RM0046
Reference manual

SPC560P34/SPC560P40 32-bit MCU family built on the embedded Power Architecture®

Introduction

The SPC560P40/34 microcontroller is built on the Power Architecture® platform. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers. This device family integrates the most advanced and up-to-date motor control design features.

The safety features included in SPC560P40/34 (such us fault collection unit, safety port or flash memory and SRAM with ECC) support the design of system applications where safety is a requirement.

Contents

List of tables

List of figures

Preface

Overview

The primary objective of this document is to define the functionality of the SPC560P40/34 family of microcontrollers for use by software and hardware developers. The SPC560P40/34 family is built on Power Architecture® technology and integrates technologies that are important for today's electrical hydraulic power steering (EHPS), electric power steering (EPS), airbag applications, anti-lock braking systems (ABS), and motor control applications.

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 ST Web site at www.st.com.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC560P40/34 device. 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 device as a whole
- The functionality of the individual modules on the device

In the latter, any device-specific information is presented in the section "Information Specific to This Device" at the beginning of the chapter.

References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC560P40/34:

- IEEE-ISTO 5001™ - 2003 and 2010, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

1 Introduction

1.1 The SPC560P40/34 microcontroller family

The SPC560P40/34 microcontroller is built on the Power Architecture® platform. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers. This device family integrates the most advanced and up-to-date motor control design features.

The safety features included in SPC560P40/34 (such us fault collection unit, safety port or flash memory and SRAM with ECC) support the design of system applications where safety is a requirement.

The SPC560P40/34 addresses low-end chassis applications and implements the Harvard bus interface version of the e200z0h core.

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture Book E architecture. The e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance. The e200z0h processor integrates an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable to sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by branch unit to allow single-cycle branches in some cases. The e200z0h core is a single-issue, 32-bit Power Architecture technology VLE only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). Instead of the base Power Architecture instruction set support, the e200z0h core only implements the VLE (variable length encoding) APU, providing improved code density.

The SPC560P40/34 has a single level of memory hierarchy consisting of 20 KB on-chip SRAM and 320 KB (256 KB program + 64 KB data) of on-chip flash memory. Both the SRAM and the flash memory can hold instructions and data.

The timer functions of the SPC560P40/34 are performed by the eTimer Modular Timer System and FlexPWM. The eTimer module implements enhanced timer features (six channels) including dedicated motor control quadrature decode functionality and DMA support; the FlexPWM module consists of four submodules controlling a pair of PWM channels each: three submodules may be used to control the three phases of a motor and the additional pair to support DC-DC converter width modulation control.

Off-chip communication is performed by a suite of serial protocols including CANs, enhanced SPIs (DSPI), and SCIs (LINFlex).

The System Integration Unit Lite (SIUL) performs several chip-wide configuration functions. Pad configuration and general-purpose input/output (GPIO) are controlled from the SIUL. External interrupts and reset control are also found in the SIUL. The internal multiplexer sub-block (IOMUX) provides multiplexing of daisy chaining the DSPIs and external interrupt signal.

As the SPC560P40/34 is built on a wider legacy of Power Architecture-based devices, when applicable and possible, reuse or enhancement of existing IP, design and concepts is adopted.

1.2 Target applications

The SPC560P40/34 belongs to an expanding range of automotive-focused products designed to address and target the following chassis and safety market segments:

- Electric hydraulic power steering (EHPS)
- Lower end of electric power steering (EPS)
- Airbag applications
- Anti-lock braking systems (ABS)
- Motor control applications

EHPS and EPS systems typically feature sophisticated and advanced electrical motor control periphery with special enhancements in the area of pulse width modulation, highly flexible timers, and functional safety.

1.2.1 Application examples

Electric power steering

Figure 1 outlines a typical electric power steering application built around the SPC560P40/34 microcontroller.

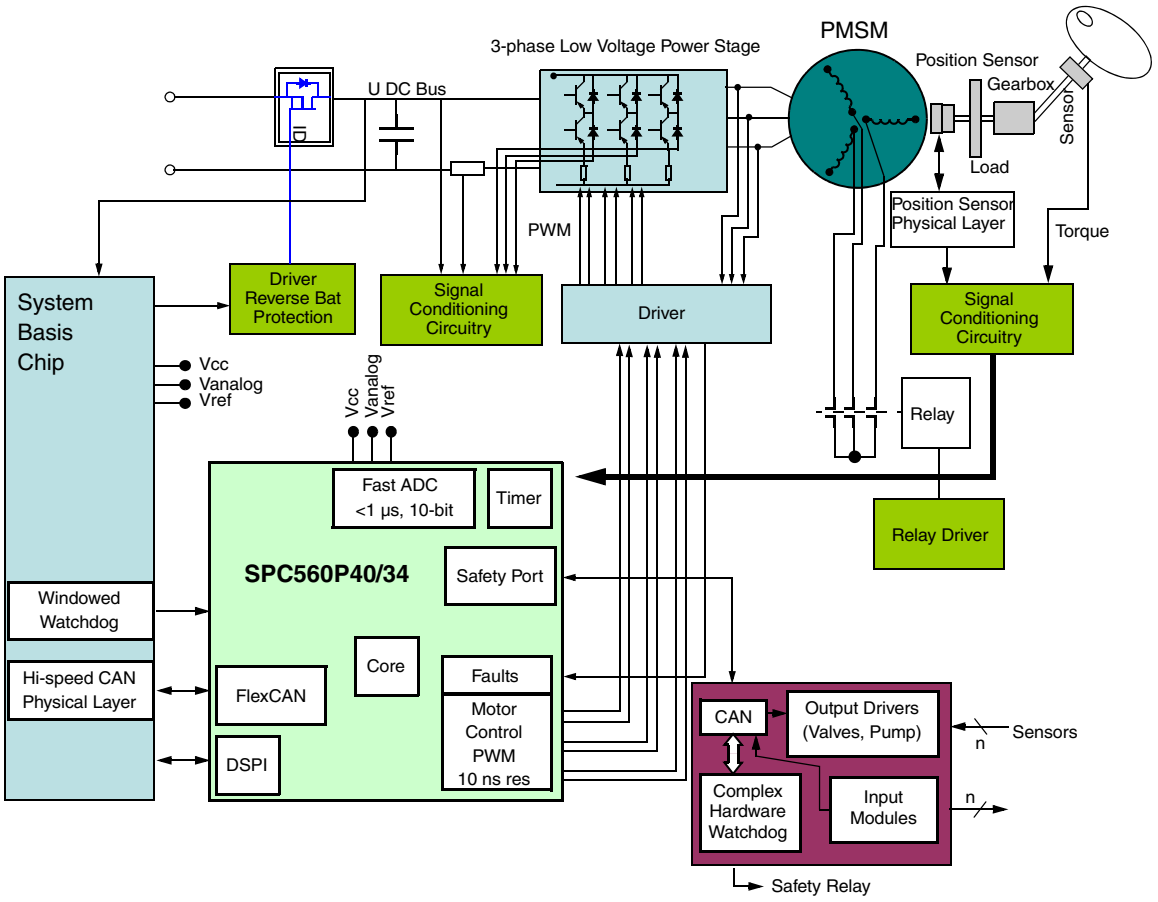


Figure 1. Electric power steering application

Airbag

Figure 2 outlines a typical airbag application built around the SPC560P40/34 microcontroller.

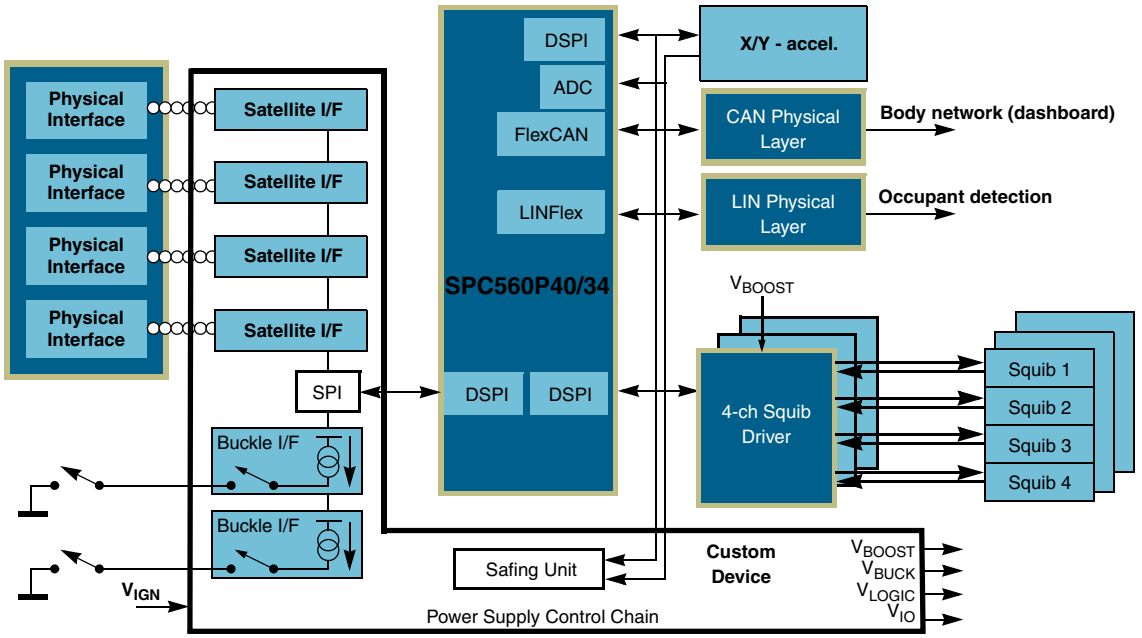


Figure 2. Airbag application

1.3 Features

Table 1 provides a summary of different members of the SPC560P40/34 family and their features—relative to full-featured version—to enable a comparison among the family members and an understanding of the range of functionality offered within this family.

Table 1. SPC560P40/34 device comparison

Feature	SPC560P34 Full-featured	SPC560P40 Full-featured
Code flash memory (with ECC)	192 KB	256 KB
Data flash memory / EE option (with ECC)	64 KB	
SRAM (with ECC)	12 KB	20 KB
Processor core	32-bit e200z0h	
Instruction set	VLE (variable length encoding)	
CPU performance	0–64 MHz	
FMPLL (frequency-modulated phase-locked loop) module	1	

Table 1. SPC560P40/34 device comparison (continued)

Feature		SPC560P34 Full-featured	SPC560P40 Full-featured
INTC (interrupt controller) channels		120	
PIT (periodic interrupt timer)		1 (with four 32-bit timers)	
eDMA (enhanced direct memory access) channels		16	
FlexCAN (controller area network)		1(1)	2 1,(2)
Safety port		No	Yes (via second FlexCAN module)
FCU (fault collection unit)		Yes	
CTU (cross triggering unit)		Yes	Yes
eTimer		1 (16-bit, 6 channels)	
FlexPWM (pulse-width modulation) channels		8 (capture capabity not supported)	8 (capture capability not supported)
Analog-to-digital converter (ADC)		1 (10-bit, 16 channels)	
LINFlex		2 (1 × Master/Slave, 1 × Master only)	2 (1 × Master/Slave, 1 × Master only)
DSPI (deserial serial peripheral interface)		2	3
CRC (cyclic redundancy check) unit		Yes	
Junction temperature sensor		No	
JTAG controller		Yes	
Nexus port controller (NPC)		Yes (Nexus Class 1)	
Supply	Digital power supply(3)	3.3 V or 5 V single supply with external transistor	
	Analog power supply	3.3 V or 5 V	
	Internal RC oscillator	16 MHz	
	External crystal oscillator	4–40 MHz	
Packages		LQFP64 LQFP100	
Temperature	Standard ambient temperature	–40 to 125 °C	

