONFIG_OFFSET_TIME field descriptions). In Section 36.3.2.10, DDR Extra Attributes Register, added note below Figure 36-20 (SDR read timing diagram).
39 Nexus Development Interface (NDI)	In Section 39.2.3.6, Nexus double data rate (DDR) mode, removed setting CORE_CLK/8 for MCKO_DIV. Added a note in Section 39.6.2.3, Message rules.
40 Oscillators	Updated text in Section 40.2, IRCOSC 16 MHz internal RC oscillator. (The description of functioning of IRC after POR is updated).
41 Parallel Digital Interface (PDI)	Added a new paragraph - “When DMA_EN and DEN are set,....”, in Section 41.4.1.1, ADC mode.
44 Power Management Controller (PMC)	Added a note above Section 44.1, Introduction, mentioning that PMC and PMU are same.
46 Reset Generation module (MC_RGM)	In Section 46.3.1.2, Destructive Event Status Register (RGM_DES), added a note explaining functioning of RGM.F_POR field during power-up phase.
53 Temperature Sensor (TSENS)	In Section 53.1, Introduction, Figure 53-1 (TSENS block diagram) and Section 53.5, Modes of operation, changed reference of CTR1[TSENSOR_SEL] to CTR1[31]. In Section 53.6.1, TSENS calibration constants, updated Table 53-1 (TSENS calibration constants). Updated Section 53.6.2, Equations for converting TSENS voltage to device temperature.

A.2 Changes between revisions 8 and 9

Revision 9 uses change bars (vertical lines in the page margins) to indicate where technical (non-editorial) changes occurred.

Table A-2. Changes between revisions 8 and 9

Chapter	Description
1 Introduction	In Section 1.2, MPC5675K microcontroller comparison, Table 1-1 (MPC5675K family device comparison), added footnotes to stipulate the peripheral instances that are used on derivative devices (same change as in Data Sheet Revision 7): • Added footnote to **MPC5673K** DSPI module: “DSPI_0 and DSPI_1.“ • Added footnote to **MPC5673K** I2C module: “I2C_0 and I2C_1.“ • Added footnote to **MPC5673K** LinFlex module: “LinFlex_0, LinFlex_1, and LinFlex_2“ In Section 1.2.1, Block diagram, Figure 1-1 (MPC5675K block diagram), corrected mistaken label “Core_1” to “Core_0”.
2 Memory Map	In Section 2.1, Introduction, • In Table 2-1 (MPC5675K system memory map overview), removed the column for End Address for consistency with other tables. • In Table 2-2 (MPC5675K memory map), added footnotes “Peripherals from address 0x8FF0_4000 to address 0xFFEF_FFFF are tied to PBRIDGE_1” and “Peripherals from address 0xFFF0_4000 to address 0xFFFF_FFFF are tied to PBRIDGE_0.”
3 Signal Description	In Section 3.2.3, System pins, Table 11 (LBGA473 MAPBGA system pins), added footnote to ball R23 and T23: “PCR234 can be used to control the slew rate of DRAM CLK and DRAM CLKB.”
6 e200z7 Core	In Section 6.2.3.4, Interrupt registers, added NOTE: “After power-on, the IVORn registers are not initialized to a known value. The user software must initialize the IVORn registers before any other register after power-on.”
8 Flash Memory	In Section 8.3.2.2, Margin read, added description that the margin read feature should not be used in the user application.
10 Peripheral Bridge (PBRIDGE)	In Section 10.4.2, Memory map, Table 10-1 (PBRIDGE module organization), corrected “Peripheral connections” for DPM PBRIDGE_0 to "Peripherals with PCTL number less than 64." and DPM PBRIDGE_1 to "Peripherals with PCTL number greater than 63."

Table A-2. Changes between revisions 8 and 9 (continued)

Chapter	Description
11 Analog-to-Digital Converter (ADC)	Added Section 11.2.4, Debug mode to describe debug mode does not affect the function of the module. In Section 11.3.1.12, Presampling Control Register (PSCR), added the PREVAL1 bitfield and modified description of PREVAL0. In Section 11.3.1.14, Conversion Timing Register 0 (CTR0): • Modified the register description to apply for external channels. • In Table 11-19 (xMinimum AD_ck frequency), added column “Value for INPCMP in Eqn. 2“ for clarification. n Section 11.3.1.15, Conversion Timing Register 1 (CTR1): • Modified the register description to apply for internal channels. • Changed bit 31 TSENSOR_SEL to be a part of the INPSAMP bitfield and added description for swtiching TSENS operating mode to this field. In Section 11.3.1.32, Self-Test Data Register 2 (STDR2), added description for delay needed before reading results from the S1 self-test algorythm to the FDATA and IDATA fields. **Previous errata ERR004146 integrated into the reference manual**: Added a NOTE to Section 11.4.2.3, Injected channel conversion. **Previous errata ERR004168 integrated into the reference manual**: Added a NOTE to Section 11.4.2.4, Abort conversion. **Previous errata ERR004186 integrated into the reference manual**: Added a NOTE to Section 11.4.2.4, Abort conversion. In Section 11.4.4, ADC sampling and conversion timing, added NOTE “Teval is equivalent to datasheet parameter tADC_E. Tsample is equivalent to datasheet parameter tADC_S.” for clarification. In Section 11.4.11.5, Self-test analog watchdog, clarified description for delay needed before reading results from the S1 self-test algorythm to be about the ADC clock.
18 Cross-Triggering Unit (CTU)	**Previous errata ERR004166 integrated into the reference manual**: Added a NOTE to Section 18.4.1.16, FIFO Signed Left-Aligned Data x Register (FLx). Added Section 18.5.7, Debug mode to describe debug mode does not affect the function of the module.
21 Deserial Serial Peripheral Interface (DSPI)	In Section 21.4.4.5, Continuous selection format, added NOTE: “User must fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. While transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0). When operating in slave mode, ensure that when the last-entry in the TXFIFO is completely transmitted (that is the corresponding TCF flag is asserted and TXFIFO is empty), the slave is deselected for any further serial communication; otherwise an underflow error occurs.” In Section 21.4.5, Continuous serial communications clock, added NOTE: “When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the MCR[CLR_TXF] field before initiating transfer.”
22 External Bus Interface (EBI)	Added new section Section 22.2.2.4, External calibration bus clock divider.
23 Error Correction Status Module (ECSM)	In Section 23.3.3.1, ECC Configuration Register (ECR), added NOTE “If you choose to use the ECC error reporting functionality, the user software should enable this immediately after initialization of the core registers such as IVOR and MMU. An ECC error occurring before this functionality is enabled will not be reported. Note that the flash block also needs to enable this functionality by setting UT0[SBCE].”

Table A-2. Changes between revisions 8 and 9 (continued)

Chapter	Description
25 Fault Collection and Control Unit (FCCU)	Added Section 25.5, Debug mode to describe debug mode does not affect the function of the module.
26 Fast Ethernet Controller (FEC)	Added Section 26.2.5, Debug mode to describe debug mode does not affect the function of the module.
27 FlexCAN Module	**Previous errata ERR003407 integrated into the reference manual**: Added a NOTE to Section 27.3.2, Message buffer structure. **Previous errata ERR002656 integrated into the reference manual**: Added a NOTE to Section 27.4.6.1, Transmission abort mechanism.
28 Motor Control Pulse Width Modulator Module (FlexPWM)	**Previous errata ERR003355 integrated into the reference manual**: added a NOTE to Section 28.3.2.14, DMA Enable Register (DMAEN) and Section 28.8, DMA. **Previous errata ERR003511 integrated into the reference manual**: Removed all references to MCTRL[IPOL] in the chapter and added a NOTE to Section 28.4.3.8.2, Manual correction.
29 FlexRay Communication Controller (FLEXRAY)	Added Section 29.1.6.3, Debug mode to describe debug mode does not affect the function of the module.
31 Inter-Integrated Circuit Bus Controller Module (I2C)	Added Section 31.1.4.1, Debug mode to describe debug mode does not affect the function of the module.
34 LIN Controller (LINFlexD)	Added Section 34.2.3, Debug mode to describe debug mode does not affect the function of the module. **Previous errata ERR004340 integrated into the reference manual**: Added a NOTE to Section 34.9.4, UART receiver.
35 Mode Entry Module (MC_ME)	**Previous errata ERR003570 integrated into the reference manual**: Added a CAUTION to Section 35.4.3.11, Processor low-power mode exit.
36 Multi-Port DDR DRAM Controller (MDDRC)	Added Section 36.1.2, Debug mode to describe debug mode does not affect the function of the module. In Section 36.3.2.1, DDR System Configuration Register (DDR_SYS_CONFIG), Table 36-2 (DDR_SYS_CONFIG field descriptions), changed Description of EARLY ODT field from “This bit needs to be set if write latency is 1 (WDLY = 001) and on-die termination is used with DDR2 DRAM. It makes sure the DRAM controller asserts the ODT signal going to the DRAM one clock ahead of issuing the write command.” to “This bit controls the assertion of the MODT signal when on-die termination (ODT) is used with DDR2 DRAM. If set to 0 the MODT is asserted two clocks before the write data. If set to 1 the MODT is asserted three clocks before the write data. It may be necessary, depending on the time required to turn on termination resistors in the memory device, to set EARLY_ODT=1. Doing so will insert an extra clock cycle between non-consecutive back-to-back write cycles but has no other performance impact.” In Section 36.3.2.2.3, DDR Time Configuration Register 2 (DDR_TIME_CONFIG2), Table 36-8 (Timing parameters), removed all references to 32-bit mode operations and associated timing values.

Table A-2. Changes between revisions 8 and 9 (continued)

Chapter	Description
37 Multi-Port DRAM Controller Priority Manager (PRIOMAN)	Added Section 37.1.2, Debug mode to describe debug mode does not affect the function of the module.
39 Nexus Development Interface (NDI)	**Previous errata ERR003449 integrated into the reference manual**: Added a CAUTION to Section 39.6.1, NPC reset configuration.
42 Periodic Interrupt Timer (PIT)	In Section 42.5.1, General, added NOTE: “The PIT and the DMA need to run at the same frequency (synchronous). For the PIT frequency it must be considered the PIT runs with the peripheral set 0 clock, which clocks also DSPI's, FlexCAN's, and LinFlex.”
44 Power Management Controller (PMC)	In Section 44.5, Register map, added TRIM1 register. Added Section 44.5.3, Trimming Register 1 (TRIM1). In Section 44.5.1, Configuration Register (CFGR), in Table 44-3 (CFGR field descriptions), deleted reference to WRITE_ONCE_TRIM_REG. In Section 44.9, LVD core, corrected “Rising LVD threshold voltage is documented under the LvdC symbol in the MPC5675K data sheet.” to “The LVD threshold voltage is documented under the LvdC symbol in the MPC5675K data sheet.” In Section 44.10, HVD core, corrected “Rising LVD threshold voltage is documented under the HvdC symbol in the MPC5675K data sheet.” to “The HVD threshold voltage is documented under the HvdC symbol in the MPC5675K data sheet.” In Section 44.12, ADC measure channel • Added "After measuring the voltage of the 3.3V LVDx, a multiplication factor of 2.3875 can be applied to the result to obtain the actual voltage level." • Deleted duplicate list item and corrected point reference names.
45 Register Protection (REG_PROT)	In Section 45.6, MPC5675K registers under protection, Section Table 45-5., MPC5675K register protection, removed line for register PCR235.
46 Reset Generation Module (MC_RGM)	**Previous errata ERR004334 integrated into the reference manual**: Added a CAUTION to Section 46.4.3, External reset.
48 System Integration Unit Lite (SIUL)	In Section 48.5.1, SIUL memory map, Table 48-2 (SIUL memory map), changed “Pad Configuration Registers 0–233” to “Pad Configuration Registers 0–234”. In Section 48.5.2.8, Pad Configuration Registers (PCRn), • Figure 48-9 (Pad Configuration Registers (PCRn)), changed the end base address from “PCR233” to “PCR234”. • Table 48-10 (PCRn field descriptions), in the HYS bitfield description, added "Note: For PCRn where the HYS bitfield is listed as "X", the input hysteresis input for the pad is hardwired to '1' (enabled), and is not configurable. For PCRn where HYS is available, the HYS bit controls hysteresis as described above.” • Table 48-11 (PCRn addresses), added table entry for PCR234.
50 Self-Test Control Unit (STCU)	In Section 50.4.3.7, STCU LBIST Status Register (STCU_LBS), Section 50.4.3.11, STCU MBIST Status Low Register (STCU_MBSL), and Section 50.4.3.12, STCU MBIST Status High Register (STCU_MBSH), added “The flags are cleared by hardware upon next reset which is able to restart BIST.” for clarification.
52 Software Watchdog Timer (SWT)	In Section 52.3.2.7, SWT Service Key Register (SWT_SK), corrected title of Figure and Table.

Table A-2. Changes between revisions 8 and 9 (continued)

Chapter	Description
53 Temperature Sensor (TSENS)	In Section 53.1, Introduction, Figure 53-1 (TSENS block diagram)and Section 53.5, Modes of operation, changed reference to CTR1[TSENSOR_SEL] to CTR1[31].
A Memory Map	Removed section from book.
B Revision History	Updated for changes in Rev. 9.

A.3 Changes between revisions 7 and 8

Revision 8 uses change bars (vertical lines in the page margins) to indicate where technical (non-editorial) changes occurred.

Table A-3. Changes between revisions 7 and 8

Chapter	Description
Throughout	Minor modifications to functional safety-related descriptions.
1 Introduction	In Section 1.1, The MPC5675K microcontroller, added the Qorivva and SafeAssure information. In Table 1-1 (MPC5675K family device comparison), for the CPU/Data Cache entry, changed "16 KB, 4-way with EDC (SoR)" to "16 KB, 4-way with Parity (SoR)". In Section 1.2.1, Block diagram, Figure 1-1 (MPC5675K block diagram), • Added missing modules (PMC, SPE2, VLE, and flash. • Added an arrow each from Core_0 and Core_1 to the XBAR modules to represent the data path. • Updated the Redundancy Checkers to reflect the actual implementation. • Renamed the “JTAG/Nexus” block to “Debug”, with JTAG and Nexus shown as submodules. In Section 1.4.1, High-performance e200z7d core processor and Section 1.4.9, Cache memory, removed the bullet “Supports tag and data parity" and added the following bullets: — Supports tag and data cache parity — Supports EDC for instruction cache In Section 1.4.11, DRAM controller, added the support for “DDR 2 (optional)” bullet. In Section 1.4.14, Deserial Serial Peripheral Interface (DSPI) modules, updated the chip selects subbullet from 8 DSPI_0 and 4 DSPI_1/2 chip selects to “Six chip selects, expandable to 64 with external multiplexers“. In Section 1.4.16, FlexCAN, added bullet that there are safety CAN features on 1 CAN module.
2 Memory Map	In Section 2.1, Introduction, Table 2-1 (MPC5675K system memory map overview), changed header “Size (MB)” to “Size (KB)”.