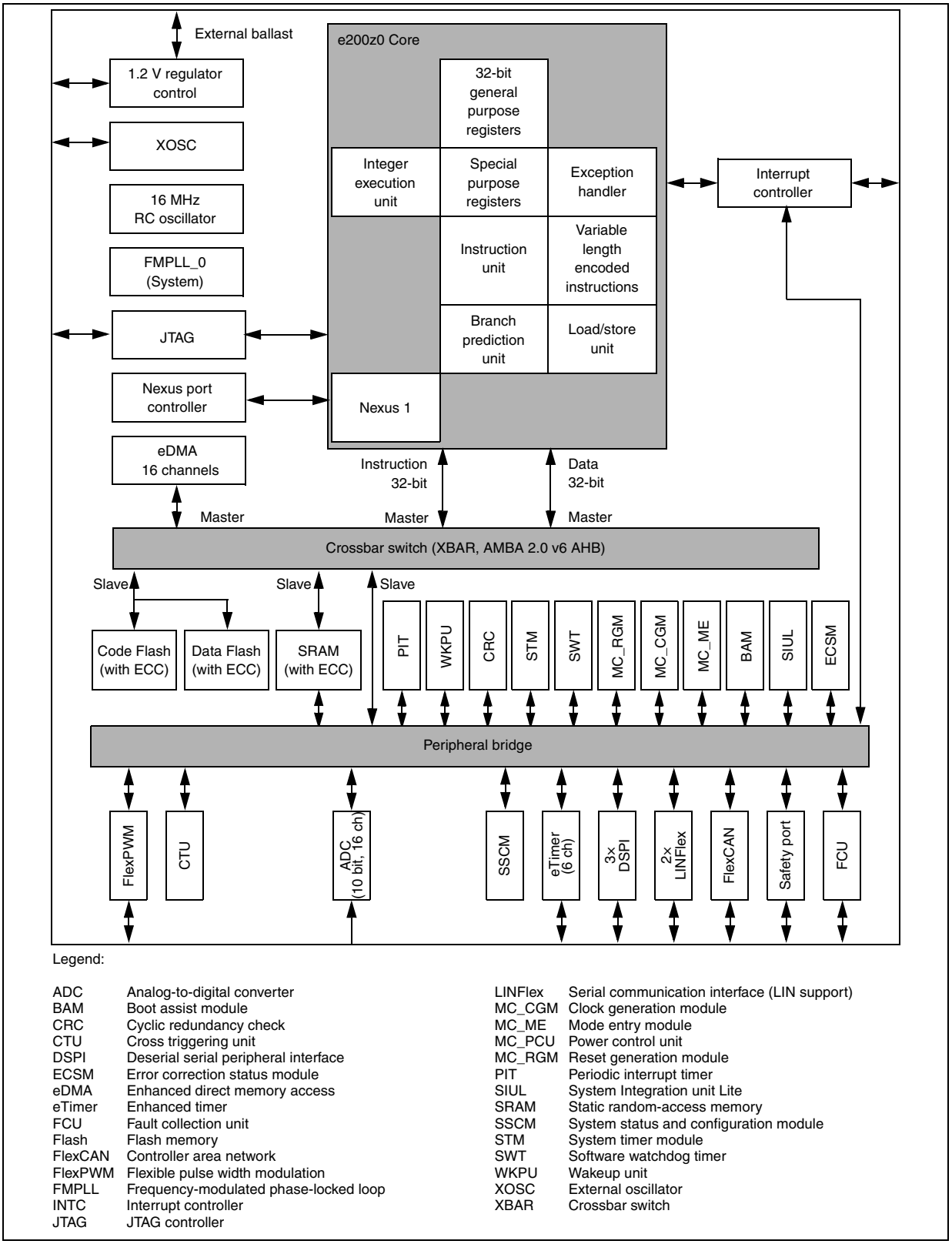
1. Each FlexCAN module has 32 message buffers.
2. One FlexCAN module can act as a safety port with a bit rate as high as 8 Mbit/s at 64 MHz.
3. The different supply voltages vary according to the part number ordered.

SPC560P40/34 is available in two configurations having different features: Full-featured and airbag. *Table 2* shows the main differences between the two versions of the SPC560P40 MCU.

Table 2. SPC560P40 device configuration differences

Feature	Configuration	
	Airbag	Full-featured
SRAM (with ECC)	16 KB	20 KB
FlexCAN (controller area network)	1	2
Safety port	No	Yes (via second FlexCAN module)
FlexPWM (pulse-width modulation) channels	No	8 (capture capability not supported)
CTU (cross triggering unit)	No	Yes

Figure 1.4 shows a top-level block diagram of the SPC560P40/34 microcontroller.



Legend:

ADC	Analog-to-digital converter
BAM	Boot assist module
CRC	Cyclic redundancy check
CTU	Cross triggering unit
DSPI	Deserial serial peripheral interface
ECSM	Error correction status module
eDMA	Enhanced direct memory access
eTimer	Enhanced timer
FCU	Fault collection unit
Flash	Flash memory
FlexCAN	Controller area network
FlexPWM	Flexible pulse width modulation
FMPLL	Frequency-modulated phase-locked loop
INTC	Interrupt controller
JTAG	JTAG controller
LINFlex	Serial communication interface (LIN support)
MC_CGM	Clock generation module
MC_ME	Mode entry module
MC_PCU	Power control unit
MC_RGM	Reset generation module
PIT	Periodic interrupt timer
SIUL	System Integration unit Lite
SRAM	Static random-access memory
SSCM	System status and configuration module
STM	System timer module
SWT	Software watchdog timer
WKPU	Wakeup unit
XOSC	External oscillator
XBAR	Crossbar switch

Figure 3. Block diagram (SPC560P40 full-featured configuration)

1.4 Critical performance parameters

- Fully static operation, 0–64 MHz
- –40 °C to 150 °C junction temperature
- Low power design
  - Less than 450 mW power dissipation
  - Halt and STOP mode available for power reduction
  - Resuming from Halt/STOP mode can be initiated via external pin
- Fabricated in 90 nm process
- 1.2 V nominal internal logic
- Nexus pins operate at V_{DDIO} (no dedicated power supply)
  - Unused pins configurable as GPIO
- 10-bit ADC conversion time < 1 µs
- Internal voltage regulator (VREG) with external ballast transistor enables control with a single input rail
  - 3.0 V–3.6 V or 4.5 V–5.5 V input supply voltage
- Configurable pins
  - Selectable slew rate for EMI reduction
  - Selectable pull-up, pull-down, or no pull on all pins
  - Selectable open drain
  - Support for 3.3 V or 5 V I/O levels

1.5 Chip-level features

On-chip modules available within the family include the following features:

- Single issue, 32-bit CPU core complex (e200z0h)
  - Compliant with Power Architecture™ embedded category
  - Variable Length Encoding (VLE)
- Memory
  - Up to 256 KB on-chip Code Flash with ECC and erase/program controller
  - Up to additional 64 (4 × 16) KB on-chip Data Flash with ECC for EEPROM emulation
  - Up to 20 KB on-chip SRAM with ECC
- Fail-safe protection
  - Programmable watchdog timer
  - Non-maskable interrupt
  - Fault collection unit
- Nexus L1 interface

- Interrupts and events
  - 16-channel eDMA controller
  - 16 priority level controller
  - Up to 25 external interrupts
  - PIT implements four 32-bit timers
  - 120 interrupts are routed via INTC
- General purpose I/Os
  - Individually programmable as input, output or special function
  - 37 on LQFP64
  - 64 on LQFP100
- 1 general purpose eTimer unit
  - 6 timers each with up/down capabilities
  - 16-bit resolution, cascadeable counters
  - Quadrature decode with rotation direction flag
  - Double buffer input capture and output compare
- Communications interfaces
  - 2 LINFlex channels (1 × Master/Slave, 1 × Master Only)
  - Up to 3 DSPI channels with automatic chip select generation (up to 8/4/4 chip selects)
  - 1 FlexCAN interface (2.0B Active) with 32 message buffers
  - 1 safety port based on FlexCAN with 32 message buffers and up to 8 Mbit/s at 64 MHz capability usable as second CAN when not used as safety port
- One 10-bit analog-to-digital converter (ADC)
  - Up to 16 input channels (16 ch on LQFP100 and 12 ch on LQFP64)
  - Conversion time < 1 µs including sampling time at full precision
  - Programmable Cross Triggering Unit (CTU)
  - 4 analog watchdogs with interrupt capability
- On-chip CAN/UART bootstrap loader with Boot Assist Module (BAM)
- 1 FlexPWM unit
  - 8 complementary or independent outputs with ADC synchronization signals

1.6 Module features

1.6.1 High performance e200z0 core processor

The e200z0 Power Architecture core provides the following features:

- High performance e200z0 core processor for managing peripherals and interrupts
- Single issue 4-stage pipeline in-order execution 32-bit Power Architecture CPU
- Harvard architecture
- Variable length encoding (VLE), allowing mixed 16- and 32-bit instructions
  - Results in smaller code size footprint
  - Minimizes impact on performance

- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- Thirty-two 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Long cycle time instructions, except for guarded loads, do not increase interrupt latency
- Extensive system development support through Nexus debug port
- Non-maskable interrupt support

1.6.2 Crossbar switch (XBAR)

The XBAR multi-port crossbar switch supports simultaneous connections between three master ports and three slave ports. The crossbar supports a 32-bit address bus width and a 32-bit data bus width.

The crossbar allows for two concurrent transactions to occur from any master port to any slave port; but one of those transfers must be an instruction fetch from internal flash memory. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will be stalled until the higher priority master completes its transactions. Requesting masters will be treated with equal priority and will be granted access a slave port in round-robin fashion, based upon the ID of the last master to be granted access.

The crossbar provides the following features:

- 3 master ports:
  - e200z0 core complex instruction port
  - e200z0 core complex Load/Store Data port
  - eDMA
- 3 slave ports:
  - Flash memory (Code and Data)
  - SRAM
  - Peripheral bridge
- 32-bit internal address, 32-bit internal data paths
- Fixed Priority Arbitration based on Port Master
- Temporary dynamic priority elevation of masters

1.6.3 Enhanced direct memory access (eDMA)

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 16 programmable channels, with minimal intervention from the host processor. The hardware micro architecture includes a DMA engine which performs source and destination address calculations, and the actual

data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels.

The eDMA module provides the following features:

- 16 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 either post-increment or to 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
- DMA transfers possible between system memories, DSPIs, ADC, FlexPWM, eTimer and CTU
- Programmable DMA channel multiplexer allows assignment of any DMA source to any available DMA channel with as many as 30 request sources
- eDMA abort operation through software

1.6.4 Flash memory

The SPC560P40/34 provides 320 KB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used for instruction and/or data storage. The flash memory module is interfaced to the system bus by a dedicated flash memory controller. It supports a 32-bit data bus width at the system bus port, and a 128-bit read data interface to flash memory. The module contains four 128-bit wide prefetch buffers. Prefetch buffer hits allow no-wait responses. Normal flash memory array accesses are registered and are forwarded to the system bus on the following cycle, incurring two wait-states.

The flash memory module provides the following features:

- As much as 320 KB flash memory
  - 6 blocks (32 KB + 2×16 KB + 32 KB + 32 KB + 128 KB) code flash memory
  - 4 blocks (16 KB + 16 KB + 16 KB + 16 KB) data flash memory
  - Full Read-While-Write (RWW) capability between code flash memory and data flash memory
- Four 128-bit wide prefetch buffers to provide single cycle in-line accesses (prefetch buffers can be configured to prefetch code or data or both)
- Typical flash memory access time: no wait-state for buffer hits, 2 wait-states for page buffer miss at 64 MHz
- Hardware managed flash memory writes handled by 32-bit RISC Krypton engine
- Hardware and software configurable read and write access protections on a per-master basis
- Configurable access timing allowing use in a wide range of system frequencies
- Multiple-mapping support and mapping-based block access timing (up to 31 additional cycles) allowing use for emulation of other memory types
- Software programmable block program/erase restriction control
- Erase of selected block(s)

- Read page sizes
  - Code flash memory: 128 bits (4 words)
  - Data flash memory: 32 bits (1 word)
- ECC with single-bit correction, double-bit detection for data integrity
  - Code flash memory: 64-bit ECC
  - Data flash memory: 32-bit ECC
- Embedded hardware program and erase algorithm
- Erase suspend and program abort
- Censorship protection scheme to prevent flash memory content visibility
- Hardware support for EEPROM emulation

1.6.5 Static random access memory (SRAM)

The SPC560P40/34 SRAM module provides up to 20 KB of general-purpose memory.

The SRAM module provides the following features:

- Supports read/write accesses mapped to the SRAM from any master
- Up to 20 KB general purpose SRAM
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- Typical SRAM access time: no wait-state for reads and 32-bit writes; 1 wait-state for 8- and 16-bit writes if back-to-back with a read to same memory block

1.6.6 Interrupt controller (INTC)

The interrupt controller (INTC) provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems. The INTC handles 128 selectable-priority interrupt sources.

For high-priority interrupt requests, the time from the assertion of the interrupt request by the peripheral to the execution of the interrupt service routine (ISR) by the processor has been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR has to be executed. It also provides a wide number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol (PCP) for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the same resource can not preempt each other.

The INTC provides the following features:

- Unique 9-bit vector for each separate interrupt source
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source

- Ability to modify the ISR or task priority: modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- 1 external high priority interrupt (NMI) directly accessing the main core and I/O processor (IOP) critical interrupt mechanism

1.6.7 System status and configuration module (SSCM)

The system status and configuration module (SSCM) provides central device functionality.

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search code flash for bootable sector
  - DMA status
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

1.6.8 System clocks and clock generation

The following list summarizes the system clock and clock generation on the SPC560P40/34:

- Lock detect circuitry continuously monitors lock status
- Loss of clock (LOC) detection for PLL outputs
- Programmable output clock divider (÷1, ÷2, ÷4, ÷8)
- FlexPWM module and eTimer module running at the same frequency as the e200z0h core
- Internal 16 MHz RC oscillator for rapid start-up and safe mode: supports frequency trimming by user application

1.6.9 Frequency-modulated phase-locked loop (FMPLL)

The FMPLL allows the user to generate high speed system clocks from a 4–40 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor, output clock divider ratio are all software configurable.

The FMPLL has the following major features:

- Input clock frequency: 4–40 MHz
- Maximum output frequency: 64 MHz
- Voltage controlled oscillator (VCO)—frequency 256–512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency-modulated PLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth (±0.25% to ±4% deviation from center frequency): programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation

1.6.10 Main oscillator

The main oscillator provides these features:

- Input frequency range: 4–40 MHz
- Crystal input mode or oscillator input mode
- PLL reference

1.6.11 Internal RC oscillator

This device has an RC ladder phase-shift oscillator. The architecture uses constant current charging of a capacitor. The voltage at the capacitor is compared by the stable bandgap reference voltage.

The RC oscillator provides these features:

- Nominal frequency 16 MHz
- ±5% variation over voltage and temperature after process trim
- Clock output of the RC oscillator serves as system clock source in case loss of lock or loss of clock is detected by the PLL
- RC oscillator is used as the default system clock during startup

1.6.12 Periodic interrupt timer (PIT)

The PIT module implements these features:

- 4 general-purpose interrupt timers
- 32-bit counter resolution
- Clocked by system clock frequency
- Each channel usable as trigger for a DMA request

1.6.13 System timer module (STM)

The STM implements these 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

1.6.14 Software watchdog timer (SWT)

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- Reset configuration inputs allow timer to be enabled out of reset

1.6.15 Fault collection unit (FCU)

The FCU provides an independent fault reporting mechanism even if the CPU is malfunctioning.

The FCU module has the following features:

- FCU status register reporting the device status
- Continuous monitoring of critical fault signals
- User selection of critical signals from different fault sources inside the device
- Critical fault events trigger 2 external pins (user selected signal protocol) that can be used externally to reset the device and/or other circuitry (for example, a safety relay)
- Faults are latched into a register

1.6.16 System integration unit – Lite (SIUL)

The SPC560P40/34 SIUL controls MCU pad configuration, external interrupt, general purpose I/O (GPIO), and internal peripheral multiplexing.

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 general purpose input output (GPIO) control of up to 49 input/output pins and 16 analog input-only pads (package dependent)
- All GPIO pins can be independently configured to support pull-up, pull-down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins, except ADC channels, can be alternatively configured as both general purpose input or output pins
- ADC channels support alternative configuration as general purpose inputs
- Direct readback of the pin value is supported on all pins through the SIUL
- Configurable digital input filter that can be applied to some general purpose input pins for noise elimination
- Up to 4 internal functions can be multiplexed onto 1 pin

1.6.17 Boot and censorship

Different booting modes are available in the SPC560P40/34: booting from internal flash memory and booting via a serial link.

The default booting scheme uses the internal flash memory (an internal pull-down resistor is used to select this mode). Optionally, the user can boot via FlexCAN or LINFlex (using the boot assist module software).

A censorship scheme is provided to protect the content of the flash memory and offer increased security for the entire device.

A password mechanism is designed to grant the legitimate user access to the non-volatile memory.

Boot assist module (BAM)

The BAM is a block of read-only memory that is programmed once and is identical for all SPC560Pxx devices that are based on the e200z0h core. The BAM program is executed

every time the device is powered on if the alternate boot mode has been selected by the user.

The BAM provides the following features:

- Serial bootloading via FlexCAN or LINFlex
- Ability to accept a password via the used serial communication channel to grant the legitimate user access to the non-volatile memory

1.6.18 Error correction status module (ECSM)

The ECSM provides a myriad of miscellaneous control functions regarding program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, wakeup control for exiting sleep modes, and information on platform memory errors reported by error-correcting codes and/or generic access error information for certain processor cores.

The Error Correction Status Module supports a number of miscellaneous control functions for the platform. The ECSM includes these features:

- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- For test purposes, optional registers to specify the generation of double-bit memory errors are enabled on the SPC560P40/34.

The sources of the ECC errors are:

- Flash memory
- SRAM

1.6.19 Peripheral bridge (PBRIDGE)

The PBRIDGE implements the following features:

- Duplicated periphery
- Master access privilege level per peripheral (per master: read access enable; write access enable)
- Write buffering for peripherals
- Checker applied on PBRIDGE output toward periphery
- Byte endianess swap capability

1.6.20 Controller area network (FlexCAN)

The SPC560P40/34 MCU contains one controller area network (FlexCAN) module. This module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily 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 contains 32 message buffers.

The FlexCAN module provides the following features:

- Full implementation of the CAN protocol specification, version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Up to 8-bytes data length
  - Programmable bit rate up to 1 Mbit/s
- 32 message buffers of up to 8-bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Programmable loop-back mode supporting self-test operation
- 3 programmable mask registers
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- 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)
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Transmit features
  - Supports configuration of multiple mailboxes to form message queues of scalable depth
  - Arbitration scheme according to message ID or message buffer number
  - Internal arbitration to guarantee no inner or outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - 8 mailboxes configurable as a 6-entry receive FIFO
  - 8 programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter

1.6.21 Safety port (FlexCAN)

The SPC560P40/34 MCU has a second CAN controller synthesized to run at high bit rates to be used as a safety port. The CAN module of the safety port provides the following features:

- Identical to the FlexCAN module
- Bit rate up to 8 Mbit/s at 64 MHz CPU clock using direct connection between CAN modules (no physical transceiver required)
- 32 message buffers of up to 8-bytes data length
- Can be used as a second independent CAN module

1.6.22 Serial communication interface module (LINFlex)

The LINFlex (local interconnect network flexible) on the SPC560P40/34 features the following:

- Supports LIN Master mode (both instances), LIN Slave mode (only one instance) and UART mode
- LIN state machine compliant to LIN1.3, 2.0 and 2.1 specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store Identifier and up to 8 data bytes
  - Supports message length of up to 64 bytes
  - Detection and flagging of LIN errors (sync field, delimiter, ID parity, bit framing, checksum, and time-out)
  - 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; Self Test; LIN bus stuck dominant detection
  - Interrupt-driven operation with 16 interrupt sources
- LIN slave mode features:
  - Autonomous LIN header handling
  - Autonomous LIN response handling
  - Optional discarding of irrelevant LIN responses using ID filter
- 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 or 9-bit words)
  - 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
  - 2 receiver wake-up methods

1.6.23 Deserial serial peripheral interface (DSPI)

The deserial serial peripheral interface (DSPI) module provides a synchronous serial interface for communication between the SPC560P40/34 MCU and external devices.

The DSPI modules provide these features:

- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase

- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- Up to 8 chip select lines available:
  - 8 on DSPI_0
  - 4 each on DSPI_1 and DSPI_2
- 8 clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for deglitching
- FIFOs for buffering up to 4 transfers on the transmit and receive side
- Queueing operation possible through use of the I/O processor or eDMA
- General purpose I/O functionality on pins when not used for SPI

1.6.24 Pulse width modulator (FlexPWM)

The pulse width modulator module (PWM) contains four PWM submodules each of which is set up to control a single half-bridge power stage. There are also three 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.

The FlexPWM block implements the following features:

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- Clock frequency same as that used for e200z0h core
- 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 ADC trigger events can be generated per PWM cycle via hardware
- Write protection for critical registers
- 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
- eDMA support with automatic reload
- 2 fault inputs
- Capture capability for PWMA, PWMB, and PWMX channels not supported

1.6.25 eTimer

The SPC560P40/34 includes one eTimer module which provides six 16-bit general purpose up/down timer/counter units with the following features:

- Clock frequency same as that used for the e200z0h core
- 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
  - External event counting: max. count rate = peripheral clock/2
  - Internal clock counting: max. count rate = peripheral clock
- Counters are:
  - Cascadable
  - Preloadable
- Programmable count modulo
- Quadrature decode capabilities
- Counters can share available input pins
- Count once or repeatedly
- Pins available as GPIO when timer functionality not in use

1.6.26 Analog-to-digital converter (ADC) module

The ADC module provides the following features:

Analog part:

- 1 on-chip analog-to-digital converter
  - 10-bit AD resolution
  - 1 sample and hold unit
  - Conversion time, including sampling time, less than 1 µs (at full precision)
  - Typical sampling time is 150 ns minimum (at full precision)
  - DNL/INL ±1 LSB
  - TUE < 1.5 LSB
  - Single-ended input signal up to 3.3 V/5.0 V
  - 3.3 V/5.0 V input reference voltage
  - ADC and its reference can be supplied with a voltage independent from V_{DDIO}
  - ADC supply can be equal or higher than V_{DDIO}
  - ADC supply and ADC reference are not independent from each other (both internally bonded to same pad)
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles

Digital part:

- 16 input channels
- 4 analog watchdogs comparing ADC results against predefined levels (low, high, range) before results are stored in the appropriate ADC result location
- 2 modes of operation: Motor Control mode or Regular mode
- Regular mode features
  - Register based interface with the CPU: control register, status register and 1 result register per channel
  - ADC state machine managing 3 request flows: regular command, hardware injected command and software injected command
  - Selectable priority between software and hardware injected commands
  - DMA compatible interface
- CTU-controlled mode features
  - Triggered mode only
  - 4 independent result queues (1×16 entries, 2×8 entries, 1×4 entries)
  - Result alignment circuitry (left justified and right justified)
  - 32-bit read mode allows to have channel ID on one of the 16-bit part
  - DMA compatible interfaces

1.6.27 Cross triggering unit (CTU)

The cross triggering unit allows 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.