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
4 Functional Safety	In Section 4.4.2, Software-triggered BIST during operation, • Changed "... because testing every 10 hours (trip time)..." to "... because testing only during destructive reset (poweron reset)..." • Deleted "(10 ms)" In Section 4.6, Monitoring: • Changed "...the process safety time of 10 ms" to "10 ms". • Changed "... two temperature sensors" to "... one temperature sensor" In Section 4.7, Software measures, • Changed "Several software measures are required to achieve safety integrity for this microcontroller. Software has to trigger these test features at least every 10 ms (process safety time)." to "Several software measures are recommended to achieve functional safety integrity for this microcontroller. Software has to trigger these test features at least once within the process safety time or fault tolerant time interval. • In the sentence "No software-based core self-test routine library is necessary for this microcontroller." changed "necessary" to "mandatory". • Changed "ADC: Configuration registers checked" to "ADC: ADC configuration registers checked" • Changed "eTimer: Configuration registers checked" to "eTimer: Timer configuration registers checked"
5 Boot and Operating Modes	In Section 5.5.2, Decoupled Parallel mode (DPM), added "Reciprocal comparison of data caclulated independently on both CPUs may be required by software to achieve respective functional safety integrity."
6 e200z7 Core	In Section 6.1.2.4, Cache features, added the bullet "Parity protection for the DCache". In Section 6.3.4, L1 cache features, added the bullet "Supports parity for the data cache".
7 General-Purpose Static RAM (SRAM)	In Section 7.2.3.2, Decoupled Parallel Mode (DPM), Figure 7-2 (SRAM integration in DPM), • Corrected color of RC Checker SRAMC to gray • Corrected color of SRAM Array0 and SRAM array 1 to blue
8 Flash Memory	In Section 8.2.5, Test sector, Table 8-30 (Test flash memory information) • for Word name "TSENS1_CAL W1/W2" and "TSENS1_CAL W3/W4", changed "Temperature sensor 0 Calibration data..." to "Temperature sensor Calibration data..." in Function and deleted the Note "2 × 12-bit words". • for Word name "ADC0_CAL W4", "ADC0_CAL W5", "ADC1_CAL W4", ADC2_CAL W5", "ADC2_CAL W4", "ADC2_CAL W5", "ADC3_CAL W4", and "ADC3_CAL W5",changed "STAW0HR/L" to "STAW0RH/L" in Note. • for Word name "ADC3_CAL W8", changed "2 × 12-bit words (STAW2R)" to "1 × 12-bit word (STAW2R)". • changed Address "0x0410", "0x0420", "0x0450", "0x0490", "0x04A0", "0x04B0", "0x04C0", "0x04D0", "0x04E0", "0x04F0", and "0x0500", to "Reserved".
9 Multi-Layer AHB Crossbar Switch (XBAR)	In Section 9.1.1, Logical master IDs, changed "The logical master IDs for the two cores are different in DPM so that they both can access the same XBAR" to "The logical master IDs for the two cores are different in DPM so that they both can access either XBAR_0 or XBAR_1". In Section 9.2.1, Register summary, changed "... an error response is returned if an unimplemented location is accessed within the XBAR" to "... a bus error is returned if an unimplemented location is accessed within the XBAR". In Section 9.3.10.4.4, Slave port state machine parking, changed "…be guaranteed that no other master has had access to it." to "...delaying the access of other masters to it."

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
11 Analog-to-Digital Converter (ADC)	In Section 11.3.1.23, Self-Test Configuration Register 1 (STCR1), Table 11-28 (STCR1 field descriptions), for the “INPSAMP_S” field, changed “0x7F Reserved, 0x80 Minimum value, 0xFF Maximum value” to “0xFE Reserved, 0xFF Sampling phase duration value”. In Section 11.4.1, ADC channel muxing, Table 11-45 (ADC_0 and ADC_1 channel muxing) and Table 11-46 (ADC_2 and ADC_3 channel muxing), added a footnote regarding minimum sample time when performing conversions of the internal PMC channels. In Section 11.4.9, Power-down mode, added "After an exit from power-down mode, the first conversion can be started after 5 μs, otherwise the result can be affected by the improper setting of the ADC analog operating point." In Section 11.4.11.3, CTU mode, added Note : “If the CTU trigger arrives while normal chain execution was ongoing and the current normal channel is in evaluation phase of conversion, then the current channel might not be restored back after CTU injected conversion finishes. For example, if conversions ch0 → ch1 → ch2 → ch3 → ch4 are ongoing and the CTU trigger arrives at ch2 (evaluation phase) then ch2 would be aborted and would not be restored back after the injected CTU channel conversion is over; converted Data of ch2 is lost for the current executing chain.” In Section 11.4.11.6, Watchdog timer, deleted the following note: “To prevent an interrupt request that occurs after enabling a watchdog timer (WDTE) and before the start of ADC self test (NSTART) from delaying the ADC self test start, it is possible to swap the order of WDTE and NSTART programming. Specifically, the programmer can use the following sequence of NSTART and WDTE programming: - Start the normal conversion by setting MCR[NSTART] - Enable watchdog timer by setting STAWxR[WDTE] - Freeze the above settings by writing MCR[STCL] (configuration lock)”
12 Boot Assist Module (BAM)	In Section 12.6.7.2.1, Choosing the host baud rate, Table 12-10 (Maximum and minimum recommended baud rates), for the table headers “Max baud rate for guaranteed < 2.5% deviation” and “Min baud rate for guaranteed < 2.5% deviation”, deleted “guaranteed”.
13 Clock Architecture	In Section 13.5.1, FlexCAN clock domains, added “NOTE: Robustness regarding clocks of Communication Protocol Engines The clock domain (Protocol Clock) of the Communication Protocol Engines (FlexRay Module and FlexCAN Module) can either be fed from one of the on-chip frequency modulated PLLs (FMPLL0 or FMPLL1) or directly from the external crystal oscillator. To improve the robustness of the system regarding clock disturbs picked up by components of the external and internal crystal oscillator circuitry, a low pass filter is integrated within the circuitry of the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1). Using the on-chip frequency modulated PLLs (FMPLL0 and FMPLL1) as the clock source for Communication Protocol Engines (FlexRay Module and FlexCAN Module) will reduce the probability of external clock disturb related failures. Using directly the external crystal oscillator as clock source for a Communication Protocol Engine (FlexRay Module or FlexCAN Module) may require additional system measures to improve the robustness regarding clock disturbs picked up by the external and internal crystal oscillator circuitry, as the clock direct from the external crystal oscillator is not filtered by an integrated filter circuitry.” In Section 13.5.2, FlexRay clock domains, replaced “NOTE: PE Clock Source — Functional Safety” with the same NOTE for FlexCAN, above.
14 Clock Generation Module (MC_CGM)	In Section 14.4.1.4, Auxiliary Clock Generation, Figure 14-17 (MC_CGM Auxiliary Clock 0 Generation Overview), Figure 14-18 (MC_CGM Auxiliary Clock 1 Generation Overview), and Figure 14-19 (MC_CGM Auxiliary Clock 2 Generation Overview), removed the unused connection.

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
18 Cross-Triggering Unit (CTU)	In Section 18.4.1.10, Commands List Register x (CLRx), Table 18-14 (CLRx field descriptions (channel conversion commands)) and Table 18-15 (CLRx field descriptions (self-test commands)), aadded to LC field description " The CTU state machine triggers the ADC conversions configured into the command list until it finds a command with the LC bit set to one (this command is not executed)." and "Note: This bit may be called FC or LC depending on the device, but the functionality is identical." In Section 18.6.3, ADC results, • changed "FIFO1", "FIFO2", "FIFO3", and "FIFO4" to "FIFO0", "FIFO1", "FIFO2", and "FIFO3", respectively. • changed "... 10-bit resolution ..." to "... 12-bit resolution ..." In Section 18.8, STOP mode • Changed "The FIFOs are considered a lot like memory mapped registers, otherwise there could be some problems if a read operation occurs during the MDIS bit set period. When the clock is started after an MDIS bit setting or a stop signal, some mistakes could occur. " to "The FIFOs are considered a lot like memory mapped registers. When the clock is started after a stop signal, some mistakes could occur. " • Changed "For this reason after a stop signal after a MDIS bit setting the FIFO have to be empty." to "For this reason after a stop signal the FIFO must be empty."
19 Enhanced Direct Memory Access (eDMA)	In Section 19.4.7.2, Dynamic channel linking, removed all references to dynamic scatter/gather and TCD.e_sg bits. Added Section 19.4.7.3, Dynamic scatter/gather.
20 eDMA Channel Mux (DMACHMUX)	In Section 20.4, DMACHMUX request source slot mapping, Table 20-5 (DMACHMUX source slot mapping), changed "FIFO1", "FIFO2", "FIFO3", and "FIFO4" to "FIFO0", "FIFO1", "FIFO2", and "FIFO3", respectively.
25 Fault Collection and Control Unit (FCCU)	In Section 25.7.22, FCCU IRQ Enable Register (FCCU_IRQ_EN), in Figure 25-28 (FCCU IRQ Enable Register (FCCU_IRQ_EN)), changed the bit "CFG_TOIEN" to "CFG_TO_IEN". In Section 25.8.8.1, Monitoring JTAGC and NPC, changed "It is important to safe state NPC signals before asserting JCOMP to the device since this may trigger the monitoring circuit. Users should disable Nexus tracing before JCOMP is asserted." to "Asserting JCOMP to the device while Nexus is working may trigger the monitoring circuit. Therefore the user should disable Nexus tracing to safe state NPC signals, before JCOMP is asserted." for clarification.
26 Fast Ethernet Controller (FEC)	In Section 26.1.1, Overview, added paragraph "MPC5675K uses a 36 x 128 array for the FIFO RAM (about 576 bytes). Of that, the default size of the Tx FIFO (as determined by the FRSR[R_FSTART] field) is 64 of the available 128 entries or 288 bytes, but this is programmable via the FRSR register." In Section 26.1.3, Features, added "36 x 128 array FIFO RAM (about 576 bytes)"

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
28 Motor Control Pulse Width Modulator Module (FlexPWM)	In Section 28.1.4.2, PWM submodule, Figure 28-2 (PWM submodule block diagram), added flipflop to generate PWM01. Changed PWMA/PWMB to PWM23/PWM45 respectively. In Section 28.2.2, PWMX[n]—Auxiliary PWM signal, changed “IPOL” to “MCTRL[IPOL]”. In Section 28.3.1, FlexPWM module memory map, added “All PWM registers are 16 bits wide.” In Section 28.3.2.2, Initial Count Register (INIT), changed “LDOK” to “MCTRL[LDOK]” and “LDMOD” to “CTRL1[LDMOD]” for clarity. In Section 28.3.2.3, Control 2 Register (CTRL2), in Table 28-4 (CTRL2 field descriptions) for the FORCE field, changed “SEL23” to “DTSRCSEL[SEL23]” and “SEL45” to “DTSRCSEL[SEL45]” for clarity. In Section 28.3.2.6, Value Register 1 (VAL1), added to NOTE “Since the VAL1 register is used to determine the turn off count of the PWMX output and also to define the end count of the PWM cycle, the PWMX output cannot achieve 100% duty cycle.” In Section 28.3.2.18, Capture Control X Register (CAPTCTRLX), in Table 28-14 (CAPTCTRLX field descriptions) for the ARMX field, added “In order to restart capture functionality, disable the capture function by setting ARMX=0, read CVAL0 and CVAL1 until CX0CNT and CX1CNT are zero, then re-enable capture by setting ARMX=1.” In Section 28.3.2.21, Capture Value 0 Cycle Register (CVAL0CYC) and Section 28.3.2.23, Capture Value 1 Cycle Register (CVAL1CYC), added “CVAL0CYC will count up and roll over to 0. CVAL0CYC and CVAL1CYC can be used to verify that the rising and falling edges occurred in the same PWM cycle.” In Section 28.4.2.7, Synchronous switching of multiple outputs, changed “...for the next FORCE_OUT event.” to “...for the next FORCE_OUT event using the DTSRCSEL[SEL23 and SEL45] bits.” for clarification. In Section 28.4.3.9, Output logic, changed Figure 28-57 (Output logic section) to include the mask feature. In Section 28.8, DMA, in Table 28-27 (DMA summary), corrected “Capture FIFO X!” to “Capture FIFO X1”.
30 Frequency-Modulated Phase-Locked Loop (FMPLL)	In In Section 30.2, Overview, • Moved the “MPC5675K has two PLLs:...” sentence to the top of the paragraph. • Added “... (although the one for motor control peripherals is intended to be operated without FM)” In In Section 30.3, Features, added “(default)” after “PLL mode with crystal reference”. In Section 30.6.1, Normal mode, added example setting value for the *phi* equation. In Section 30.6.3, Normal mode with frequency modulation, added “Note that frequency modulation needs to be disabled on the PLL driving the motor control clocks.” In Section 30.7, Recommendations, • Added “OSC_CTL[EOCV] ensures that the external oscillator clock signal is stable before it can be selected by the system.” • Added the following NOTE: “Progressive clock switch is very fast and the current transition here can be too fast for certain regulators to handle. The following programming and switching will help both internal and external regulators during the start-up of the FMPLL. 1. Program PLL for 180 Mhz with ODF at 3 so PLL output is 22.5 Mhz. 2. Enable PLL. Wait for 200 ms from this point. PLL lock is coarse lock and not fine lock. 3. Shift System clk to PLL output. 4. Change ODF setting from 3 → 2 → 1 → 0 in 10 ms intervals.”

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
32 Interrupt Controller (INTC)	In Section 32.6, Interrupt sources, Table 32-10 (Interrupt sources for INTC_0 and INTC_1), • For IRQ # 125, corrected"ipi_int" to "I2C_0_IBSR[IBIF]". • For IRQ # 126, corrected"ipi_int" to "I2C_1_IBSR[IBIF]". • For IRQ # 251, changed “irq_misc_b” to “external alarm interrupt line”. • For IRQ # 292, corrected"ipi_int" to "I2C_2_IBSR[IBIF]" and "IIC_2" to "I2C_2".