It implements the following features:

- Double buffered trigger generation unit with up to 8 independent triggers generated from external triggers
- Trigger generation unit configurable in sequential mode or in triggered mode
- Each trigger can be appropriately delayed 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 up to 24 ADC commands
- Each trigger capable of generating consecutive commands
- ADC conversion command allows to control ADC channel, single or synchronous sampling, independent result queue selection

1.6.28 Nexus Development Interface (NDI)

The NDI (Nexus Development Interface) block is compliant with Nexus Class 1 of the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility. The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for this device. The NDI block interfaces to the host processor and internal busses to provide development support as per the IEEE-ISTO 5001-2003 Nexus Class 1 standard. The development support provided includes access to the MCU's internal memory map and access to the processor's internal registers.

The NDI provides the following features:

- Configured via the IEEE 1149.1
- All Nexus port pins operate at V_{DDIO} (no dedicated power supply)
- Nexus Class 1 supports Static debug

1.6.29 Cyclic redundancy check (CRC)

The CRC computing unit is dedicated to the computation of CRC off-loading the CPU. The CRC module features:

- Support for CRC-16-CCITT (*x*25 protocol):
  - $x^{16} + x^{12} + x^5 + 1$
- Support for CRC-32 (Ethernet protocol):
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Zero wait states for each write/read operations to the CRC_CFG and CRC_INP registers at the maximum frequency

1.6.30 IEEE 1149.1 JTAG controller

The JTAG controller (JTAGC) block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. All data input to and output from the JTAGC block is communicated in serial format. The JTAGC block is compliant with the IEEE standard.

The JTAG controller provides the following features:

- IEEE test access port (TAP) interface 4 pins (TDI, TMS, TCK, TDO)
- Selectable modes of operation include JTAGC/debug or normal system operation.
- 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS
  - IDCODE
  - EXTEST
  - SAMPLE
  - SAMPLE/PRELOAD
- 5-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC
  - ACCESS_AUX_TAP_ONCE
- 3 test data registers:
  - Bypass register
  - Boundary scan register (size parameterized to support a variety of boundary scan chain lengths)
  - Device identification register
- TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry

1.6.31 On-chip voltage regulator (VREG)

The on-chip voltage regulator module provides the following features:

- Uses external NPN (negative-positive-negative) transistor
- Regulates external 3.3 V/5.0 V down to 1.2 V for the core logic
- Low voltage detection on the internal 1.2 V and I/O voltage 3.3 V

1.7 Developer environment

The following development support is available:

- Automotive Evaluation Boards (EVBs) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG and Nexus interfaces
- Autocode generation tools
- Initialization tools

1.8 Package

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: www.st.com. ECOPACK® is an ST trademark.

SPC560P40/34 family members are offered in the following package types:

- 64-pin LQFP, 0.5 mm pitch, 10 mm × 10 mm outline
- 100-pin LQFP, 0.5 mm pitch, 14 mm × 14 mm outline

2 SPC560P40/34 memory map

Table 3 shows the memory map for the SPC560P40/34. All addresses on the SPC560P40/34, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

Table 3. Memory map

Start address	End address	Size (KB)	Region name
On-chip memory			
0x0000_0000	0x0003_FFFF	256	Code Flash Array 0
0x0004_0000	0x001F_FFFF	1792	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash Array 0 Shadow Sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash Array 0 Test Sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash Array 0
0x0081_0000	0x00C0_1FFF	4040	Reserved
0x00C0_2000	0x00C0_3FFF	8	Data Flash Array 0 Test Sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Flash Emulation Mapping
0x2000_0000	0x3FFF_FFFF	524288	Reserved
0x4000_0000	0x4000_4FFF	20	SRAM
0x4000_5000	0xC3F8_0000	1048536	Reserved
On-chip peripherals			
0xC3F8_0000	0xC3F8_7FFF	32	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code Flash 0 Configuration (CFLASH_0)
0xC3F8_C000	0xC3F8_FFFF	16	Data Flash 0 Configuration (DFLASH_0)
0xC3F9_0000	0xC3F9_3FFF	16	System Integration Unit Lite (SIUL)
0xC3F9_4000	0xC3F9_7FFF	16	WakeUp Unit (WKUP)
0xC3F9_8000	0xC3FD_7FFF	256	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	System Status and Configuration Module (SSCM)
0xC3FD_C000	0xC3FD_FFFF	16	Mode Entry module (ME)
0xC3FE_0000	0xC3FE_3FFF	16	Clock Generation Module (CGM, XOSC, IRC, FMPLL_0, CMU0)
0xC3FE_4000	0xC3FE_7FFF	16	Reset Generation Module (RGM)
0xC3FE_8000	0xC3FE_BFFF	16	Power Control Unit (PCU)(1)
0xC3FE_C000	0xC3FE_FFFF	16	Reserved

Table 3. Memory map (continued)

Start address	End address	Size (KB)	Region name
0xC3FF_0000	0xC3FF_3FFF	16	Periodic Interrupt Timer (PIT)
0xC3FF_4000	0xC3FF_FFFF	48	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	Analog to Digital Converter 0 (ADC_0)
0xFFE0_4000	0xFFE0_BFFF	32	Reserved
0xFFE0_C000	0xFFE0_FFFF	16	CTU_0
0xFFE1_0000	0xFFE1_7FFF	32	Reserved
0xFFE1_8000	0xFFE1_BFFF	16	eTimer_0
0xFFE1_C000	0xFFE2_3FFF	32	Reserved
0xFFE2_4000	0xFFE2_7FFF	16	FlexPWM_0
0xFFE2_8000	0xFFE3_FFFF	96	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlex_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlex_1
0xFFE5_0000	0xFFE6_7FFF	128	Reserved
0xFFE6_8000	0xFFE6_BFFF	16	Cyclic Redundancy Check (CRC)
0xFFE6_C000	0xFFE6_FFFF	16	Fault Collection Unit (FCU)
0xFFE7_0000	0xFFE7_FFFF	64	Reserved
0xFFE8_0000	0xFFEF_FFFF	512	Mirrored (range 0xC3F8_0000 – 0xC3FF_FFFF)
0xFFF0_0000	0xFFF3_7FFF	224	Reserved
0xFFF3_8000	0xFFF3_BFFF	16	Software Watchdog (SWT_0)
0xFFF3_C000	0xFFF3_FFFF	16	System Timer Module (STM_0)
0xFFF4_0000	0xFFF4_3FFF	16	Error Correction Status Module (ECSM)
0xFFF4_4000	0xFFF4_7FFF	16	Enhanced Direct Memory Access Controller (eDMA)
0xFFF4_8000	0xFFF4_BFFF	16	Interrupt Controller (INTC)
0xFFF4_C000	0xFFF8_FFFF	272	Reserved
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_0
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_1
0xFFF9_8000	0xFFF9_BFFF	16	DSPI_2
0xFFF9_C000	0xFFFB_FFFF	144	Reserved
0xFFFC_0000	0xFFFC_3FFF	16	FlexCAN_0 (CAN0)
0xFFFC_4000	0xFFFD_BFFF	96	Reserved
0xFFFD_C000	0xFFFD_FFFF	16	DMA Multiplexer (DMA_MUX)
0xFFFE_0000	0xFFFE_7FFF	32	Reserved
0xFFFE_8000	0xFFFE_BFFF	16	Safety Port (FlexCAN)

Table 3. Memory map (continued)

Start address	End address	Size (KB)	Region name
0xFFFE_C000	0xFFFF_BFFF	64	Reserved
0xFFFF_C000	0xFFFF_FFFF	16	Boot Assist Module (BAM)

1. This address space contains also VREG registers. See *34, "Voltage Regulators and Power Supplies."*

3 Signal Description

This chapter describes the signals of the SPC560P40/34. It includes a table of signal properties and detailed descriptions of signals.

3.1 100-pin LQFP pinout

Figure 4 and *Figure 5* shows the pinout of the 100-pin LQFP.

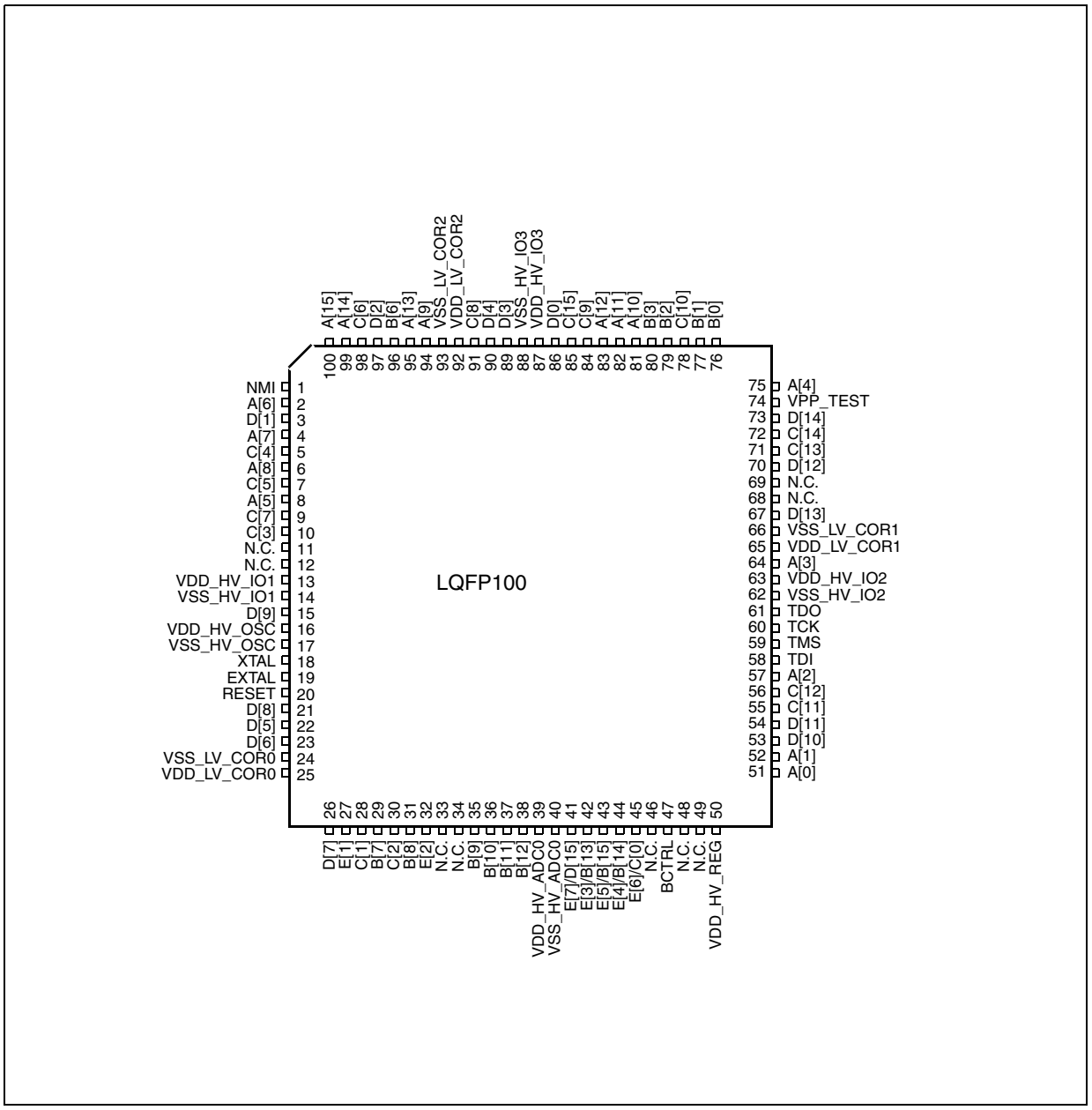


Figure 4. 100-pin LQFP pinout – Full featured configuration (top view)

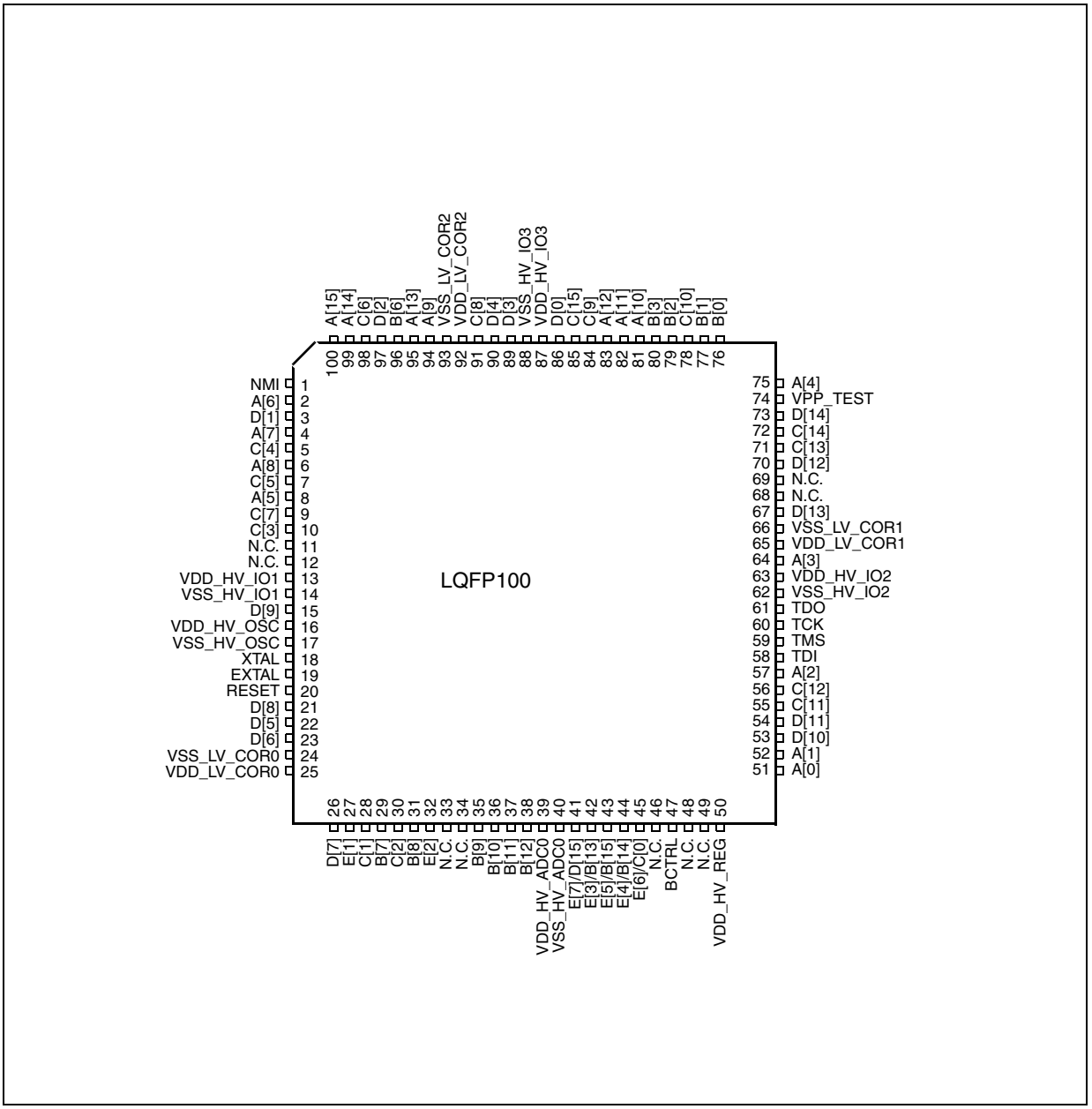


Figure 5. 100-pin LQFP pinout – Airbag configuration (top view)

3.2 64-pin LQFP pinout

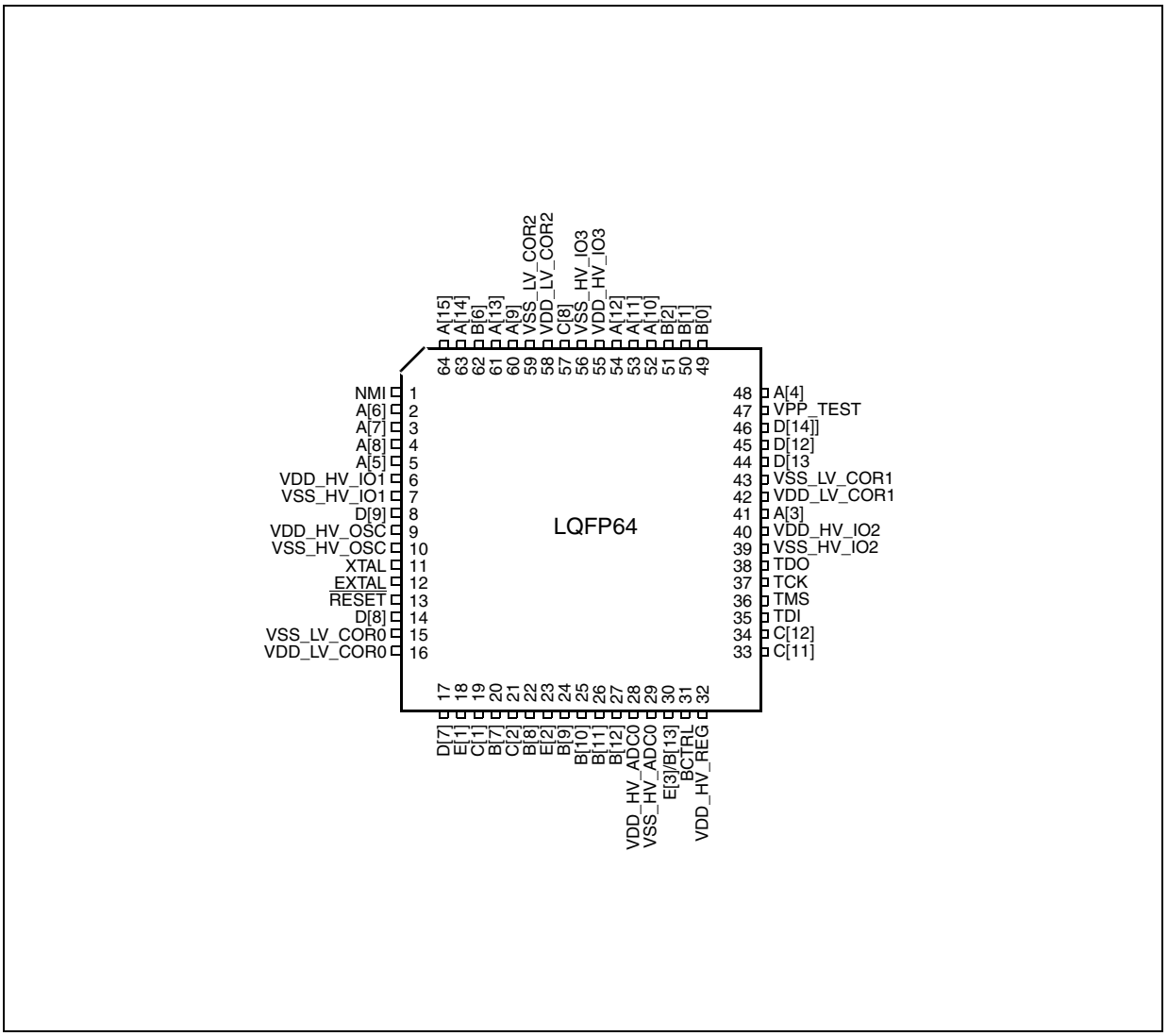


Figure 6. 64-pin LQFP pinout – Full featured configuration (top view)

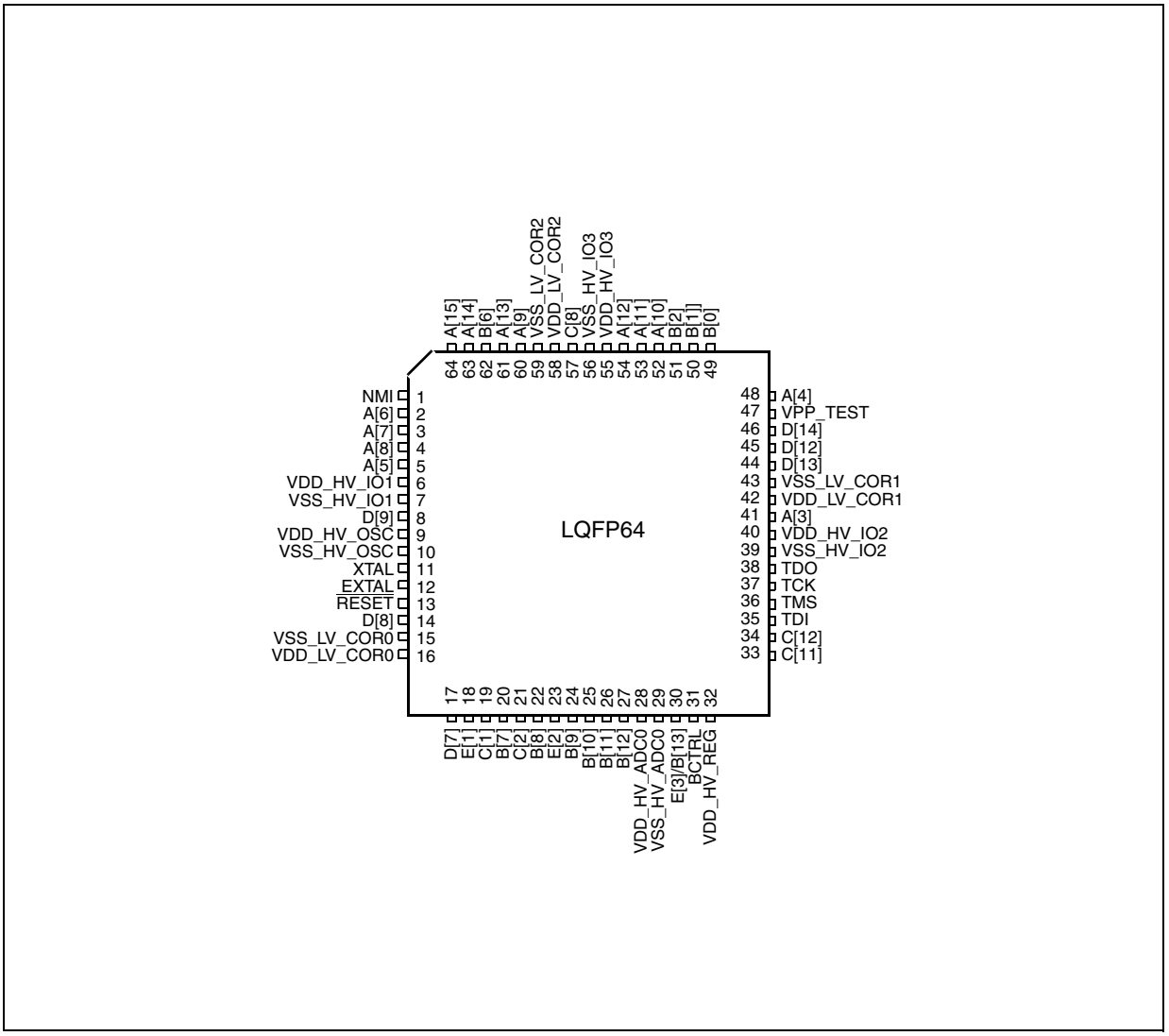


Figure 7. 64-pin LQFP pinout – Airbag configuration (top view)

3.3 Pin description

The following sections provide signal descriptions and related information about the functionality and configuration of the SPC560P40/34 devices.