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
35 Mode Entry Module	In Section 35.1.2, Features, in the “Peripheral clock gating control” bullet, changed “ME_PCTL0...143” to “ME_PCTL*n*”. In Table 35-1 (MC_ME mode descriptions): • Changed “Hardware failure” to “Hardware failure from any mode” in the SAFE Entry description. • Added “SAFE via software or hardware failure“ to the TEST Exit description. • Added “DRUN” to the RUN0...3 Exit description. In Table 35-2 (MC_ME register description), removed the “...” entries and added the full descriptions of the following registers that were missing: • ME_RUN_PC2 through ME_RUN_PC6. • ME_LP_PC2 through ME_LP_PC6. In Table 35-3 (MC_ME memory map): • Combined the two adjacent reserved spaces into a single entry. • Added a footnote to the ME_PCTL0...143 entry that explains occupied versus unoccupied peripheral locations. In the introductory text of Section 35.3.2, Register Description, added a sentence regarding read-only fields. In Table 35-11 (Mode Configuration Registers (ME_<mode>_MC) field descriptions), added the following note to the IRCOSCON entry: “This bit is meaningful only in Test Mode.“ In Table 35-12 (Peripheral Status Registers 0…4 (ME_PS0…4) field descriptions), removed the text regarding “If no peripheral is mapped on a particular position“ from the S_*<periph>* entry. In Section 35.3.2.19, Run Peripheral Configuration Registers (ME_RUN_PC0…7), changed “run modes“ to “non-low-power modes“. In Section 35.3.2.20, Low-Power Peripheral Configuration Registers (ME_LP_PC0…7), changed “non-run modes“ to “low-power modes“. In Section 35.3.2.21, Peripheral Control Registers (ME_PCTLn): • Changed “ME_PCTL0...143” to “ME_PCTL*n*” in the section title, the register figure title, and the field description table title. • Changed “run modes” to “non-low-power modes”. • Changed “non-run modes” to “low-power modes”. In Figure 35-23 (MC_ME mode diagram), changed “non-recoverable failure” to “non-recoverable hardware failure” for the RESET state. In Section 35.4.2.5, RUN0…3 modes, removed SAFE mode from the bullet list. In Section 35.4.2.6, HALT0 mode, Section 35.4.3.7, Flash modules switch-on, and Section 35.4.3.16, Flash switch-off, specified that low-power mode applies only to code flash while power-down mode applies to both code and data flashes. In Section 35.4.2.7, STOP0 mode, updated the configuration information. In Table 35-16 (MC_ME resource control overview): • IRCOSC, STOP0 mode: Added shading and removed check mark. • PLL1, STOP0 mode: Added shading and removed check mark. • DFLASH, HALT0 mode: Changed “low-power” to “normal”. • MVREG, STOP0 mode: Added shading and removed check mark. • MVREG, HALT0 mode: Added shading and removed check mark. In Section 35.4.3.3, Peripheral clocks disable, changed “ME_PCTL0...143” to “ME_PCTL*n*”. In Section 35.4.3.6, Clock sources switch-on and Section 35.4.3.14, Clock sources (with no dependencies) switch-off: • Added note that “System PLL” is “PLL0”. • Added note that “Secondary PLL” is “PLL1”. In Section 35.4.3.9, Peripheral clocks enable, changed “ME_PCTL0...143” to “ME_PCTL*n*”.

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
35 Mode Entry Module (continued)	In Section 35.4.3.12, System clock switching, added a note that “System PLL” is “PLL0”. In Table 35-17 (MC_ME system clock selection overview), for “16 MHz internal RC oscillator”, removed “default” from RESET, SAFE, and STOP0 modes. In Section 35.4.6, Peripheral clock gating and Figure 35-25 (MC_ME application example flow diagram), changed “ME_PCTL0...143” to “ME_PCTL*n*”.
44 Power Management Controller (PMC)	In Section 44.5.1, Configuration Register (CFGR), Table 44-3 (CFGR field descriptions), Section 44.5.4, Trimming Register 2 (TRIM2), Table 44-6 (TRIM2 field descriptions), and Section 44.5.5, Trimming Register 3 (TRIM3), Table 44-7 (TRIM3 field descriptions), changed "Internal Regulation Mode" to "Internal VREG mode" and "External Regulation Mode" to "External VREG mode" for consistency.
46 Reset Generation Module (MC_RGM)	In Section 46.2, External signal description, corrected “...bidirectional reset pin RESET and the boot mode pins etimer0_ETC[3] and etimer1_ETC[4:3]” to “...bidirectional reset pin RESET and the boot mode pins etimer0_ETC[4:3] and etimer1_ETC[3]”. In Section 46.3.1.1, Functional Event Status Register (RGM_FES), • Added “"FCCU hard reaction request" and "FCCU soft reaction request" correspond to "long reset reaction" and "short reset reaction" of the FCCU module, respectively.” • In Table 46-3 (RGM_FES field descriptions), for the F_SOFT_FUNC, added “This bit is set when a mode change request has been made with the target mode = “0000””. In Section 46.3.1.2, Destructive Event Status Register (RGM_DES), replaced the existing NOTE with “NOTE - The F_POR flag is also set when a low voltage is detected on the 1.2 V supply, even if the low voltage is detected after power-on has completed. - The F_LVD27_VREG flag may still have the value “0” after a dip has occurred on the 2.7 V supply during a non-monotonic power-on sequence. The F_POR flag will, however, still be set in this case as expected after each power-on sequence. - The F_HVD12 flag may still have the value “0” after an overshoot has occurred on the 1.2 V supply during a non-monotonic power-on sequence. The F_POR flag will, however, still be set in this case as expected after each power-on sequence.” In Section 46.3.1.3, Functional Event Reset Disable Register (RGM_FERD), changed “Each byte can be written only once after power-on reset.” to “Second and third bytes of this register can be written only once after power-on reset.” In Section 46.4.6, Boot mode capturing, corrected “The MC_RGM samples etimer0_ETC[3] and etimer1_ETC[4:3] .” to “The MC_RGM samples etimer0_ETC[4:3] and etimer1_ETC[3]”.
48 System Integration Unit Lite (SIUL)	In Section 48.5.2.1, MCU ID Register 1 (MIDR1), Table 48-3 (MIDR1 field descriptions) • For the PKG field, added “257MapBGA = 01000, 473MapBGA = 11101”. • For the MAJOR_MASK field, added “(Cut 2.0) Major Mask = 0001“. • For the MINOR_MASK field, added “Minor Mask = 0000“. In Section 48.5.2.2, MCU ID Register 2 (MIDR2), Table 48-4 (MIDR2 field descriptions) • For the FLASH_SIZE_2 field, added “(MPC5675K)” after the description of value “0000” to show default. • For the EE field, added “(MPC5675K)” after the description of value “1” to show default. • For the FR field, added “(MPC5675K)” after the description of value “1” to show default. In Section 48.5.2.6, Interrupt Falling-Edge Event Enable Register (IFEER), Table 48-27 (IFCPR field descriptions), changed “internal oscillator” to “IRC clock” for consistency.
50 Self-Test Control Unit (STCU)	In Section 50.1.2.1, Reported errors, added the following bullets: - Read the STCU_MBSH flag register to determine which MBISTs failed. - Read the STCU_MBEH flag register to determine which MBISTs did not finish.

Table A-3. Changes between revisions 7 and 8 (continued)

Chapter	Description
53 Temperature Sensor (TSENS)	In Section 53.6.2, Equations for converting TSENS voltage to device temperature • Corrected Equation 1 (changed “C1 - C2” to “C2- C1”). • Corrected Equation 6 (moved "(ratioT - ratioCold)" from the denominator to the numerator of the division).
A Memory Map	In Table A-1 (MPC5675K module base addresses) and Table A-2 (MPC5675K detailed register map), added register entries for CMU, PMU, and OSC.
B Revision History	Updated for changes in Rev. 8.

A.4 Changes between revisions 6 and 7

Revision 7 uses change bars (vertical lines in the page margins) to indicate where technical (non-editorial) changes occurred.

Table A-4. Changes between revisions 6 and 7

Chapter	Description
Throughout	Editorial changes and improvements.
1 Introduction	In Section 1.1, The MPC5675K microcontroller, changed “targeting ISO26262” to “capable of being used in an ISO26262”. In Table 1-1 (MPC5675K family device comparison): • Deleted references to the 289-pin package. • Revised the DSPI entry to reflect the proper number of chip selects on MPC5675K and MPC5674K. • Revised the FlexRay entry (was optional for all chips, is present on MPC5675K and optional on the others). • Deleted the “Clock output” entry. In Figure 1-1 (MPC5675K block diagram), added SWT_0 and SWT_1. In Section 1.3, Feature summary, deleted “Replicated 32 channel eDMA controller”. In Section 1.4.11, DRAM controller, deleted “DDR 2 (optional)”. Revised Section 1.4.14, Deserial Serial Peripheral Interface (DSPI) modules, to reflect the accurate number of available chip selects.
3 Signal Description	In Section 3.2.1, Multiplexed pins, changed “ebi_D” to “ebi_AD”.
4 Functional Safety	In Section 4.1, Overview, added a note to see the Safety Manual (SM).
8 Flash Memory	In Figure 8-1 (Flash memory subsystem architecture), added addresses for DFM0. In Figure 8-13 (Platform Flash Configuration Register 0 (PFCR0)—LSM), marked bits 17–23 as reserved. In Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO), deleted field ST (is reserved with a reset value of 00). Extensive revisions to Section 8.2.5, Test sector.
10 Peripheral Bridge (PBRIDGE)	In Section 10.4.2, Memory map: • Changed “memory map organization” to “module organization”. • Renamed Table 10-1 (PBRIDGE module organization) (was “PBRIDGE module base addresses”) and added the “Peripheral connections” column to explain the allocation of peripherals in LSM and DPM.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
11 Analog-to-Digital Converter (ADC)	In Section 11.4.2, Analog channel conversion, replaced the values for “Minimum conversion time” and “Minimum sampling time” with references to the MPC5675K data sheet. In Section 11.4.4, ADC sampling and conversion timing: • Changed “(not including external multiplexing)” to “(excluding external multiplexing and storing of converted data in the data register)”. • Deleted the “Max AD_ck frequency and related allowed configuration settings” and replaced it with a reference to the MPC5675K data sheet. In Table 11-49 (Test channel activation), deleted the entry for CTU trigger mode. In Section 11.4.11.3, CTU mode: • Changed “Step1: VDD/VREF test” to “Step1: VDD/VBGAP test”. • Deleted [gnd]. In Section 11.4.11.5, Self-test analog watchdog, for algorithm S Step1, changed “if integer part lies in the range” to “if integer part lies outside the range”.
13 Clock Architecture	In Table 13-1 (MPC5675K clock signals), changed the minimum frequency for CLK_OUT (was 10 MHz, is 2 MHz). In Section 13.3.5, Auxiliary clock dividers, added a divide-by-16 option. In Section 13.4.2, Clock domains and clock tree, changed “functional channels” to “lakes of the SoR”. In Section 13.5.5, IPS bus clock sync bridge, changed “IPS Bus Sync Bridge” to “PBRIDGE”.
15 Clock Monitor Unit (CMU)	In Section 15.3.1.2, Frequency meter, added “When FXOSC > FIRC and MDR=0xFFFFF, the FDR overflows and the value stored cannot be relied upon. At the IP level, the frequency relationship between the measured and reference frequency in not known and may vary across products. Based on the frequency relationship in a particular product, the application must decide the appropriate value of MDR to avoid rollover of FDR. MDR=1 is not allowed as the FDR counter counts from 0 to MDR–1.”

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
18 Cross-Triggering Unit (CTU)	Renamed the CRU_ADC_R field (is CTU_ADC_R) throughout the chapter. In Figure 18-1 (CTU block diagram), deleted ETIMER0_TRG and added ETIMER2_TRG, ETIMER3_TRG, and ETIMER4_TRG. In Section 18.3, CTU overview: • Deleted “The CTU interfaces to the following peripherals” and the subsequent list. • Changed “can detect either a rising or a falling edge” to “can detect a rising edge, a falling edge, or both edges”. • Revised the list beginning with “The CTU comprises the following”. In Table 18-2 (CTU memory map) and Figure 18-28 (Cross Triggering Unit Count Range Register (CTU_CNT_RANGE)), changed the offset of CTU_CNT range (was 0x00D0, is 0x00D2). In Table 18-2 (CTU memory map), changed the starting offset of the last reserved space (was 0x00CE, is 0x00D6). In Table 18-3 (CTU register buffering and synchronization): • For CTUDF, changed the value in the “Double-buffered” column (was Yes, is No). • Changed the entry at 0x00CC (was for CTUPCR, is for CTU_EXP_A). In Table 18-4 (TGSISR field descriptions), changed “eTimer_2” to “ETIMER1_IN” and “eTimer_1” to “ETIMER0_IN”. In Table 18-6 (MRS_SM encoding), changed “eTimer_2” to “ETIMER1_IN” and “eTimer_1” to “ETIMER0_IN”. In Section 18.4.1.5, TGS Counter Reload Register (TGSCRR): • Corrected the register abbreviation (was TGRCRR). • Added “It is used to reload the counter when MRS (in triggered mode) or ES (in sequential mode) is one. The value of this register is load in the motor control domain after the master reload and is used only after an other master reload.”. In Section 18.4.1.6, Commands List Control Register 1 (CLCR1) and Section 18.4.1.7, Commands List Control Register 2 (CLCR2), added “It selects one of the 24 CLR commands as the first command when a trigger is generated and sends it to the ADC”. In Section 18.4.1.10, Commands List Register x (CLRx), deleted the duplicate paragraph. In Figure 18-17 (FIFO Status Register (FST)), changed the reset values of bits EMP3, EMP2, EMP1, and EMP0 to 1. In Table 18-20 (FST field descriptions), replaced “interrupt” with more specific information about the conditions. In Section 18.4.1.17, Cross Triggering Unit Error Flag Register (CTUEFR), revised the definition of the register. In Figure 18-21 (Cross Triggering Unit Interrupt Flag Register (CTUIFR)), changed all fields to w1c. In Figure 18-24 (Cross Triggering Unit Control Register (CTUCR)), changed the T*n*_SG, CTU_ADC_R, and MRS_SG fields to appear as read/write. In Table 18-27 (CTUCR field descriptions): • For the T*n*_SG, CTU_ADC_R, and MRS_SG fields, added “You may program this field to 1, but it will always read 0”. • For the DFE and GRE fields, added “You may read this field and program it to 1; other operations are not allowed.”. In Section 18.5.1, Interaction with other peripherals, revised the bulleted lists for CTU_0 and CTU_1.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
18 Cross-Triggering Unit (CTU) (continued)	In Figure 18-29 (CTU interaction with other peripherals): • Changed OUT_TRIGO to OUT_TRIG0. • Changed OUT_TRIGI to OUT_TRIG1. • Changed the duplicate PWM_ODD_3 to PWM_ODD_2. In Figure 18-30 (TGS in triggered mode), replaced the inputs to the OR gate with one 32-line muxed input. In Figure 18-32 (TGS in sequential mode): • Replaced the inputs to the OR gate with one 32-line muxed input. • Replaced the inputs to the “Master Reload Selection” block with one 32-line muxed input. In Section 18.6, Scheduler sub unit (SU), revised the bulleted list of SU outputs. Revised Figure 18-35 (Scheduler sub unit) to reflect the four outputs ETIMER1_TRG, ETIMER2_TRG, ETIMER3_TRG, and ETIMER4_TRG. In Section 18.9.2, CTU faults and errors, changed “user ensures no trigger event occurs during another one is processed, but if user makes a mistake and a trigger event occurs when another one is processed, the incoming trigger event will be lost and an error occurs” to “the user must ensure that no trigger event will occur while the current trigger event is being processed—otherwise, the incoming trigger will be lost and an error will occur”. In Section 18.9.3, CTU interrupt/DMA requests, added abbreviations to each interrupt.
19 Enhanced Direct Memory Access (eDMA)	In Section 19.1.2, Features: • Changed “send an interrupt request to the CPU” to “send an interrupt request to the INTC”. • Changed “Peripheral-paced hardware requests (one per channel)” to “Peripheral hardware requests (one per channel, assigned in the DMA Channel Mux)”.
Chapter 21 Deserial Serial Peripheral Interface (DSPI)	Revised Table 21-2 (CS Lines implemented on device) to show the correct implemented CS lines for the DSPI modules.
22 External Bus Interface (EBI)	In Section 22.1.3.1, Single master mode, added “External master mode is not supported on this chip.”. In Table 22-5 (EBI_MCR field descriptions), added the note “External master mode is not supported on this chip” to the EXTM field description.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
23 Error Correction Status Module (ECSM)	In Section 23.2, Features: • Removed the text “platform“ and “if error correcting codes (ECC) are implemented” from the “Registers for capturing information“ bullet. • Removed the text “for certain processor core micro-architectures” from the “Access address information“ bullet. Added the following registers to Table 23-2 (ECSM registers), Table 23-3 (ECSM 32-bit memory map), and Section 23.3.2, Registers description: • PLAMC • PLASC In Table 23-2 (ECSM registers), changed the reset value of MRSR (was 0x00, is 0x40). In Section 23.3.2.2, Revision (REV) register, changed the description to state only “The REV register specifies a revision number. It can only be read from the IPS programming model. Any attempted write is ignored.” In Table 23-5 (REV field descriptions), deleted “specified by an input signal to define”. In Section 23.3.2.6, Miscellaneous Reset Status Register (MRSR), revised the register definition. In Section 23.3.2.7, Miscellaneous Interrupt Register (MIR), changed “ECSMIR” to “MIR” in the text. In Section 23.3.3, ECC registers: • Changed “There are a number of program-visible registers” to “There are a number of registers.” • Removed the text “If the design does not include ECC on the memories, these addresses are reserved locations within the ECSM’s programming model.“ In Section 23.3.3.1, ECC Configuration Register (ECR), revised the introductory text to state only “The ECR is an 8-bit control register for specifying which types of memory errors are reported to the FCCU”. In Table 23-12 (ECR field descriptions), changed “Reporting” to “Reporting to the FCCU” and deleted the text “The occurrence of.... REDR registers” in all field descriptions. In Table 23-13 (ESR field descriptions): • Changed EPR1BR to ER1BR. • Changed EPF1BR to EF1BR. In Section 23.3.3.2, ECC Status Register (ESR): • Changed “ECSM_ECC2BIT_IRQ” to “ECSM_IRQ35” and “ECSM_ECC_IRQ” to “ECSM_IRQ36”. • Replaced the first three notes with “The ECSM also provides ECC error injection registers through which non-correctable ECC errors can be injected in the RAM. The injection of errors via these register would trigger a non-correctable ECC interrupt (IRQ35). Since the non-correctable error occurs in RAM/flash memory, the reaction is determined by the RGM_FERD[D_FL_ECC_RCC] bit. Depending on this bit, the ECC event may cause a reset, safe mode entry, or an interrupt.”
24 Enhanced Motor Control Timer (eTimer)	In Section 24.1.2, Features, added “Interrupt generation”. In Section 24.3.3.3, Capture Register 1 (CAPT1), changed “the register has been emptied” to “a register read occurs to empty a spot in the FIFO”. In Section 24.3.3.4, Capture Register 2 (CAPT2), changed “the register has been emptied” to “a register read occurs to empty a spot in the FIFO”. In Table 24-10 (CTRL1 field descriptions): • Revised the CNTMODE descriptions for values 001 and 101. • For SECSRC, added “A counter may not select its own output as the secondary source.” In Table 24-11 (CTRL2 field descriptions): • For VAL, added “or when the OFLAG is forced from another channel due to the COFRC bit being set.” • For FORCE, changed the last sentence to a note. In Figure 24-13 (Status Register (STS)), changed all fields to w1c.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
25 Fault Collection and Control Unit (FCCU)	In Table 25-1 (Abbreviations), added NCT and HNSHK. In Section 25.2, Main features: • Changed “33 critical faults” to “37 critical faults”. • Changed “21 non-critical faults” to “18 non-critical faults”. In Section 25.3, Block diagram, changed “software watchdog timer (SWT)” to “WDOG”. In Table 25-2 (FCCU memory map), changed the reset value for FCCU_CFG (was 0x0000_0000, is 0x0000_0008). Restored the FCCU_SCFS register to Table 25-2 (FCCU memory map) and Section 25.7.18, FCCU SC Freeze Status Register (FCCU_SCFS). In Table 25-3 (FCCU_CTRL field descriptions), for OPR=01101, changed “refer to the freeze registers” to “refer to the FCCU_SCFS register”. In Figure 25-4 (FCCU Configuration Register (FCCU_CFG)), changed the reset value (was a fixed value, is dependent on the NVM interface). In Table 25-5 (FCCU_CFG field descriptions), revised the FOP description to include FOP up to 11 1111. Changed the title of Table 25-7 (FCCU critical faults assignment (CF 0 to 39)) (was “FCCU critical faults assignment (CF 0 to 31)”) and: • Changed the CF[j] numbers for FCCU_CF_CFG1 (were numbered CF 0..5, are CF 32..37). • Revised the last entry (was “CF 6:63”, is “CF 38–63”). In Section 25.7.8, FCCU CF Status Register n (FCCU_CFSn): • Removed the reference to Table 25-12. • Changed “status/ (flag) bit CFSx” to “FCCU_CFSn[CFSx] field”. In Figure 25-14 (FCCU CF Status Register 1 (FCCU_CFS1)), deleted fields CFS40–CFS63 (are reserved). In Section 25.7.9, FCCU CF Key Register (FCCU_CFK): • Changed “Write the CFCK key” to “Write the CFK key”. • Changed “status (flag) bit CFSx” to “FCCU_CFSn[CFSx] field”. In Section 25.7.10, FCCU NCF Status Register 0 (FCCU_NCFS0), changed “status (flag) bit NCFSx” to “FCCU_NCFS0[NCFSx] field”. In Section 25.7.11, FCCU NCF Key Register (FCCU_NCFK): • Changed “status (flag) bit NCFSx” to “FCCU_NCFS0[NCFSx] field”. • Changed “to reach” to “for each”. In Section 25.7.12, FCCU NCF Enable Register 0 (FCCU_NCFE0), changed “NCF[*j*]$_0$” to “NCF[*j*]”. In Table 25-17 (FCCU_NCF_TOE0 field descriptions), changed NCFE[*j*] to FCCU_NCFE0[NCFE*x*]. In Table 25-21 (FCCU_EINOUT field descriptions), changed “fccu_eout[1]” to “fccu_1]” and “fccu_eout[0]” to “fccu_0]”. In Table 25-24 (FCCU_CFF field descriptions), replaced the entire FCFC field description. In Table 25-25 (FCCU_NCFF field descriptions), replaced the entire FNCFC field description. In Section 25.7.21, FCCU IRQ Status Register (FCCU_IRQ_STAT), changed “external interrupt line “irq_misc_b”” to “external alarm interrupt line”. In Section 25.7.22, FCCU IRQ Enable Register (FCCU_IRQ_EN), changed “external interrupt line “irq_misc_b”” to “external alarm interrupt line”. In Section 25.8.1, Definitions, changed the definition of a hardware recoverable fault. In Figure 25-31 (FCCU state diagram), added a description to the ALARM-to-ALARM transition. In Section 25.8.3, Definition of critical signals, added the expansion of TCU (test control unit).

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
25 Fault Collection and Control Unit (FCCU) (continued)	In Section 25.8.4, Self-checking capabilities, changed “frozen in a status register” to “frozen in the FCCU_SCFS register”. In Section 25.8.9, Module outputs, changed “error out” to “FCCU[0] and [1]”. In Section 25.8.13, FCCU_F interface, changed “NORMAL to ERROR” to “NORMAL to FAULT”. In Table 25-34 (Dual-rail encoding), changed “fccu_eout[1:0]” to “fccu_1:0]”. In Section 25.8.13.2, Time switching protocol, changed “fccu_eout[1:0]” to “fccu_1:0]” and “fccu_eout[0]” to “fccu_0]”.
27 FlexCAN Module	Revised Figure 27-1 (FlexCAN block diagram) to show 544 bytes of MB storage and 128 bytes of ID mask storage. In Section 27.1.3, Modes of operation, replaced the description of Module Disable Mode. In Section 27.3.1, FlexCAN memory mapping, added introductory text. In Table 27-11 (MCR field descriptions), added the note “This bit must be written in Freeze mode only” to the following bits: • FEN • SUPV • WRN_EN • SRX_DIS • LPRIO_EN • IDAM In Table 27-12 (CTRL field descriptions), added the note “This bit must be written in Freeze mode only” to the following bits: • PRESDIV • RJW • PSEG1 • PSEG2 • LPB • SMP • TSYN • LBUF • LOM • PROPSEG In Section 27.3.4.8, Error and Status Register (ESR), changed “source of four interrupts” to “source of eight interrupts”. In Table 27-17 (ESR field descriptions): • For TX_WRN, changed “TX Error Counter” to “TX Error Warning”. • For RX_WRN, changed “Rx Error Counter” to “RX Error Warning”. In Section 27.3.4.10, Interrupt Flags 1 Register (IFLAG1), deleted the paragraph “When the MCR[AEN] bit is set...”. In Section 27.4.8.7, Module disable mode, changed “This low power mode is entered when the MCR[MDIS] bit is set. If the module is disabled during freeze mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the LPM_ACK bit and clears the FRZ_ACK bit.” to “This low power mode is entered when the MDIS bit in the MCR Register is asserted. If the module is disabled during Freeze Mode, it requests to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit.”
34 LIN Controller (LINFlexD)	Added Section 34.7.1.5, Overrun.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
39 Nexus Development Interface (NDI)	In Section 39.2.1, Overview, changed “IEEE-ISTO 5001-2001” standard to “IEEE-ISTO 5001-2003”. In Section 39.2.3.1, Reset, changed bullet “Registers default back to their reset values” to “Debug registers default back to their reset values”. In Section 39.2.3.5, Censored mode: • Changed “it should not be possible” to “it is not possible”. • Removed reference to internal signal “nex_disable asserted“. In Table 39-2 (NPC signal properties): • Deleted the “Pull” column. • Removed the footnote regarding pullup/pulldown devices. • Changed the footnote “MDO0 indicates when the internal BIST mode has been completed and development tools can access the MCU.” to “MDO0 indicates when the internal BIST mode has been completed and development tools can access the MCU. MDO0 is driven high until the BIST completes.”. In Section 39.3.2, Detailed signal descriptions, removed the text regarding internal clocks. Replaced all content in Section 39.3.2.3, MDO—Message Data Out. In Section 39.3.2.4, MSEO—Message Start/End Out, added “(in normal Nexus mode, on both edges of MCKO when DDR mode is configured)”. In Section 39.4, Nexus Master IDs: • Added cross-reference to Table 9-1 (Logical master IDs) to the DTC[19:16] text. • Removed the text regarding Decoupled Parallel Mode (DPM). Table 39-3 (Nexus Crossbar Slave SRAM Data Trace Monitor 0 and 1) is now the single table for both Data Trace Monitor 0 and 1 (removed the table that covered only Data Trace Monitor 1). In Section 39.6.5.1, Decoupled Parallel Mode (DPM): • Removed reference to internal signal “ext_multi_nex_sel signal“. • Removed “NASP” acronym. In Section 39.6.6, Lock Step Mode (LSM), removed “NASP” acronym. In Section 39.8, Crossbar slave port data trace module (NXSS), removed the redundant “All messages and register accesses are compliant with IEEE-ISTO-5001” text. In Table 39-23 (DTC field description), removed the LSM and DPM decoding values and added a cross-reference to Table 9-1 (Logical master IDs). In Section 39.8.5.3, Data Trace Messaging (DTM), added restriction that a volume of trace information that is too great for port bandwidth will affect data access traces. In Figure 39-26 (Data write/read message format), changed “TSTAMP” to “Not used”. In Section 39.8.5.5.2, DTM queueing, removed “programmable depth”.
44 Power Management Controller (PMC)	In Section 44.12, ADC measure channel, updated the ADC_CHANNEL_SEL field values and descriptions.
46 Reset Generation Module (MC_RGM)	Changed “core reset” to “core debug reset” throughout the chapter. In Table 46-3 (RGM_FES field descriptions), replaced the F_CORE field descriptions.
48 System Integration Unit Lite (SIUL)	In Section 48.4, External signal description, deleted “Each GPIO port communicates via 16 I/O channels. In order to use the pad as a GPIO, the corresponding Pad Configuration Registers (PCR) for all pads used in the port must be configured as GPIO rather than as the alternate pad function.”.
49 System Status and Configuration Module (SSCM)	In Table 49-1 (SSCM memory map) and Section 49.3.1, Register descriptions, deleted the SSCM_PSAR and SSCM_CLR registers. In Table 49-4 (SSCM_MEMCONFIG field descriptions), for JPIN, changed “Bits [21:12] contain” to “Contains”.

Table A-4. Changes between revisions 6 and 7 (continued)

Chapter	Description
50 Self-Test Control Unit (STCU)	In Section 50.2, STCU main features, added the expansion of IPS (Internal Peripheral System). In Figure 50-20 (STCU MBIST Critical FM High Register (STCU_MBCFMH)), changed bits 5–12 to reflect the correct field progression. In Figure 50-22 (STCU MBIST Stay-In-Reset FM High Register (STCU_MBSFMH)), changed bits 5–12 to reflect the correct field progression.
53 Temperature Sensor	In Table 53-1 (TSENS calibration constants): • Changed C2 address from 0x0008 to 0x0006. • Changed 150 °C temperatures to 145 °C. • Changed 25 °C temperatures to –40 °C. In Section 53.6.2, Equations for converting TSENS voltage to device temperature: • Updated the equation for constant factor K. • Changed ratios “R1”, “R2”, and “R” to “ratioHot”, “ratioCold”, and “ratioT”, respectively. • Updated the equation for device temperature T.
A Memory Map	In Table A-2 (MPC5675K detailed register map), deleted the SSCM_PSAR and SSCM_CLR registers.)
B Revision History	Updated for changes in Rev. 7. Added content explaining the use of clickable cross-references.

A.5 Changes between revisions 5 and 6

Revision 6 uses change bars (vertical lines in the page margins) to indicate where technical (non-editorial) changes occurred.

Table A-5. Changes between revisions 5 and 6

Chapter	Description
Throughout	Editorial changes and improvements.
1 Introduction	In Section 1.4.16, FlexCAN, deleted “Safety CAN features on 1 CAN module as implemented on MPC5604PSPC560P40”.
2 Memory Map	In Table 2-2, revised the block sizes (in KB) at the following addresses as indicated: • 0x0020_4000: Was 32496, is 496 • 0x0040_4000: Was 3568, is 4080 • 0x6000_0000: Was 524288, is 785408 • 0x8FF2_8000: Was 48, is 64 • 0x8FF4_C000: Was 224, is 850896 • 0xC3E7_8000: Was 32, is 1072 • 0xC3FF_8000: Was 288, is 981024 • 0xFFE7_0000: Was 64, is 576 • 0xFFF5_4000: Was 160, is 240 (adjacent reserved areas consolidated) In Table 2-2, in the SRAM entries that applied to MPC5673K, removed mention of MPC5675K.
6 e200z7 Core	In Figure 6-3, deleted footnote 3. In Section 6.2.1, Register set, changed “the supervisor mode programmer’s model DCRs.” to “the supervisor mode programmer’s model DCRs and PMRs.”