3.3.1 Power supply and reference voltage pins

Table 4 lists the power supply and reference voltage for the SPC560P40/34 devices.

Table 4. Supply pins

Supply		Pin	
Symbol	**Description**	**64-pin**	**100-pin**
VREG control and power supply pins. Pins available on 64-pin and 100-pin packages			
BCTRL	Voltage regulator external NPN ballast base control pin	31	47
$V_{DD\_HV\_REG}$ (3.3 V or 5.0 V)	Voltage regulator supply voltage	32	50
ADC_0 reference and supply voltage. Pins available on 64-pin and 100-pin packages			
$V_{DD\_HV\_ADC0}$(1)	ADC_0 supply and high reference voltage	28	39
$V_{SS\_HV\_ADC0}$	ADC_0 ground and low reference voltage	29	40
Power supply pins (3.3 V or 5.0 V). Pins available on 64-pin and 100-pin packages			
$V_{DD\_HV\_IO1}$	Input/output supply voltage	6	13
$V_{SS\_HV\_IO1}$	Input/output ground	7	14
$V_{DD\_HV\_IO2}$	Input/output supply voltage and data Flash memory supply voltage	40	63
$V_{SS\_HV\_IO2}$	Input/output ground and Flash memory HV ground	39	62
$V_{DD\_HV\_IO3}$	Input/output supply voltage and code Flash memory supply voltage	55	87
$V_{SS\_HV\_IO3}$	Input/output ground and code Flash memory HV ground	56	88
$V_{DD\_HV\_OSC}$	Crystal oscillator amplifier supply voltage	9	16
$V_{SS\_HV\_OSC}$	Crystal oscillator amplifier ground	10	17
Power supply pins (1.2 V). Pins available on 64-pin and 100-pin packages			
$V_{DD\_LV\_COR0}$	1.2 V supply pins for core logic and PLL. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	16	25
$V_{SS\_LV\_COR0}$	1.2 V supply pins for core logic and PLL. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR}$ pin.	15	24
$V_{DD\_LV\_COR1}$	1.2 V supply pins for core logic and data Flash. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	42	65
$V_{SS\_LV\_COR1}$	1.2 V supply pins for core logic and data Flash. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR}$ pin.	43	66
$V_{DD\_LV\_COR2}$	1.2 V supply pins for core logic and code Flash. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	58	92
$V_{SS\_LV\_COR2}$	1.2 V supply pins for core logic and code Flash. Decoupling capacitor must be connected betwee.n these pins and the nearest $V_{DD\_LV\_COR}$ pin.	59	93

1. Analog supply/ground and high/low reference lines are internally physically separate, but are shorted via a double-bonding connection on $V_{DD\_HV\_ADCx}$/$V_{SS\_HV\_ADCx}$ pins.

3.3.2 System pins

Table 5 and *Table 6* contain information on pin functions for the SPC560P40/34 devices. The pins listed in *Table 5* are single-function pins. The pins shown in *Table 6* are multi-function pins, programmable via their respective pad configuration register (PCR) values.

Table 5. System pins

Symbol	Description	Direction	Pad speed(1)		Pin	
			SRC = 0	SRC = 1	64-pin	100-pin
Dedicated pins						
NMI	Non-maskable Interrupt	Input only	Slow	—	1	1
XTAL	Analog output of the oscillator amplifier circuit—needs to be grounded if oscillator is used in bypass mode	—	—	—	11	18
EXTAL	Analog input of the oscillator amplifier circuit, when the oscillator is not in bypass mode Analog input for the clock generator when the oscillator is in bypass mode	—	—	—	12	19
TDI	JTAG test data input	Input only	Slow	—	35	58
TMS	JTAG state machine control	Input only	Slow	—	36	59
TCK	JTAG clock	Input only	Slow	—	37	60
TDO	JTAG test data output	Output only	Slow	Fast	38	61
Reset pin						
$\overline{\text{RESET}}$	Bidirectional reset with Schmitt trigger characteristics and noise filter	Bidirectional	Medium	—	13	20
Test pin						
VPP_TEST	Pin for testing purpose only. To be tied to ground in normal operating mode.	—	—	—	47	74

1. SRC values refer to the value assigned to the Slew Rate Control bits of the pad configuration register.

3.3.3 Pin multiplexing

Table 6 defines the pin list and muxing for the SPC560P40/34 devices.

Each row of *Table 6* shows all the possible ways of configuring each pin, via alternate functions. The default function assigned to each pin after reset is the ALT0 function.

SPC560P40/34 devices provide three main I/O pad types, depending on the associated functions:

- *Slow pads* are the most common, providing a compromise between transition time and low electromagnetic emission.
- *Medium pads* provide fast enough transition for serial communication channels with controlled current to reduce electromagnetic emission.
- *Fast pads* provide maximum speed. They are used for improved NEXUS debugging capability.

Medium and Fast pads can use slow configuration to reduce electromagnetic emission, at the cost of reducing AC performance. For more information, see “Pad AC Specifications” in the device datasheet.

Table 6. Pin muxing

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
Port A (16-bit)									
A[0]	PCR[0]	ALT0 ALT1 ALT2 ALT3 —	GPIO[0] ETC[0] SCK F[0] EIRQ[0]	SIUL eTimer_0 DSPI_2 FCU_0 SIUL	I/O I/O I/O O I	Slow	Medium	—	51
A[1]	PCR[1]	ALT0 ALT1 ALT2 ALT3 —	GPIO[1] ETC[1] SOUT F[1] EIRQ[1]	SIUL eTimer_0 DSPI_2 FCU_0 SIUL	I/O I/O O O I	Slow	Medium	—	52
A[2]	PCR[2]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[2] ETC[2] — A[3] SIN ABS[0] EIRQ[2]	SIUL eTimer_0 — FlexPWM_0 DSPI_2 MC_RGM SIUL	I/O I/O — O I I I	Slow	Medium	—	57
A[3]	PCR[3]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[3] ETC[3] CS0 B[3] ABS[1] EIRQ[3]	SIUL eTimer_0 DSPI_2 FlexPWM_0 MC_RGM SIUL	I/O I/O I/O O I I	Slow	Medium	41	64
A[4]	PCR[4]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[4] — CS1 ETC[4] FAB EIRQ[4]	SIUL — DSPI_2 eTimer_0 MC_RGM SIUL	I/O — O I/O I I	Slow	Medium	48	75
A[5]	PCR[5]	ALT0 ALT1 ALT2 ALT3 —	GPIO[5] CS0 — CS7 EIRQ[5]	SIUL DSPI_1 — DSPI_0 SIUL	I/O I/O — O I	Slow	Medium	5	8
A[6]	PCR[6]	ALT0 ALT1 ALT2 ALT3 —	GPIO[6] SCK — — EIRQ[6]	SIUL DSPI_1 — — SIUL	I/O I/O — — I	Slow	Medium	2	2

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
A[7]	PCR[7]	ALT0 ALT1 ALT2 ALT3 —	GPIO[7] SOUT — — EIRQ[7]	SIUL DSPI_1 — — SIUL	I/O O — — I	Slow	Medium	3	4
A[8]	PCR[8]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[8] — — — SIN EIRQ[8]	SIUL — — — DSPI_1 SIUL	I/O — — — I I	Slow	Medium	4	6
A[9]	PCR[9]	ALT0 ALT1 ALT2 ALT3 —	GPIO[9] CS1 — B[3] FAULT[0]	SIUL DSPI_2 — FlexPWM_0 FlexPWM_0	I/O O — O I	Slow	Medium	60	94
A[10]	PCR[10]	ALT0 ALT1 ALT2 ALT3 —	GPIO[10] CS0 B[0] X[2] EIRQ[9]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O I/O O O I	Slow	Medium	52	81
A[11]	PCR[11]	ALT0 ALT1 ALT2 ALT3 —	GPIO[11] SCK A[0] A[2] EIRQ[10]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O I/O O O I	Slow	Medium	53	82
A[12]	PCR[12]	ALT0 ALT1 ALT2 ALT3 —	GPIO[12] SOUT A[2] B[2] EIRQ[11]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O O O O I	Slow	Medium	54	83
A[13]	PCR[13]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[13] — B[2] — SIN FAULT[0] EIRQ[12]	SIUL — FlexPWM_0 — DSPI_2 FlexPWM_0 SIUL	I/O — O — I I I	Slow	Medium	61	95

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
A[14]	PCR[14]	ALT0 ALT1 ALT2 ALT3 —	GPIO[14] TXD — — EIRQ[13]	SIUL Safety Port_0 — — SIUL	I/O O — — I	Slow	Medium	63	99
A[15]	PCR[15]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[15] — — — RXD EIRQ[14]	SIUL — — — Safety Port_0 SIUL	I/O — — — I I	Slow	Medium	64	100
Port B (16-bit)									
B[0]	PCR[16]	ALT0 ALT1 ALT2 ALT3 —	GPIO[16] TXD — DEBUG[0] EIRQ[15]	SIUL FlexCAN_0 — SSCM SIUL	I/O O — — I	Slow	Medium	49	76
B[1]	PCR[17]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[17] — — DEBUG[1] RXD EIRQ[16]	SIUL — — SSCM FlexCAN_0 SIUL	I/O — — — I I	Slow	Medium	50	77
B[2]	PCR[18]	ALT0 ALT1 ALT2 ALT3 —	GPIO[18] TXD — DEBUG[2] EIRQ[17]	SIUL LIN_0 — SSCM SIUL	I/O O — — I	Slow	Medium	51	79
B[3]	PCR[19]	ALT0 ALT1 ALT2 ALT3 —	GPIO[19] — — DEBUG[3] RXD	SIUL — — SSCM LIN_0	I/O — — — I	Slow	Medium	—	80
B[6]	PCR[22]	ALT0 ALT1 ALT2 ALT3 —	GPIO[22] CLKOUT CS2 — EIRQ[18]	SIUL Control DSPI_2 — SIUL	I/O O O — I	Slow	Medium	62	96

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
B[7]	PCR[23]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[23] — — — AN[0] RXD	SIUL — — — ADC_0 LIN_0	Input only	—	—	20	29
B[8]	PCR[24]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[24] — — — AN[1] ETC[5]	SIUL — — — ADC_0 eTimer_0	Input only	—	—	22	31
B[9]	PCR[25]	ALT0 ALT1 ALT2 ALT3 —	GPIO[25] — — — AN[11]	SIUL — — — ADC_0	Input only	—	—	24	35
B[10]	PCR[26]	ALT0 ALT1 ALT2 ALT3 —	GPIO[26] — — — AN[12]	SIUL — — — ADC_0	Input only	—	—	25	36
B[11]	PCR[27]	ALT0 ALT1 ALT2 ALT3 —	GPIO[27] — — — AN[13]	SIUL — — — ADC_0	Input only	—	—	26	37
B[12]	PCR[28]	ALT0 ALT1 ALT2 ALT3 —	GPIO[28] — — — AN[14]	SIUL — — — ADC_0	Input only	—	—	27	38
B[13]	PCR[29]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[29] — — — AN[6] emu. AN[0] RXD	SIUL — — — ADC_0 emu. ADC_1[6] LIN_1	Input only	—	—	30	42