Table A-5. Changes between revisions 5 and 6 (continued)

Chapter	Description
7 General-Purpose Static RAM (SRAM)	Restored the legends in Figure 7-1 and Figure 7-2 to their proper size.
8 Flash Memory	Removed duplicate bullet items from Section 8.1.1, Features. In Table 8-2: • Changed the block size at 0x00C0_0000 (was 4088 KB, is 8 KB). • Changed the block size at 0x00C0_2000 (was 8 KB, is 4088 KB). In the register field descriptions (Table 8-4, Table 8-7, Table 8-8, Table 8-15, Table 8-28), deleted redundant textual information that is already represented in the register figure (examples: “This bit is cleared on reset”, “This field is read/write”). In Table 8-13, in the PFCR1[B1_RWWC] field description, changed “0xx Terminate any attempted read-while-write/erase with an error response” to “0xx Reserved. This bit combination causes a malfunction.”. In Section 8.2.4, Shadow sector, deleted the note “NVUSRO is implemented in the code flash memory modules, but not in the data flash memory module”.
11 Analog-to-Digital Converter (ADC)	In Section 11.4.11.6, “Watchdog timer”, added the following note: “To prevent an interrupt request that occurs after enabling a watchdog timer (WDTE) and before the start of ADC self test (NSTART) from delaying the ADC self test start, it is possible to swap the order of WDTE and NSTART programming. Specifically, the programmer can use the following sequence of NSTART and WDTE programming: - Start the normal conversion by setting MCR[NSTART] - Enable watchdog timer by setting STAWxR[WDTE] - Freeze the above settings by writing MCR[STCL] (configuration lock)”
22 External Bus Interface (EBI)	In the figures in Section 22.4.2, External bus operations, changed “DATA[16:3” to “DATA[16:31]”.
23 Error Correction Status Module (ECSM)	In Section 23.3.2.3, IPS Module Configuration (IMC) register, changed “32 low-order IPS” to “11 low-order IPS”. In Section 23.3.2.6, Miscellaneous User-Defined Control Register (MUDCR), deleted the duplicated register figure and table.
32 Interrupt Controller (INTC)	In Table 32-10: • For IRQ 156, changed “Reserve FLAG[14]” to “Reserve FLAG[15]”. • Changed the IRQ at offset 0x0FC0 (was 225, is 252).
35 Mode Entry Module (MC_ME)	In Section 35.4.3.3, Peripheral clocks disable, Section 35.4.3.5, Processor and system memory clock disable, Section 35.4.3.7, Flash modules switch-on, and Section 35.4.4, Protection of mode configuration registers, changed WARNING to CAUTION.
37 Multi-Port DRAM Controller Priority Manager (PRIOMAN)	Added PERF_MNTR_ADDR, PERF_MNTR_READ_CNTR , and PERF_MNTR_WRITE_CNTR registers to Table 37-1. Added complete register names and abbreviations in the following sections: • Section 37.3.2.1, Priority Manager Configuration 1 Register (PRIOMAN_CONFIG1) • Section 37.3.2.2, Priority Manager Configuration 2 Register (PRIOMAN_CONFIG2) • Section 37.3.2.3, High Priority Configuration Register (HIPRIO_CONFIG) • Section 37.3.2.4, Look-up Table n Main Upper Registers (LUT0–LUT2) • Section 37.3.2.5, Look-up Table n Main Lower Registers (LUT0–LUT2) Added Section 37.3.2.11, Performance Monitor 1 and 2 Address Registers, Section 37.3.2.12, Performance Monitor Read Counter Register, and Section 37.3.2.13, Performance Monitor Write Counter Register. Replaced missing Figure 37-19.

Table A-5. Changes between revisions 5 and 6 (continued)

Chapter	Description
46 Reset Generation Module (MC_RGM)	In Section 46.3.1.3, Functional Event Reset Disable Register (RGM_FERD), changed WARNING to CAUTION. In the note in Section 46.3.1.6, “Functional Event Short Sequence Register (RGM_FESS)”, changed “set to 0x0000” to “set to 0x2000”.
B Revision History	Updated for changes in Rev. 6. Revised Table B-3 to reflect all changes actually present in Rev. 4. Revised Table B-2 to reflect all changes actually present in Rev. 5. Added a note explaining references to sections, figures, and tables. Added content explaining the use of change bars in Rev. 6.

A.6 Changes between revisions 4 and 5

Table A-6. Changes between revisions 4 and 5

Chapter	Description
Throughout	Editorial changes and improvements.
Preface	Added explanation about “cut1” and cut2” device versions.
1 Introduction	In Table 1-1: • In the ADC entry, added information for the 289-pin package. • In the Packages entry, added “289 pins”. In Section 1.4.16, FlexCAN, changed “MPC5604P” to “MPC5604PSPC560P40”.
2 Memory Map	In Table 2-1, corrected “EBI” at address 0x4000_0000 to “SRAM. “
3 Signal Description	In Section 3.2.3, System pins, deleted “289 MAPBGA” from the introductory sentence.
5 Boot and Operating Modes	Added sentence to the end of Section 5.6, Selecting LSM or DPM: “This could cause unexpected results to occur if the Core_1 startup code does not take short reset into account. Core_1 code can check the last reset source using RGM flags. If it finds that the reset was due to short reset, then it can poll for a Core_0 signal to wait before executing further code. “
7 General-Purpose Static RAM (SRAM)	Shortened the legend in Figure 7-1.
8 Flash Memory	In Table 8-13, in B1_P1_BFE and B1_P0_BFE bitfield descriptions, changed “128-bit” to “32-bit.” Added note to Section 8.2.4.1, Nonvolatile Private Censorship Password 0 Register (NVPWD0): These bits form the lower 32 bits of the Censorship password. Added note to Section 8.2.4.2, Nonvolatile Private Censorship Password 1 Register (NVPWD1): These bits form the upper 32 bits of the Censorship password.
11 Analog-to-Digital Converter (ADC)	Updated the conversion sequence in Section 11.4.11.6, Watchdog timer.
14 Clock Generation Module (MC_CGM)	In Figure 14-6, System Clock Divider Configuration Register 1 (CGM_SC_DC1), corrected DE0 and DIV0 to DE1 and DIV1.

Table A-6. Changes between revisions 4 and 5 (continued)

Chapter	Description
23 Error Correction Status Module (ECSM)	In Section 23.3.2.6, Miscellaneous User-Defined Control Register (MUDCR): • Added DRCLRn bits to Figure 23-6 and Table 23-9. • Marked the FXSBE6-4 and FXSBE1 bits as reserved in Figure 23-6.
24 Enhanced Motor Control Timer (eTimer)	In Figure 24-5, Figure 24-6, and Figure 24-8, changed “Access: User read/write” to “Access: User read-only”.
25 Fault Collection and Control Unit (FCCU)	In Section 25.6.4, FCCU CF Configuration Register n (FCCU_CF_CFGn), changed sentence “In the MPC5675K device at boot time, all critical faults are configured as hardware recoverable.” to “All the critical faults are software recoverable by default.” In Figure 25-6, changed bits 0–23 (CFC63:CFC40) to reserved.
32 Interrupt Controller	In Section 32.5.5.2, Ensuring coherency, added procedure for non-OSEK applications.
33 JTAG Controller (JTAGC)	In Table 33-2, Device Identification Register field descriptions: • In the DC field description, changed “Freescale Austin” to “MPC5675K”. • In the PIN field description, added cut1 and cut2 values.

Table A-6. Changes between revisions 4 and 5 (continued)

Chapter	Description
44 Power Management Controller (PMC)	In Figure 44-1, PMC block diagram, changed “LDO 3.3 V Regulator” to “LVD VDDFLASH.” In Section 44.4, External signals description: • Changed description of VDD_HV_ADV from “ADC voltage supply” to “ADC voltage 3.3 V supply” • Changed description of VDD_HV_PMU from “Voltage regulator supply” to “Voltage regulator 3.3 V or 5.0 V supply” In Table 44-3, CFGR field descriptions: • In HVD_1p2_CORE_EN bitfield, changed “1.2V Core HVD Enable” to “VDD_LV_COR 1.2V Core HVD Enable.” • In HVD_1p2_CORE_EN bitfield, changed “LVD” to “HVD.” • In LVD_ADC3p3_EN bitfield, changed “ADC LVD Enable” to “VDD_HV_ADV LVD Enable.” In Section 44.7, VRC SMPS controller, removed the sentence “and FM modulation is used to reduce the EM emission at 1 MHz.” In Section 44.9, LVD core: • Removed paragraph “The assertion and negation voltages are adjustable via software by writing to the LVDC_TRIM field of the TRIMR register, which selects one of the 64 voltages available through the appropriate tapped output. The reset value of the 6-bit register is "00_1000", corresponding to the nominal LvdC voltage.” • Changed “corresponding ADCM control register” to “CFGR[ADC_CHANNEL_SEL] bitfield.” In Section 44.10, HVD core: • Removed paragraph “The assertion and negation voltages are adjustable via software by writing to the HVDC_TRIM field of the TRIMR register, which selects one of the 64 voltages available through the appropriate tapped output. The reset value of the 6-bit register is "01_0111", corresponding to the nominal HvdC voltage.” • Changed “corresponding ADCM control register” to “CFGR[ADC_CHANNEL_SEL] bitfield.” In Section 44.11, LVD 3.3 V: • Removed paragraph “The assertion and negation voltages are adjustable via software by writing the corresponding supply field of the TRIMR register (LVDADC_TRIM, LVDFLASH_TRIM, LVDIO_TRIM, LVDREG_TRIM), which selects one of the 16 voltages available through the appropriate tapped output. The reset value of the 4-bit register is "1111", corresponding to the nominal Lvd33 voltage.” • Changed “corresponding ADCM control register” to “CFGR[ADC_CHANNEL_SEL] bitfield. In Section 44.12, ADC measure channel: • Changed “ADCM control register” to “CFGR[ADC_CHANNEL_SEL] bitfield.” • Changed “each ADC (1, 2, 3, and 4)” to “ADC[0:3].”
49 System Status and Configuration Module (SSCM)	Updated Table 49-4 to reflect cut1 and cut2 identification.
B Revision History	Updated for Revision 5.

A.7 Changes between revisions 3 and 4

Table A-7. Changes between revisions 3 and 4

Chapter	Description
Throughout	Editorial changes and improvements.
1 Introduction	In Section 1.4.11, DRAM controller, deleted the following features: • Supports three 64-bit-wide AHB slave ports • DRAM interface clock speed up to 90MHz
2 Memory Maps	Added Table 2-1 (MPC5675K system memory map overview). Updated Table 2-2 to correctly show reserved flash memory areas. In Table 2-2: • Changed “MMn” to “CFMn”. • For location 0x0020_4000, changed the size from 496 to 32496. • Collapsed the reserved entries in locations 0x0040_4000 through 0x0080_0000 into a single entry (size 3568). • Changed “Data flash array n” to “DFMn”. • For location 0xC3F8_C000, changed “Data Flash 0 Configuration” to “DFM0 Data flash 0 Configuration”.
3 Signal Description	In Figure 3-1, for pins C4 and R2, changed “fccu0 F” to “FCCU_F”. In Figure 3-2, for pin C4, changed “fccu0 F” to “FCCU_F”. In Figure 3-3, for pin AA2, changed “fccu0 F” to “FCCU_F”. Added footnote “Do not connect pin directly to a power supply or ground” for MDO[0:15] and MSEO[0:1] pins to Table 3-1, Table 3-2, Table 3-7, and Table 3-8. In Section 3.2.3, System pins: • Added “289 MAPBGA” to the introductory sentence. • Changed “fccu0 F” to “FCCU_F”.
4 Functional Safety	Updated the first paragraph in the chapter to state: “The MPC5675K microcontroller offers a set of features to support using it for applications that need to fulfill functional safety requirements.”
5 Boot and Operating Modes	In Section 5.6, Selecting LSM or DPM, added new information about the external reset sequence.
6 e200z7 Core	In Section 6.1.1, Features, under “Load/store unit (LSU)” bullet, changed “Two-cycle load latency” to “Three-cycle load latency.” In Figure 6-3, added footnote 3.

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
8 Flash Memory	In Section 8.1, Introduction: • Rewrote the opening text. • Appended the text “Read-While-Write (RWW) is supported between different flash memory modules” with the exception "but not within a flash memory module". Updated features list in Section 8.1.1, Features. Moved the "Flash Subsystem Integration in LSM" and "Flash Subsystem Integration in DPM" figures to Section 8.2.7, Interaction in LSM and DPM In Table 8-12, PFCR0 field descriptions, changed B02_RWWC description “0xx Terminate any attempted read-while-write/erase with an error response.” to “0xx Reserved. This bit combination causes a malfunction.” Corrected ECC descriptions to show code flash memory as 64-bit +8-bit ECC and data flash memory as 32-bit+7-bit ECC. In Section 8.1.2.1, Flash memory user mode, changed "erase the flash module" to "erase memory within the flash memory module". In Section 8.2.1, Module memory map: • Changed the flash sector for Shadow locations beginning at 0x28_0000 from "B0TF" to "B1SH". • Changed the flash sector for Test locations beginning at 0x40_0000 from "B0SH" to "B0TF". • Changed the flash sector for Test locations beginning at 0x48_0000 from "B0TF" to "B1TF". • Added the entry showing Reserved locations beginning at 0x48_4000. • Changed the entry for locations beginning at 0xC0_0000 from "Test" to "Reserved" ("Block Size" not updated). • Changed the entry for locations beginning at 0xC0_2000 from "Reserved" to "Test" ("Block Size" not updated). In Section 8.2.2, Register overview, changed "Offset from FLASH_REG_BASE" to "Offset from FLASH_BASE". In Table 8-4, changed "f90_done, MCR[DONE], and ary_access are set low" to "MCR[DONE] is cleared". In Section 8.2.3.2, Low/Mid Address Space Block Locking Register (LML), added "LML is implemented on CFM0, CFM1, and DFM0". In Section 8.2.3.3, High Address Space Block Locking Register (HBL): • Added the note "HBL is implemented in the code flash memory modules". • Deleted the content regarding the "Test flash block" from the HLK3-0 entry of the field description table. • Deleted the content regarding "In the event that blocks are not present" from the HLK3-0 entry of the field description table. In Section 8.2.3.4, Secondary Low/Mid Address Space Block Locking Register (SLL), added "SLL is implemented on CFM0, CFM1, and DFM0." In Section 8.2.3.5, Low/Mid Address Space Block Select Register (LMS), added "LMS is implemented on CFM0, CFM1, and DFM0." In Section 8.2.3.7, Address Register (ADR): • Added "ADR is implemented on CFM0, CFM1, and DFM0.” • Added "Priority Level 4, MCR[EDC]=1, Address of first ECC Single-bit error correction" to the "AD[22:2]" entry of the field description table.

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
8 Flash Memory (cont.)	In Section 8.2.3.8, Platform Flash Configuration Register 0 (PFCR0), added "PFCR0 is implemented in the code flash memory modules, but not in the data flash memory module". In Section 8.2.3.9, Platform Flash Configuration Register 1 (PFCR1), added "PFCR1 is implemented in the code flash memory modules, but not in the data flash memory module". In Section 8.2.3.10, Platform Flash Access Protection Register (PFAPR): • Added "PFAPR is implemented in the code flash memory modules, but not in the data flash memory module". • Added a note explaining bit configuration in DPM versus LSM to the ARBM field description. In Section 8.2.3.11, User Test 0 Register (UT0), added "UT0 is implemented on CFM0, CFM1, and DFM0". In Section 8.2.3.13, User Test 2 Register (UT2), changed "Base + 0x0040" to "Base + 0x0044". In Section 8.2.3.14, User Multiple Input Signature Register 0 (UMISR0), added "UMISR0 is implemented on CFM0, CFM1, and DFM0." In Section 8.2.3.15, User Multiple Input Signature Register 1 (UMISR1), added "UMISR1 is implemented on CFM0, CFM1, and DFM0." Consolidated the two distinct “Shadow sector” sections into one, making the following changes to the “Shadow Sector Structure for Code Flash Module 0” tables in the process: • Removed the first instance of the table. • Revised the second instance of the table (Table 8-23) by changing locations 0x0000-0x0003 (were "Nonvolatile System Reset Configuration", are "Reserved") and adding reset values for locations 0x3DD8-0x3DE7 and locations 0x3E18-0x3E1B. In Section 8.2.4, Shadow sector, added the note "NVUSRO is implemented in the code flash memory modules, but not in the data flash memory module". In Section 8.2.4.5, Nonvolatile User Options Register (NVUSRO), added the statement "This register is mirrored in the SSCM_UOPS register. See Section 49.3.1.8, User Option Status (SSCM_UOPS) Register."

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
8 Flash Memory (cont.)	Restructured the content of Section 8.2.6, Functional description as follows: • In Section 8.2.6.1, Data flash memory module structure, merged the "Data Flash" subsection content into the "Data flash memory module structure" section. • In Section 8.2.6.2, Code flash memory module structure, merged the "Code Flash" subsection content into the "Code flash memory module structure" section. • In Section 8.2.6.3, User mode operation, moved the statement "Notice that no Read-While-Modify is available" to the end of the "Data flash memory" subsection and revised it (is "Read-While-Modify is not available, because the data flash memory module has only one block"). • In Section 8.2.6.5, Reset, changed "registers that require updating from Test block or KRAM information" to "registers that require updating from Test block". • Renamed the subsection titled "Power-On" to Section 8.2.6.4, Data flash memory module power-on. • In Section 8.2.6.4, Data flash memory module power-on, changed "between the two flash modules" to "between the two code flash memory and the data flash memory modules". • Renamed the subsection titled "Slave Mode" to Section 8.2.6.7, "Data flash memory module slave status. • Revised and provided more details in Section 8.2.6.7, Data flash memory module slave status. • Renamed the subsection titled "Sleep Mode (Low Power Mode)" to Section 8.2.6.8, Sleep mode. In Section 8.3.1.1, Word program and Section 8.3.1.2, Sector erase: • Changed the list item containing "MCR[DONE], f90_done and ary_access go low" to "MCR[DONE] goes low". • Changed the list item containing "Wait until the MCR[DONE] bit goes high. f90_done and ary_access also go high" to "Wait until the MCR[DONE] bit goes high". In Section 8.3.2.1, Array Integrity Self Check, changed the list item containing "the related password" to "the related password (0xF9F9_9999)". In Section 8.3.3, Error Correction Code (ECC): • Deleted the empty subsection "Code Flash ECC". • Deleted the empty subsection "Data Flash ECC". • In Section 8.3.3.1, ECC algorithms features, expanded the descriptions to include code flash memory. • Merged the subsection "Enhanced flagging" of the subsection "Bit Manipulation" into a single Section 8.3.3.2, “Bit manipulation”. Added note in Section 8.3.4.2, Censored mode, regarding reading the RCHW at boot time on a secured device.
11 Analog-to-Digital Converter (ADC)	In Table 11-28, added min/max values to INPSAMP_C, INPSAMP_RC, and INPSAMP_S bitfield descriptions.
12 Boot Assist Module (BAM)	In Section 12.6.7.2.1, Choosing the host baud rate, changed "customer will be required to ensure" to "software must ensure".

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
13 Clock Architecture	Provided cross-references to control registers in other chapters. Restructured and reorganized some sections. No technical changes made during reorganization. In Table 13-1: • Changed PERI0_CLK description “FlexCAN0, 1, 2, 3” to “FlexCAN0.” • Added DSPI2 and “FlexCAN1, 2, 3” to PERI1_CLK description. Added content to Section 13.2, Clock selection. Revised Table 13-2. Revised Section 13.3.5, Auxiliary clock dividers.
15 Clock Monitor Unit (CMU)	In Section 15.3.1.2, Frequency meter, added text: “The CMU_CSR register is writable only when register protection to the register is disabled. Hence, the user should take register protection into account if frequency meter needs to be used.”
18 Cross-Triggering Unit (CTU)	In Section 18.4.1.22, Cross Triggering Unit Digital Filter Register (CTUDF), updated the field description and changed field CTUDF[N] to CTUDF[FILTERVALUE] to match header file. In Section 18.4.1.23, Cross Triggering Unit Expected Value A Register (CTU_EXPECTED_A), changed “number of system clock cycles" to "number of system clock (SYS_CLK) cycles". In Section 18.4.1.25, Cross Triggering Unit Count Range Register (CTU_CNT_RANGE), changed "system clock cycle" to "system clock (SYS_CLK) cycles". In Section 18.6, Scheduler sub unit (SU), Updated paragraph to read: “External signals can be asynchronous with MOTC_CLK, the motor control clock. For this reason, a programmable digital filter is available. The external signal is considered at 1 if it is latched at 1 at a time of n + 1, and is considered at 0 if it is latched at 0 at a time of n + 1, where n is the value in the Cross Triggering Unit Digital Filter Register (CTUDF).”
21 Deserial Serial Peripheral Interface (DSPI)	In Section 21.3.2.4, DSPI Clock and Transfer Attributes Register n (DSPI_CTARn), updated slave mode to apply to DSPI_CTAR0 only.
22 External Bus Interface (EBI)	In Section 22.4.2, External bus operations, changed DATA[16:31] to DATA[16:3].
23 Error Correction Status Module (ECSM)	In Section 23.3.3.2, ECC Status Register (ESR), added the note "To generate IRQ 35, the non-correctable ECC interrupt RGM (IRQ35), FCCU reset and ECC reset should be programmed as SAFE mode entry and not as reset, using the FERD register. When a corrupted ECC location is read, IRQ35 + machine check exception + NMI exception is generated. Thus, the handlers need to be written in a way that all exceptions and interrupts are serviced properly, and the handlers bring the MPC5675K back to DRUN mode from SAFE mode and also clear all the flags, including flags set in the FCCU.". Modified the reset-related footnote (was applicable to the entire register, now is applicable only to bit 0) in the following sections: • Section 23.3.3.5, Flash ECC Master Number Register (FEMR) • Section 23.3.3.7, Flash ECC Data Register High/Low (FEDRH/L) • Section 23.3.3.8, RAM ECC Address Register (REAR) • Section 23.3.3.9, RAM ECC Syndrome Register (RESR) • Section 23.3.3.10, RAM ECC Master Number Register (REMR) • Section 23.3.3.11, RAM ECC Attributes (REAT) register • Section 23.3.3.12, RAM ECC Data Register High/Low (REDRH/L)

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
24 Enhanced Motor Control Timer (eTimer)	In Section 24.3.5.2, DMA Request Select Registers (DREQ*n*): • Updated DREQ*n* [DREQ*n*[4:0]] bitfield descriptions. • Added Note following DMA Request Select Registers (DREQ*n*) section: "**Note**: If the DREQ*n* registers contain the same value, then multiple DMA requests can be generated in response to a single flag/condition. To avoid this issue, ensure each of the DREQ*n* registers has a unique value prior to enabling DMA requests." In Section 24.4.1, General, changed "The counter can..." to "The counter/timer can...". In Section 24.7, eTimer Initialization, changed "DREQ1 and DREQ2 registers" to "DREQ0 and DREQ1 registers."
25 Fault Collection and Control Unit (FCCU)	Added "Reset Value" column to Table 25-2, FCCU memory map. Updated reset values in register figures. Updated FCCU_CF_CFG0 Critical Fault assignment description for CF 28:31 to "Reserved for internal test logic monitoring" in Table 25-7. Updated Table 25-7 and Table 25-9 to add Module and Source columns to provide additional critical and non-critical source information.
27 FlexCAN Module	Renamed Section 27.4.7.1, Rx FIFO Misalignment to Section 27.4.7.1, Precautions when using Global Mask and Individual Mask registers and updated contents.
28 Motor Control Pulse Width Modulator Module (FlexPWM)	In Table 28-26, replaced the internal core interrupt names with external names.
30 Frequency-Modulated Phase-Locked Loop (FMPLL)	In Section 30.5.1, Control Register (CR), marked bit 25 as "reserved."
31 Inter-Integrated Circuit Bus Controller Module (I^2C)	In Section 31.1, Introduction, removed sentence "The device is capable of operating at higher baud rates, as fast as a maximum of module clock/20, with reduced bus loading."
32 Interrupt Controller (INTC)	In Section 32.5.2, Interrupt exception handler, eliminated the "Software vector mode" and "Hardware vector mode" subsection titles. In Table 32-10, changed INT254 to "Not used."
33 JTAG Controller (JTAGC)	In Table 33-1, changed the footnote text "A weak pull may be implemented at the TDO pad for use when JTAGC is inactive" to "A weak pull may be required on TDO to prevent the signal from floating when tools are attached." In Section 33.4.4, Enabling debug of a censored device, clarified that the MPC5675K implements an enable bit in the CENSOR_CTRL register.
34 LIN Controller (LINFlexD)	In Section 34.9.1.2, 9-bit data frame, corrected description to reflect 9-bit frame. In Table 34-51, changed "$f_{ipg\_clock\_lin}$ = MHz" to "$f_{ipg\_clock\_lin}$ = 64 MHz".

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
35 Mode Entry Module (MC_ME)	In Table 35-3, updated ME_ME register to show RESET_DEST and RESET_FUNC bits. Added this NOTE: after Table 35-15: "After modifying any of the ME_RUN_PC0...7, ME_LP_PC0...7, and ME_PCTLn registers, software must request a mode change and wait for the mode change to be completed before entering debug mode in order to have consistent behavior between the peripheral clock control process and the clock status reporting in the ME_PSn registers." Added these bullets to the list in Section 35.4.4, Protection of mode configuration registers: • If PLL0 is on, XOSC must also be on. • If PLL1 is on, XOSC must also be on
37 Multi-Port DRAM Controller Priority Manager (PRIOMAN)	In Table 37-1, removed the following registers: • LUT table 3 Main Upper • LUT table 4 Main Upper • LUT table 3 Main Lower • LUT table 4 Main Lower • LUT table 3 Alternate Upper • LUT table 4 Alternate Upper • LUT table 3 Alternate Lower • LUT table 4 Alternate Lower • Granted ack counter 3 • Granted ack counter 4 • Cumulative wait counter 3 • Cumulative wait counter 4 • Summed priority counter 3 • Summed priority counter 4 Removed LUT_SEL_4, LUT SEL_3, ACK_COUNT4, and ACKCOUNT3 fields from Section 37.3.2.1, PRIOMAN_CONFIG1 Register. Removed ACK_SEL_4 and ACK_SEL_3 fields from Section 37.3.2.2, PRIOMAN_CONFIG2 Register.
39 Nexus Development Interface (NDI)	In Table 39-2, added "A weak pull up resistor may be required on TDO to prevent the signal from floating when tools are attached" to the third footnote. Corrected title of Figure 39-22.
40 Oscillators	In Figure 40-1, changed bit 25 (was reserved / read 0, is reserved / write 1 to clear). In Table 40-4, added information about RCTRIM bits.
41 Parallel Digital Interface (PDI)	In Table 41-4, changed MODE value 11 from BIST mode to TEST mode. In Figure 41-6, changed FO, FU, LC, FC, HE, and DD bits to "w1c". Updated Table 41-6 and Table 41-7 to provide better descriptions of bit status. In Figure 41-13, changed MC bit to R/W.
44 Power Management Controller (PMC)	In Table 44-3, revised the descriptions for LVD_3p3_MAIN_EN and LVD_1p2_CORE_EN.
46 Reset Generation Module (MC_RGM)	Changed instances of RESET_B to $\overline{\text{RESET}}$. In Section 46.3.1.5, Functional Event Alternate Request Register (RGM_FEAR), removed RGM_FEAR[AR_FL_ECC_RCC] AND RGM_FEAR[AR_FCCU_SAFE] bits. In the Note in Section 46.3.1.6, Functional Event Short Sequence Register (RGM_FESS), added the following paragraph: "However, short sequence resets should not be used when in a mode with the flash memory configured to a power-down state. In this case, a full PHASE1 reset is required in order to power up the flash memory . Therefore, the RGM_FESS register should be set to 0x0000."

Table A-7. Changes between revisions 3 and 4 (continued)

Chapter	Description
48 System Integration Unit Lite (SIUL)	In Table 48-2 and Section 48.5.2.8, Pad Configuration Registers (PCRn), changed reset value to “See Table 48-11.”
49 System Status and Configuration Module (SSCM)	In Figure 49-10, in footnote 2, changed "shadow block" to "Test flash" block. In Table 49-15, for the WATCHDOG_ENABLE field, added the note “In DPM, applies only to SWT_0; SWT_1 is always off until enabled by software.”
A Memory Map	In Table A-2: • Added registers for Priority Manager (Chapter 37, Multi-Port DRAM Controller Priority Manager (PRIOMAN)) • Changed the offset of the DREQ1 register (was 0x0114, is 0x0112).
B Revision History	Updated for Revision 4.