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
B[14]	PCR[30]	ALT0 ALT1 ALT2 ALT3 — — — —	GPIO[30] — — — AN[7] emu. AN[1] ETC[4] EIRQ[19]	SIUL — — — ADC_0 emu. ADC_1[6] eTimer_0 SIUL	Input only	—	—	—	44
B[15]	PCR[31]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[31] — — — AN[8] emu. AN[2] EIRQ[20]	SIUL — — — ADC_0 emu. ADC_1[6] SIUL	Input only	—	—	—	43
Port C (16-bit)									
C[0]	PCR[32]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[32] — — — AN[9] emu. AN[3]	SIUL — — — ADC_0 emu. ADC_1[6]	Input only	—	—	—	45
C[1]	PCR[33]	ALT0 ALT1 ALT2 ALT3 —	GPIO[33] — — — AN[2]	SIUL — — — ADC_0	Input only	—	—	19	28
C[2]	PCR[34]	ALT0 ALT1 ALT2 ALT3 —	GPIO[34] — — — AN[3]	SIUL — — — ADC_0	Input only	—	—	21	30
C[3]	PCR[35]	ALT0 ALT1 ALT2 ALT3 —	GPIO[35] CS1 — TXD EIRQ[21]	SIUL DSPI_0 — LIN_1 SIUL	I/O O — O I	Slow	Medium	—	10

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
C[4]	PCR[36]	ALT0 ALT1 ALT2 ALT3 —	GPIO[36] CS0 X[1] DEBUG[4] EIRQ[22]	SIUL DSPI_0 FlexPWM_0 SSCM SIUL	I/O I/O O — I	Slow	Medium	—	5
C[5]	PCR[37]	ALT0 ALT1 ALT2 ALT3 —	GPIO[37] SCK — DEBUG[5] EIRQ[23]	SIUL DSPI_0 — SSCM SIUL	I/O I/O — — I	Slow	Medium	—	7
C[6]	PCR[38]	ALT0 ALT1 ALT2 ALT3 —	GPIO[38] SOUT B[1] DEBUG[6] EIRQ[24]	SIUL DSPI_0 FlexPWM_0 SSCM SIUL	I/O O O — I	Slow	Medium	—	98
C[7]	PCR[39]	ALT0 ALT1 ALT2 ALT3 —	GPIO[39] — A[1] DEBUG[7] SIN	SIUL — FlexPWM_0 SSCM DSPI_0	I/O — O — I	Slow	Medium	—	9
C[8]	PCR[40]	ALT0 ALT1 ALT2 ALT3	GPIO[40] CS1 — CS6	SIUL DSPI_1 — DSPI_0	I/O O — O	Slow	Medium	57	91
C[9]	PCR[41]	ALT0 ALT1 ALT2 ALT3	GPIO[41] CS3 — X[3]	SIUL DSPI_2 — FlexPWM_0	I/O O — O	Slow	Medium	—	84
C[10]	PCR[42]	ALT0 ALT1 ALT2 ALT3 —	GPIO[42] CS2 — A[3] FAULT[1]	SIUL DSPI_2 — FlexPWM_0 FlexPWM_0	I/O O — O I	Slow	Medium	—	78
C[11]	PCR[43]	ALT0 ALT1 ALT2 ALT3	GPIO[43] ETC[4] CS2 —	SIUL eTimer_0 DSPI_2 —	I/O I/O O —	Slow	Medium	33	55
C[12]	PCR[44]	ALT0 ALT1 ALT2 ALT3	GPIO[44] ETC[5] CS3 —	SIUL eTimer_0 DSPI_2 —	I/O I/O O —	Slow	Medium	34	56

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
C[13]	PCR[45]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[45] — — — EXT_IN EXT_SYNC	SIUL — — — CTU_0 FlexPWM_0	I/O — — — I I	Slow	Medium	—	71
C[14]	PCR[46]	ALT0 ALT1 ALT2 ALT3	GPIO[46] — EXT_TGR —	SIUL — CTU_0 —	I/O — O —	Slow	Medium	—	72
C[15]	PCR[47]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[47] — — A[1] EXT_IN EXT_SYNC	SIUL — — FlexPWM_0 CTU_0 FlexPWM_0	I/O — — O I I	Slow	Medium	—	85
Port D (16-bit)									
D[0]	PCR[48]	ALT0 ALT1 ALT2 ALT3	GPIO[48] — — B[1]	SIUL — — FlexPWM_0	I/O — — O	Slow	Medium	—	86
D[1]	PCR[49]	ALT0 ALT1 ALT2 ALT3	GPIO[49] — — EXT_TRG	SIUL — — CTU_0	I/O — — O	Slow	Medium	—	3
D[2]	PCR[50]	ALT0 ALT1 ALT2 ALT3	GPIO[50] — — X[3]	SIUL — — FlexPWM_0	I/O — — O	Slow	Medium	—	97
D[3]	PCR[51]	ALT0 ALT1 ALT2 ALT3	GPIO[51] — — A[3]	SIUL — — FlexPWM_0	I/O — — O	Slow	Medium	—	89
D[4]	PCR[52]	ALT0 ALT1 ALT2 ALT3	GPIO[52] — — B[3]	SIUL — — FlexPWM_0	I/O — — O	Slow	Medium	—	90
D[5]	PCR[53]	ALT0 ALT1 ALT2 ALT3	GPIO[53] CS3 F[0] —	SIUL DSPI_0 FCU_0 —	I/O O O —	Slow	Medium	—	22

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
D[6]	PCR[54]	ALT0 ALT1 ALT2 ALT3 —	GPIO[54] CS2 — — FAULT[1]	SIUL DSPI_0 — — FlexPWM_0	I/O O — — I	Slow	Medium	—	23
D[7]	PCR[55]	ALT0 ALT1 ALT2 ALT3	GPIO[55] CS3 F[1] CS4	SIUL DSPI_1 FCU_0 DSPI_0	I/O O O O	Slow	Medium	17	26
D[8]	PCR[56]	ALT0 ALT1 ALT2 ALT3	GPIO[56] CS2 — CS5	SIUL DSPI_1 — DSPI_0	I/O O — O	Slow	Medium	14	21
D[9]	PCR[57]	ALT0 ALT1 ALT2 ALT3	GPIO[57] X[0] TXD —	SIUL FlexPWM_0 LIN_1 —	I/O O O —	Slow	Medium	8	15
D[10]	PCR[58]	ALT0 ALT1 ALT2 ALT3	GPIO[58] A[0] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	53
D[11]	PCR[59]	ALT0 ALT1 ALT2 ALT3	GPIO[59] B[0] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	54
D[12]	PCR[60]	ALT0 ALT1 ALT2 ALT3 —	GPIO[60] X[1] — — RXD	SIUL FlexPWM_0 — — LIN_1	I/O O — — I	Slow	Medium	45	70
D[13]	PCR[61]	ALT0 ALT1 ALT2 ALT3	GPIO[61] A[1] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	44	67
D[14]	PCR[62]	ALT0 ALT1 ALT2 ALT3	GPIO[62] B[1] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	46	73

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
D[15]	PCR[63]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[63] — — — AN[10] emu. AN[4]	SIUL — — — ADC_0 emu. ADC_1[6]	Input only	—	—	—	41
Port E (16-bit)									
E[1]	PCR[65]	ALT0 ALT1 ALT2 ALT3 —	GPIO[65] — — — AN[4]	SIUL — — — ADC_0	Input only	—	—	18	27
E[2]	PCR[66]	ALT0 ALT1 ALT2 ALT3 —	GPIO[66] — — — AN[5]	SIUL — — — ADC_0	Input only	—	—	23	32
E[3]	PCR[67]	ALT0 ALT1 ALT2 ALT3 —	GPIO[67] — — — AN[6]	SIUL — — — ADC_0	Input only	—	—	30	42
E[4]	PCR[68]	ALT0 ALT1 ALT2 ALT3 —	GPIO[68] — — — AN[7]	SIUL — — — ADC_0	Input only	—	—	—	44
E[5]	PCR[69]	ALT0 ALT1 ALT2 ALT3 —	GPIO[69] — — — AN[8]	SIUL — — — ADC_0	Input only	—	—	—	43

Table 6. Pin muxing (continued)

Port pin	PCR register	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction[4]	Pad speed[5]		Pin	
						SRC = 0	SRC = 1	64-pin	100-pin
E[6]	PCR[70]	ALT0 ALT1 ALT2 ALT3 —	GPIO[70] — — — AN[9]	SIUL — — — ADC_0	Input only	—	—	—	45
E[7]	PCR[71]	ALT0 ALT1 ALT2 ALT3 —	GPIO[71] — — — AN[10]	SIUL — — — ADC_0	Input only	—	—	—	41

1. ALT0 is the primary (default) function for each port after reset.
2. Alternate functions are chosen by setting the values of the PCR.PA bitfields inside the SIU module. PCR.PA = 00 → ALT0; PCR.PA = 01 → ALT1; PCR.PA = 10 → ALT2; PCR.PA = 11 → ALT3. This is intended to select the output functions; to use one of the input functions, the PCR.IBE bit must be written to '1', regardless of the values selected in the PCR.PA bitfields. For this reason, the value corresponding to an input only function is reported as "—".
3. Module included on the MCU.
4. Multiple inputs are routed to all respective modules internally. The input of some modules must be configured by setting the values of the PSMIO.PADSELx bitfields inside the SIUL module.
5. Programmable via the SRC (Slew Rate Control) bits in the respective Pad Configuration Register.
6. ADC0.AN emulates ADC1.AN. This feature is used to provide software compatibility between SPC560P40/34 and SPC560P50. Refer to ADC chapter of reference manual for more details.

3.4 CTU / ADC / FlexPWM / eTimer connections

Figure 8 shows the interconnections between the CTU, ADC, FlexPWM, and eTimer.