A.8 Changes between revisions 2 and 3

Table A-8. Changes between revisions 2 and 3

Chapter	Description
Throughout	Editorial changes and improvements.
Preface	Removed description of “read to clear” bit format.
1 Introduction	Updated feature descriptions: • Restated core frequency as 180 MHz. • Provided more details for 3.3 V supply voltages. • Corrected number of MMU TLB entries to 64. • EBI available on 473-pin package. • Corrected number of ADC input channels to 22.
3 Signal Description	Changed D15 (257 MAPBGA) from VPP_TEST to VSS_HV_IO. Changed D11 (473 MAPBGA) from VPP_TEST to VSS_HV_IO. Changed “fccu0 F[0]” to “FCCU_F[0]” and “fccu0 F[1]” to “FCCU_F[1]”.
4 Functional Safety	Removed references to *MPC5675K Safety Application Guide*. Added Section 4.3, Sphere of Replication (SoR). In Section 4.4, Built-In Self-Test (BIST), added sentence “BIST is a mechanism that permits a device to test itself. On the MPC5675K, BIST enables the reliability require BIST is a mechanism that permits a device to test itself. On the MPC5675K, BIST enables the reliability requirements necessary to achieve SIL3 and ISO26262.”
5 Boot and Operating Modes	In Section 5.3.2, Reset Configuration Half-Word (RCHW), removed sentence “This device series supports only VLE code, so for future compatibility the flag must always be set to 1.” Added Section 5.6, Selecting LSM or DPM.
6 e200z7 Core	Replaced significant portions of this chapter with updated material from the introduction to the updated e200z760n3 core manual. Core features are unchanged in silicon, but the description of these features is enhanced.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
7 General-Purpose Static RAM	Reduced memory speed specification to 90 MHz. In Section 7.2.1, Overview, removed bullet item “1-1-1-1 @ 110 MHz at 0 WS“ In Section 7.5, SRAM ECC mechanism, corrected “32-bit data bus” to “64-bit data bus.” In Section 7.6, Functional description, corrected “32-bit writes” to “64-bit writes.” Updated Section 7.7, Initialization and application information.
8 Flash Memory	Significant technical and editorial updates throughout the chapter. Users should consider this to be a new chapter. Only a partial list of changes is provided. Changed “fccu0 F[0]” to “FCCU_F[0]” and “fccu0 F[1]” to “FCCU_F[1]”. Updated Figure 8-1. In Section 8.1.1, Features, deleted list item “Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel.” In Section 8.2.3.1, Module Configuration Register (MCR), added EDC bit. In Figure 8-4 and Table 8-4, updated MAS[2:0] field description. In Section 8.2.3.8, Platform Flash Configuration Register 0 (PFCR0), added note to B02_RWWC Bank0+2 Read-While-Write Control bitfield. In Section 8.2.3.9, Platform Flash Configuration Register 1 (PFCR1), added note to B1_RWWC Bank1 Read-While-Write Control bitfield. Added Table 8-31, Test flash information.
9 Multi-Layer AHB Crossbar Switch (XBAR)	Relocated register information to near the start of the chapter. In Table 9-1, changed PROT[3] in footnote to HMASTER[3]. Changed reset values for MPRn registers to 0x0400_3210. Removed info regarding undefined length AHB bus burst accesses. Removed MGPCRn register. In Section 9.2.1, Register Summary, changed “four registers” to “two registers” and updated description. In Figure 9-2, removed HPE7, HPE5, HPE4 bits; marked corresponding PARK values in Table 9-9 as reserved. In Section 9.3.5, XBAR_2 Configuration, updated master parking description. Deleted Figure 9-4, XBAR Slave Port Block Diagram.
10 Peripheral Bridge (PBRIDGE)	Updated Table 10-3 to show reserved areas and correctly display MPROTn and PACRn implementation.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
11 Analog-to-Digital Converter (ADC)	Removed list of input channels from Section 11.1, Introduction. Updated MCR[WLSIDE] bit description. Deleted sentence 'Once selected, no mode change is possible unless a reset occurs." from Section 11.2.3, Inter-module communication. Removed this sentence from NSTART bit description in MCR register: "This bit stays high while the conversion is ongoing (or pending during injection mode)." and inserted for NSTART bit in MSR register. In Table 11-10, changed description of WDGxH bit to "This field indicates whether an interrupt has been generated because the converted value is higher than the programmed higher threshold." Clarified description of PSCR[PRECONV] bit in Table 11-16. In Figure 11-17, changed INPSAMP to INPCMP. Replaced the Note in Section 11.4.5, Programmable analog watchdog, with "Avoid the situation where THRH < THRL. In this case, a WDGxH or WDGxL interrupt will always be generated, and this could lead to misinterpretation of the watchdog interrupts." Changed STCR3[MSTEP] bitfield description to: "For One-Shot mode, this bitfield defines the current step for Algorithms S, RC, and C as follows: — For Algorithm S: MSTEP = 0 to 2 — For Algorithm C: MSTEP = 0 to 16 — For Algorithm RC: MSTEP = 0 to 18 For Scan mode, this field is not used and should be programmed to 0x00. This is because in Scan mode performs only single-step execution (interleaved mode)." Added this note: "If the MCR[OVERWR] bit is set, the STSR2-4 registers are overwritten if another error occurs." following STSR2–4 registers (Section 11.3.1.28, Self-Test Status Register 2 (STSR2)), Section 11.3.1.29, Self-Test Status Register 3 (STSR3), and Section 11.3.1.30, Self-Test Status Register 4 (STSR4). Updated THRH and THRL bit descriptions for registers STAW0R (Table 11-38), STAW3R (Table 11-42), and STAW4R (Table 11-43). Removed Section 11.4.2.2, Start of normal conversion. Added Section 11.4.11, Self-testing. Replaced the Note in Section 11.4.5, Programmable analog watchdog, with "Avoid the situation where THRH < THRL. In this case, a WDGxH or WDGxL interrupt will always be generated, and this could lead to misinterpretation of the watchdog interrupts." In Section 11.4.7, Interrupts, added EOCTU, REF_RANGE, and Self-test bullets to list of interrupts. Updated Table 11-49. Updated Equation 2. Added two paragraphs to the end of Section 11.4.7, Interrupts, describing End of Conversion interrupts and Self-testing error interrupts.
12 Boot Assist Module (BAM)	Removed ABD field from Figure 12-4, Table 12-5 and ABD bit references throughout the chapter. In Section 12.6.4.6, Download Start Address, VLE Bit and Code Size, added Note describing maximum allowable code size to be downloaded. In Section 12.6.4.7, Download Data, added paragraph describing maximum allowable code size to be downloaded.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
13 Clock Architecture	Inserted new Table 13-1, MPC5675K clock signals. Updated Figure 13-1, MPC5675K System Clock Generation. Updated Figure 13-3, MPC5675K System Clock Distribution Part B. In Section 13.4.5, IPS Bus Clock Sync Bridge, changed “For motor control applications an access delay for every access to these peripherals of up to 5 additional SYS_CLK cycles introduced by this bridge is acceptable” to “For motor control applications, an access delay for every access to these peripherals of up to 4 additional SYS_CLK cycles plus 3 MOTC_CLK cycles introduced by this bridge is acceptable” In Section 13.4.1, FlexCAN Clock Domains, changed 10 Mbit/ sec to 1.0 Mbit/sec. In Section 13.4.1, FlexCAN Clock Domains, added this sentence: “The EQUAL condition applies to CANPE_CLK selected by CLK_SRC = 1. The GREATER_THAN condition applies to XOSC_CLK selected by CLK_SRC = 0.” Updated Figure 13-2 to accurately depict Clock Sync Bridge.
14 Clock Generation Module (MC_CGM)	Added information about the CGM_SC_DC1 register. In the “MC_CGM Memory Map” table: • Modified the entry at 0xC3FE_0160..017C (was “EBI registers”, is “reserved”). • Added an entry at address 0xC3FE_039C (is reserved). • Revised the starting address of the last reserved space (was 0xC3FE_03A8, is 0xC3FE_03A4). Converted the following register figures from 32-bit format to 8-bit format: • CGM_OCDS_SC • CGM_SC_DC0 • CGM_AC0_DC0 • CGM_AC1_DC0 • CGM_AC2_DC0 In the CGM_AC2_DC0 section, changed “controls the auxiliary clock 2 divider” to “controls the auxiliary clock 2 divider 0”. Revised the “MC_CGM System Clock Generation Overview” figure. Revised the “System Clock Dividers” section. In the “Dividers Functional Description” section, added a cross-reference to CGM_SC_DC1.
15 Clock Monitor Unit (CMU)	Changed CK_IRC to IRCOSC throughout the chapter. Added Section 15.5, CMU Integration. Added these notes: • In Section 15.4.1, Control status register (CMU_CSR), “The CMU_CSR register is functional only on CMU_0.” • In Section 15.4.2, Frequency display register (CMU_FDR), “The CMU_FDR register is functional only on CMU_0.” • In Section 15.4.3, High-frequency reference register A (CMU_HFREFR_A), “The CMU_HFREFR_A register is functional only on CMU_0.” • In Section 15.4.4, Low-frequency reference register A (CMU_LFREFR_A), “The CMU_LFREFR_A register is functional only on CMU_0.” • In Section 15.4.5, Interrupt status register (CMU_ISR), “The CMU_ISR register is functional on all CMU instantiations.” • In Section 15.4.6, Measurement duration register (CMU_MDR), “The CMU_MDR register is functional only on CMU_0.”
17 Coherency Unit (CU)	In footnote of Table 17-11, changed “11 = 3 requests combined” to “11 = 4 requests combined.”

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
18 Cross-Triggering Unit (CTU)	Updated bullet lists at Section 18.5.1, Interaction with other peripherals, to list CTU interaction with peripherals. In CTUCR register, changed all bits of write type “S” to “w1c.” Updated FST bit descriptions in Table 18-20. Updated FRx[ADC] and FLx[ADC] bit descriptions in Table 18-21 and Table 12-22. Replaced Table 18-32, CTU interrupts. Updated CTUIFR[Safety_error_B] and CTUIFR[Safety_error_A] bit descriptions in Table 18-24. In Table 18-23, added Note: “If software is incorrectly programmed in Dual Conversion mode, such as by programming one or more ADC commands on the same physical channel (a channel that is shared between two ADCs, e.g. Channel 11 on ADC_0 and ADC_1), the Invalid Command Error is not detected and this bit is not set. Software must ensure that in case of Dual Conversion Mode, if the channel number is the same for both ADCs then this number should not represent a shared ADC channel.” to description of ICE bit. In Section 18.4.1.25, Cross Triggering Unit Count Range Register (CTU_CNT_RANGE), replaced description of VALUE bit with “Bits masked by this register field are ignored while comparing internal up counter to the CTU_EXPECTED_A/B register.” In Section 18.5.3, Trigger Generator Sub Unit (TGS), replaced sentence “The TGS Mode is selected using the TGS_M bit in the TGS Control Register.” with “The TGS mode can be toggled between Triggered and Sequential mode with the TGSCR[TGS_M] bit.” Added numerous bit and register descriptions throughout the chapter.
19 Enhanced Direct Memory Access (eDMA)	In Table 19-5, swapped GPE and CPE bit descriptions. Removed non-functional DMACR[EBW] bit.
20 eDMA Channel Mux (DMACHMUX)	In Table 20-5, updated eTimer source resource values.
21 Deserial Serial Peripheral Interface (DSPI)	Removed brackets from PCS pin names to match the terminology used in the rest of the manual.
22 External Bus Interface (EBI)	In Table 22-1, added row for ALE. Added Section 22.4.1.12, Address Latch Enable. Removed Section 22.4.2.9, Non-Chip-Select Burst in 16-bit Data Bus Mode.
23 Error Correction Status Module (ECSM)	Updated reset values for REDR, REAT, and REDR registers. Added module mappings to IMC register. Added cross-reference to list of XBAR Logical master IDs in REMR field descriptions. Changed AXBS to XBAR in IMC field descriptions table.
24 Enhanced Motor Control Timer (eTimer)	Corrected DREQ1 address in Table 24-2. Updated STS[WDF] bit description. Updated CAPT1 and CAPT2 register descriptions to describe FIFO fill policy. Removed “Resets” and “Clocks” sections. Added Note: to Section 23.4.2.4, GATED-COUNT Mode and Section 23.4.2.6, SIGNED-COUNT Mode Added initialization sequence at end of chapter.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
25 Fault Collection and Control Unit (FCCU)	Added sentences “It is important to safe state NPC signals before asserting JCOMP to the device since this may trigger the monitoring circuit. Users should disable Nexus tracing before JCOMP is asserted.” at Section 25.7.8.1, Monitoring JTAGC and NPC. Replaced EOUT with FCCU_F. Replaced “fccu0 F[0:1] interface” with “FCCU_F interface”. In the Introduction section: • Changed “via the system pins fccu0 F[0:1]” to “via output pins”. • Changed “(fccu0 F[0:1] signaling)” to “(FCCU_F signaling)”. Revised the “Block diagram” section. Added the “Glossary and acronyms” section. In the “Main features” section, changed “fccu0 F[0:1] pins” to “FCCU_F pins”. Added the “Signal description” section. In the “Register interface” section, changed: • data[31:24] to data[0:7] • data[23:16] to data[8:15] • data[15:8] to data[16:23] • data[7:0] to data[24:31] • data[31:16] to data[0:15] • data[15:0] to data[16:31] In the “FCCU memory map” table, revised the access information to include state-dependent information. In the register descriptions, changed “fccu0 F[0:1]” and “fccu_eout” to “FCCU_F”. Deleted the following registers: • FCCU_NAFS • FCCU_AFFS • FCCU_NFFS • FCCU_FAFS • FCCU_SCFS In the FCCU_CTRL[OPR] field descriptions, deleted the terms NA, AF, NF, FA, and SC. In the FCCU_CTRL figure, revised the reset values for the NVML and OPS fields (were 0, are dependent on whether the NVM interface is used). In the “Definitions” section, changed “resynchronized and latched” to “captured”. In the “FSM description” section, deleted “(irq_alarm_b)”. Revised the “Self-checking capabilities” section. In the “Reset interface” section, removed extraneous content and revised the “Reset sources” table. Revised the “Fault recovery” section. Revised the “WKUP/NMI” section. Revised the “STCU interface” section. Renamed the section “fccu0 F[0:1] interface” (is FCCU_F interface) and revised it. In the FCCU_CF_CFG[CFCx] field description, deleted “(input fault fccu_cf[j])”. In the FCCU_NCF_CFG0[NCFCx] field description, deleted “(input fault fccu_ncf[j])”. In the FCCU_CFS section, deleted “(fccu_cf[j])”. In the FCCU_MCS section, changed “MC state” to “chip mode” and added a cross-reference to the table that explains the encodings of the modes.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
25 Fault Collection and Control Unit (FCCU) (continued)	In Section 25.6.14 FCCU NCF Time-out Register (FCCU_NCF_TO), replaced "If the fault is not recovered within the time-out period, the FCCU moves from the ALARM state to the FAULT state." with "If the fault is not recovered within the time-out period, the FCCU moves from the ALARM state to the FAULT state. If fault recovery is intended before the FCCU moves to the FAULT state, then software must ensure that the FCCU_CFG[FCCU_NCF_TO] and FCCU_CFG[SMRT] bits are appropriately programmed to give enough time for core to respond to and clear the fault." In Table 25-10, Non-critical fault assignment for FCCU_NCF_CFG0, set NCF 22:24 to "reserved."
27 FlexCAN Module	In Section 27.1.2, FlexCAN Module Features, • Changed list item "Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator" to "Programmable clock source to the CAN Protocol Interface, either FlexCAN clock or XOSC clock." • Added list item "Compatible with the ISO 11898 standard." In Table 27-12, deleted the note from the CLK_SRC bit description. Changed the CLK_SRC bit description to "This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the FlexCAN clock or the XOSC clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See Section 27.4.8.4, Protocol Timing, for more information. 0 The FlexCAN engine clock source is the XOSC clock. 1 The FlexCAN engine clock source is the FlexCAN clock." In Section 27.4.8.4, Protocol Timing, • Replaced "the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a FMPLL)" with "the XOSC clock or the FlexCAN clock". • Replaced "crystal oscillator" with "XOSC". • Replaced the NOTE: text with "In order to guarantee reliable operation, the selected CAN Protocol Interface (CPI) clock should not be faster than the peripheral clock." In Figure 27-17, — Changed "Peripheral Clock (PLL)" to "FlexCAN Clock". — Changed "Oscillator Clock (Xtal)" to "XOSC Clock". In Section 27.4.8.5, Arbitration and Matching Timing, removed list item: "A valid CAN bit timing must be programmed, as indicated in Table 27-23" In Figure 27-4, corrected bit numbers for MSB = 0. In Table 27-11 and Figure 27-5, removed WAK_SRC bit. In Section 27.4.8.4, Protocol Timing, removed tolerance guideline (up to 0.1%). In Section 27.4.8.5, Arbitration and Matching Timing, deleted example. In Section 27.4.9, Interrupts, replaced the first paragraph with "The module can generate 7 interrupt sources (5 interrupts due to ORed message buffers and 2 interrupts due to ORed interrupts from Bus Off, Error, and Tx Warning, Rx Warning, and Wake Up). See Chapter 32, Interrupt Controller (INTC), for more information." In Section 27.4.10, Bus Interface, changed third bullet item to reflect 32 MBs and address restrictions for 32 MBs. Added Section 27.4.7.1, Rx FIFO Misalignment.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
28 Motor Control Pulse Width Modulator Module (FlexPWM)	Removed fractional delay feature descriptions. Removed HALT mode feature descriptions. Inserted the following note in Section 28.3.2.7, Value Register 2 (VAL2), and Section 28.3.2.8, Value Register 3 (VAL3), **NOTE**: The value in the VAL3 register must always be greater than or equal to the value in the VAL2 register. Inserted the following note in Section 28.3.2.9, Value Register 4 (VAL4), and Section 28.3.2.10, Value Register 5 (VAL5), **NOTE**: The value in the VAL5 register must always be greater than or equal to the value in the VAL4 register. Provided complete register descriptions for CAPTCTRLX,CAPTCOMPX,CVAL0, CVAL0C, CVAL1, and CVAL1C registers.
29 FlexRay Communication Controller (FLEXRAY)	In Section 29.4, Protocol Engine Clocking, changed “The first source is the internal crystal oscillator” to “The first source is the external crystal oscillator.” In Section 29.4.1, Oscillator Clocking, changed “If the protocol engine is clocked by the internal crystal oscillator to “If the protocol engine is clocked by the external crystal oscillator.” Removed Section 29.4.2, PLL Clocking. In Table 29-8, changed FR_MCR[CLKSEL] bit description “0 PE clock source is generated by on-chip crystal oscillator.” to “0 PE clock source is generated by external crystal oscillator.” In Section 29.6.6.2.3.1, Application Transitions, and Section 29.6.6.3.1.1, Application Transitions, added the following after the second paragraph: “If the communication controller is started as a non-cold start node and configured and enabled message buffers in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing '1' to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. This should enable the message buffer disable during the LISTEN states.”
30 Frequency-Modulated Phase-Locked Loop (FMPLL)	Added “only on FMPLL_0 for the system clock” in the “Features” section. Replaced “stem clock” with “the source for an active clock on the chip” in the “Recommendations” section. Added “for the system clock” to “Use progressive clock switching” in the “Recommendations” section. Added bullet in Section 30.7, Recommendations, starting with “XOSC needs to be connected and oscillating before the FMPLL can be turned on.”
32 Interrupt Controller (INTC)	Corrected INT offset addresses between 282 and 299.
33 JTAG Controller (JTAGC)	Added footnote to “JTAG Signal Properties” table regarding weak-pull on TDO. Updated “Device Center Code” code in Device Identification Register. Added “Enabling Debug of a Censored Device” section.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
34 LIN Controller (LINFlexD)	Added note to LINIER[DRIE] bit, “In UART FIFO mode, setting the DRIE bit causes the UARTSR[DRFRFE] bit to be immediately set, indicating that the Rx FIFO is empty. This causes the DMA transfer to fail. This bit should not be set in UART FIFO mode.” Changed the layout of the LINIBRR register figure to a single 32-bit format. In the register figures, updated instances of “These fields are writable only in Initialization mode.” to “These fields are writable only in Initialization mode (LINCR1[INIT] = 1).” In the LINESR figure, changed the footnote “If LINTCSR[LTOM] is set, these fields are read-only.” to read “If LINTCSR[LTOM] = 1, these fields are read-only.“ In the LINTOCR figure, added the footnote “The HTO field can only be written in slave mode, LINCR1[MME] = 1” In Section 34.10.24, DMA Tx enable register (DMATXE), added the following note: “For the UART mode and the LIN Master mode, only the DMATXE[DTE0] bit is writable. In these modes, the DMATXE[DTEn] bits (where n Š 1) are read-only bits set at 0, and the other 15 channels are inactive. In the LIN Slave modes, the DMA Tx channels 1 through 15 can be used (when enabled in the identifier filter registers), with 1 to n DMA channels where n = the maximum number of identifier filters enabled. For example, if 3 ID filters are used, then DMATXE[DTE2:0] bits are writable.” In Section 34.10.25, DMA Rx enable register (DMARXE), added the following note: “For the UART mode and the LIN Master mode, only the DMARXE[DRE0] bit is writable. In these modes, the DMARXE[DREn] bits (where n Š 1) are read-only bits set at 0, and the other 15 channels are inactive. In the LIN Slave modes, the DMA Rx channels 1 through 15 can be used (when enabled in the identifier filter registers), with 1 to n DMA channels where n = the maximum number of identifier filters enabled. For example, if 3 ID filters are used, then DMARXE[DRE2:0] bits are writable.”
35 Mode Entry Module (MC_ME)	In Table 35-2, changed CRCU1 to CRC1. In Table 35-31, changed ME_PS3[CRCU1] to ME_PS3[CRC1]. In Section 35.3.2.18, Peripheral Status Register 3 (ME_PS3), changed ME_PS3[CRCU1] to ME_PS3[CRC1].
36 Multi-Port DDR DRAM Controller (MDDRC)	Added note that DDR2 can be used only if its minimum frequency is 90 MHz.
37 Multi-Port DRAM Controller Priority Manager (PRIOMAN)	Corrected title of Section 37.3.2.1, PRIOMAN_CONFIG1. In Section 37.4.1, Description of Operation — Overview, added paragraph beginning with “Setting ACK_SEL to 0 is appropriate for peripherals that desire high priority.”
38 Memory Protection Unit (MPU)	Updated SPERR bit description in Table 38-4. In Section 38.6.4.3, MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2), • Added Note: below first paragraph: “The logical master number depends on whether the MPC5675K is operating in LSM or DPM, as shown in Table 9-1.” • Added sentence and Note: below bulleted list: “For bus masters 4–7, only read/write access can be controlled.” • Added Note: below bulleted list: “Bus master 7 is not implemented on the MPC5675K. See Table 9-1.” • Added bits M6RE, M6WE, M5RE, M5WE, M4RE, and M4WE to Figure 38-7. • Added descriptions for bits M6RE, M6WE, M5RE, M5WE, M4RE, and M4WE to Table 38-9. • Added bits M6RE, M6WE, M5RE, M5WE, M4RE, and M4WE to Figure 38-9. Renamed Section 38.7.2, Putting it all together and XBAR error terminations, to Section 38.7.2, Functional integration details.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
39 Nexus Development Interface (NDI)	Changed clock for MCKO_DIV from SYS_CLK to CORE_CLK. Removed MCKO_DIV[2:0] row from Table 39-10. In Table 39-24, added note to the NSID bitfield description: “Note: The tool is free to set this bitfield to any value.” Added “Parameter values” section with “Device parameter values” table (Table 39-1). Added text to the “Overview” section. Added censorship information to “Features” and “Reset” sections. Added Section 39.2.3.5, Censored Mode. Updated the wording in the “Nexus Double Data Rate Mode” section. Extensive updates to Section 39.4, Nexus Master IDs. Modified Section 39.5.1.4, Port Configuration Register (PCR). Removed “Port Replacement Data Register” and “Port Replacement Data Direction Register” sections. Corrected Figure 39-14. Changed NASPS_0/1 to NXSS_0/1 in Figure 39-15. Modified the “MCKO and ipg_sync_mcko” section and renamed to “MCKO”. Added column label to tables in “Core Nexus Standalone” and “AHB Nexus Standalone” sections. Changed “launch” to “capture” in “Core Nexus Standalone” section. Changed “SRAM sniffer” to “Nexus XBAR Slave SRAM Trace monitor” in the “AHB Nexus Standalone” section title. Made improvements to the text in this section. Made updates to the “Nexus Reset Control” section. Removed the “System Clock Locked Indication.”
41 Parallel Digital Interface (PDI)	Corrected bit name PDIADCCR[ADCVALIDELAY] to PDIADCCR[ADCVALIDDELAY]. In Table 41-13, updated PDIT1R[MC] bit description. In Section 41.1.3.3, Test Mode, updated values of data streams.
44 Power Management Controller (PMC)	Changed pin names in the “Block Diagram” section to ball names. Replaced the entire “External Signals Description” section as follows: Added external pin descriptions to “External Signals Description” section. Added “External versus Internal Signals Names” table in conjunction with the above signal name changes. Removed the “PMC operating modes” section as it does not apply. Updated the “Register Map” section.
45 Register Protection (REG_PROT)	In Section 45.6, MPC5675K registers under protection, added this paragraph: “For registers not under protection from a non-supervisor write access (e.g., EBI[MCR] and I2Cn[IBAD]) (of all the I2C), the user can verify register contents with a periodic CRC to ensure that the registers values have not been altered during code execution”

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
46 Reset Generation Module (MC_RGM)	In Figure 46-1, replaced PA[4:2] with etimer0_ETC[3], etimer1_ETC[4:3]. In Section 46.1.2, Reset Sources, • Added phrase “, and the memory content must be considered to be unknown.” to the first bullet item. • Replaced next to last sentence in second bullet with “In this case, most digital modules are reset normally, and the memory content must be considered to be unknown, while the state of analog modules or specific digital modules (e.g., debug modules, flash modules) as well as the system memory content is preserved. “ • Removed “FCCU SAFE mode request” from list of Functional requests. • Removed final sentence of section, “Destructive reset source events cannot be converted to SAFE mode requests or to an interrupt to the core. “ In Section 46.3, Memory Map and Register Description, removed “Transfer errors are not caused by write accesses to these registers: • RGM_DERD • 0xC3FE_4012 • 0xC3FE_401A” from Note. In Section 46.2,External signal description, changed “boot mode pins PA[4:2]” to boot mode pins “etimer0_ETC[3], etimer1_ETC[4:3].” Changed RGM_DERD[D_ADCLVD27_IO] to RGM_DERD[D_LVD27_ADC]. In Table 46-2 and Table 46-7, added RGM_FEAR[AR_FL_ECC_RCC] bit. In Section 46.3.1, Register Descriptions, added Note: “Some fields may be read-only, and their reset value of ‘1’ or ‘0’ and the corresponding behavior cannot be changed.” In Section 46.3.1.1, Functional Event Status Register (RGM_FES), • Changed paragraph “This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write ‘1’.” to “This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits are cleared on write ‘1’ if the triggering event has already been cleared at the source.” • Added Note: “If a ‘functional’ reset source is configured to generate a SAFE mode request or an interrupt request, software needs to clear the event in the source module at least three system clock cycles before it clears the associated RGM_FES status bit in order to avoid multiple SAFE mode requests or interrupts for the same event. In order to avoid having to count cycles, it is good practice for software to check whether the RGM_FES has been properly cleared, and if not, clear it again.” In Section 46.3.1.2, Destructive Event Status Register (RGM_DES), added sentence: “It can be accessed in read only in user mode.” In Section 46.3.1.3, Functional Event Reset Disable Register (RGM_FERD), added Warning: “It is important to clear the RGM_FES register before setting any of the bits in the RGM_FERD register to ‘1’. Otherwise a redundant SAFE mode request or interrupt request may occur.” In Table 46-5, changed value 1 for bit D_SWT from “A flash, ECC, or lock-step error event generates a SAFE mode request” to “Reserved.” In Table 46-8, changed description of bit RGM_FESS[SS_SOFT_FUNC] value 1 from “The reset sequence triggered by a software (functional) reset event starts from PHASE3, skipping PHASE1 and PHASE2” to “Reserved.” Updated all of Section 46.4.6, Boot Mode Capturing.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
47 Semaphore Unit (SEMA4)	In Section 47.1, Introduction, replaced this sentence: “In LSM, an access to the SEMA4 module results in an ips_xfr_err, which filters through the PBRIDGE and becomes an hresp=error, which the core takes as a machine-check exception.” with “In LSM, an access to the SEMA4 module results in an error that the core takes as a machine-check exception.” In Section 47.6, Application information, • Removed the sentence “Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., rdcpnrx for a read CPU number instruction.” • Changed “PowerPC” to “Power Architecture.”
48 System Integration Unit Lite (SIUL)	Removed all PISR register content. Added Figure 48-41, PADSEL Muxing. Updated Table 48-11 to show reset values and pin mapping. Updated SRC bit description in Table 48-10 as follows: “Slew Rate Control 000Slowest configuration. . . . 111Fastest configuration. Note: For any given PCRn register, not all the SRC bits may be implemented. See Table 48-11 for more information. SRC bits shown as “x” are not implemented and should not be written.”
49 System Status and Configuration Module (SSCM)	Updated reset value for SSCM_UOPS register to “Varies depending on the contents of the shadow block.” Removed Section 49.5, Initialization/Application Information. Removed SSCM_SCTR register. Added Note in LSM field of SSCM_STATUS register, Table 49-3. Removed ABD field from Table 49-3 and Figure 49-2. Added note to SSCM_STATUS[LSM] bit description: “This field is used only to see what mode (LSM or DPM) the chip is in. To configure the chip to run in one of these modes, see Section 5.6, Selecting LSM or DPM.” In Section 49.3.1.8, User Option Status (SSCM_UOPS) Register, replaced Table 49-15, added bits to Figure 49-10. Removed section 49.3.1.12 PFLASHC BIU Reset Values.
50 Self-Test Control Unit (STCU)	Added Table 50-15, MBIST Fault Flags.
51 System Timer Module (STM)	In Section 51.3.2.3, STM Channel Control Register n (STM_CCRn), updated register abbreviations in register figure.
52 Software Watchdog Timer (SWT)	Rewrote description for SWT_TO[WTO] bit.

Table A-8. Changes between revisions 2 and 3 (continued)

Chapter	Description
54 Wakeup Unit (WKPU)	Changed “PLATFORM” block to “CPUs” in the “Wakeup Unit Block Diagram” figure. Changed “Wakeup Unit has one signal input that” to “NMI input pin” in the “External Signal Description” section. Changed “system wakeup sources in certain power down modes” to “chip wakeup source during STOP0 mode” in the “External Signal Description” section. Changed “Wake-up” to “Wakeup” throughout chapter for consistency. Removed the text regarding unbonded pins in the “External Signal Description” section, as this does not apply to MPC5675K. Changed “NMI input 0” to “the NMI pin” in the “NMI Status Flag Register” section. Change “per NMI input to the SoC” to “for the NMI input pin” in the “Non-Maskable Interrupts” section. Removed “per NMI input” in the “Non-Maskable Interrupts” section. Removed “Each” in the “Non-Maskable Interrupts” section. Replaced “Each” with “The” in the “NMI Management” section. Routed NDSS line around the “Wakeup Enable” block in the “NMI Pad Diagram” figure. Labeled the NMI pad in the “NMI Pad Diagram” figure.
A Memory Map	Added this appendix.
B Revision History	Added this appendix.

A.9 Changes between revisions 1 and 2

Table A-9. Changes between revisions 1 and 2

Chapter	Description
Throughout	Editorial changes and improvements.
All	Extensive revisions to entire chapter content.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MPC5675KRM
Rev. 10
11/2013