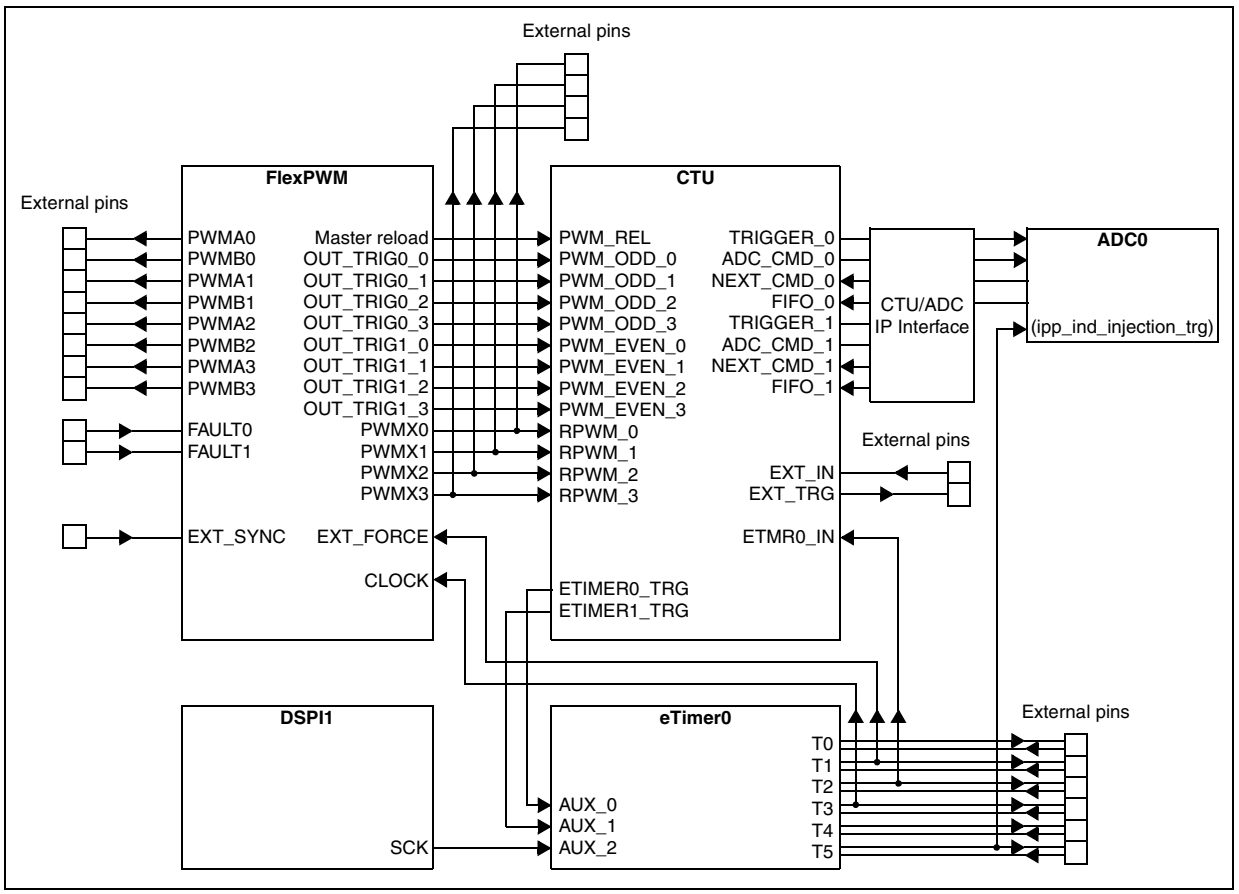


Figure 8. CTU / ADC / FlexPWM / eTimer connections

Table 7. CTU / ADC / FlexPWM / eTimer connections

Source module (Signal)	Target module (Signal)	Comment
PWM (Master Reload)	CTU (PWM Reload)	From PWM sub-module 0
PWM (OUT_TRIG0_0)	CTU (PWM_ODD_0)	OUT_TRIG0 sub-module 0
PWM (OUT_TRIG1_0)	CTU (PWM_EVEN_0)	OUT_TRIG1 sub-module 0
PWM (PWMX0)	CTU (PWM_REAL_0)	—
PWM (OUT_TRIG0_1)	CTU (PWM_ODD_1)	OUT_TRIG0 sub-module 1
PWM (OUT_TRIG1_1)	CTU (PWM_EVEN_1)	OUT_TRIG1 sub-module 1
PWM (PWMX1)	CTU (PWM_REAL_1)	—
PWM (OUT_TRIG0_2)	CTU (PWM_ODD_2)	OUT_TRIG0 sub-module 2
PWM (OUT_TRIG1_2)	CTU (PWM_EVEN_2)	OUT_TRIG1 sub-module 2
PWM (PWMX2)	CTU (PWM_REAL_2)	—

Table 7. CTU / ADC / FlexPWM / eTimer connections (continued)

Source module (Signal)	Target module (Signal)	Comment
PWM (OUT_TRIG0_3)	CTU (PWM_ODD_3)	OUT_TRIG0 sub-module 3
PWM (OUT_TRIG1_3)	CTU (PWM_EVEN_3)	OUT_TRIG1 sub-module 3
PWM (PWMX3)	CTU (PWM_REAL_3)	—
PWM (PWMA0)	SIU lite	—
PWM (PWMB0)	SIU lite	—
PWM (PWMX1)	SIU lite	—
PWM (PWMA1)	SIU lite	—
PWM (PWMB1)	SIU lite	—
PWM (PWMX2)	SIU lite	—
PWM (PWMA2)	SIU lite	—
PWM (PWMB2)	SIU lite	—
PWM (PWMX3)	SIU lite	—
PWM (PWMA3)	SIU lite	—
PWM (PWMB3)	SIU lite	—
PWM (PWMX3)	SIU lite	—
eTimer_0 (T1)	PWM (EXT_FORCE)	—
eTimer_0 (T2)	CTU (ETMR0_IN)	—
eTimer_0 (T5)	ADC_0	ADC injected conversion request signal (for non CTU mode of operation)
CTU (ETIMER0_TRG)	eTimer_0 (AUX_0)	—
CTU (ETIMER1_TRG)	eTimer_0 (AUX_1)	—
CTU (TRIGGER_0)	ADC_0 (through CTU/ADC IP Interface)	—
CTU (TRIGGER_1)	Virtual ADC_1 (through CTU/ADC IP Interface)	—
CTU (ADC_CMD_0)	ADC_0 (through CTU/ADC IP Interface)	16-bit signal
CTU (ADC_CMD_1)	Virtual ADC_1 (through CTU/ADC IP Interface)	16-bit signal
CTU (EXT_TGR)	SIU lite	—
ADC_0 (EOC)	CTU (NEXT_CMD_0)	End Of Conversion should be used as next command request signal
Virtual ADC_1 (EOC)	CTU (NEXT_CMD_1) (through CTU/ADC IP Interface)	End Of Conversion should be used as next command request signal
ADC_0	CTU (FIFO_0)	18-bit signal
Virtual ADC_1	CTU (FIFO_1) (through CTU/ADC IP Interface)	18-bit signal

Table 7. CTU / ADC / FlexPWM / eTimer connections (continued)

Source module (Signal)	Target module (Signal)	Comment
SIU lite	CTU (EXT_IN)	The same GPIO pin as used for CTU (EXT_IN) and the PWM (EXT_SYNC)
SIU lite	PWM (EXT_SYNC)	The same GPIO pin as used for CTU (EXT_IN) and the PWM (EXT_SYNC)
SIU lite	PWM (FAULT0)	—
SIU lite	PWM (FAULT1)	—
DSPI_1 (SCK)	eTimer_0 (AUX_2)	—

4 Clock Description

This chapter describes the clock architectural implementation for SPC560P40/34.

The following clock related modules are implemented on the SPC560P40/34:

- Clock, Reset, and Mode Handling
  - Clock Generation Module (CGM) (see *Chapter 5: Clock Generation Module (MC_CGM)*)
  - Reset Generation Module (RGM) (see *Chapter 8: Reset Generation Module (MC_RGM)*)
  - Mode Entry Module (ME) (see *Chapter 6: Mode Entry Module (MC_ME)*)
- High Frequency Oscillator (XOSC) (see *Section 4.7, "XOSC external crystal oscillator*)
- High Frequency RC Oscillator (IRC) (see *Section 4.6, "IRC 16 MHz internal RC oscillator (RC_CTL)*)
- FMPLL (FMPLL_0) (see *Section 4.8, "Frequency Modulated Phase Locked Loop (FMPLL)*)
- CMU (CMU_0) (see *Section 4.9, "Clock Monitor Unit (CMU)*)
- Periodic Interrupt Timer (PIT) (see *30, "Periodic Interrupt Timer (PIT)*)
- System Timer Module (STM_0) (see *31, "System Timer Module (STM)*)
- Software Watchdog Timer (SWT_0) (see *Section 27.3, "Software Watchdog Timer (SWT)*)

4.1 Clock architecture

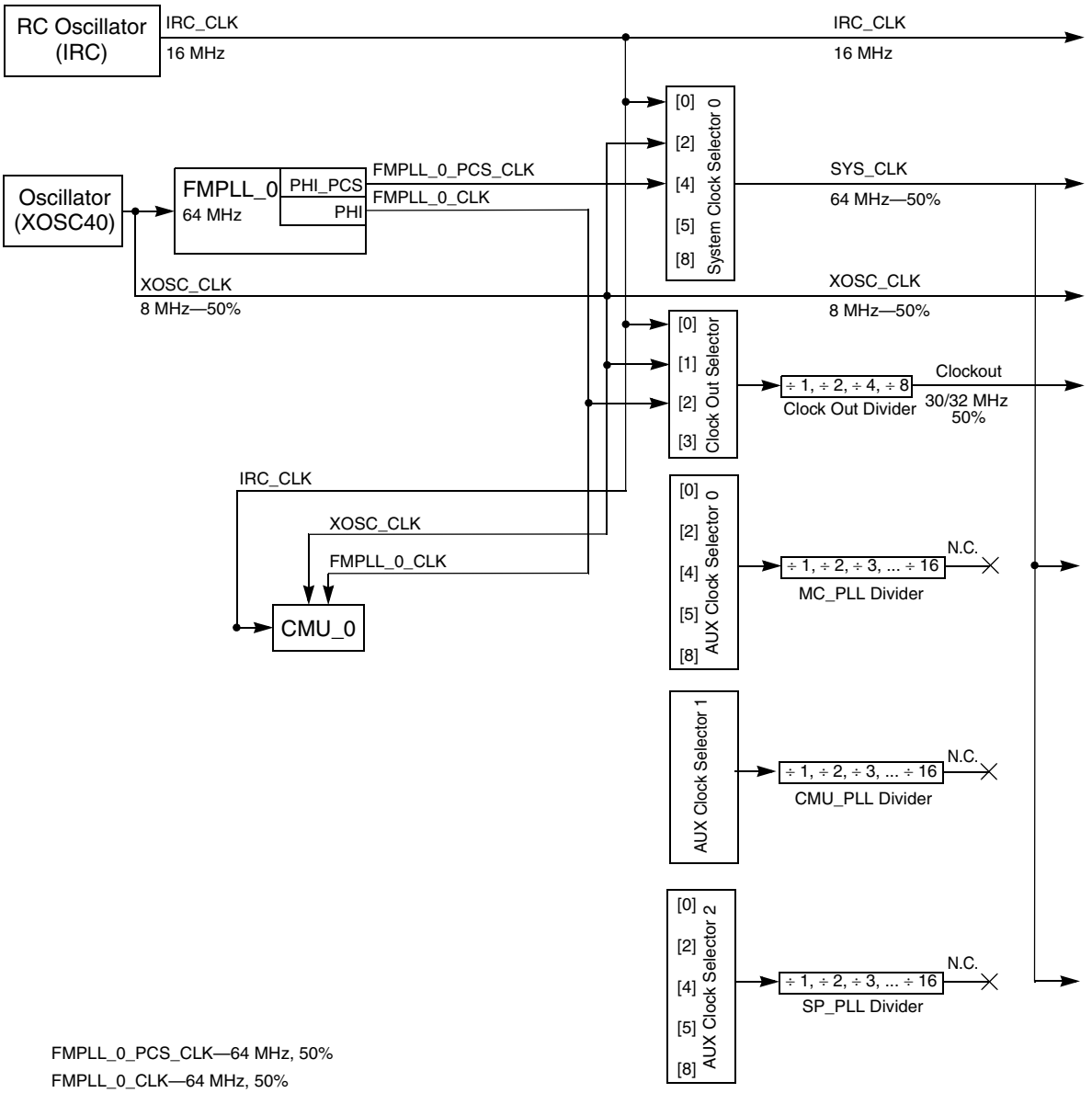
The system and peripheral clocks are generated from three sources:

- IRC—internal RC oscillator clock
- XOSC—oscillator clock
- FMPLL_0 clock output

The clock architecture is shown in *Figure 9*, *Figure 10*, and *Figure 11*.

The frequencies shown in *Figure 9* represent only one possible setting.

Note: MC_PLL_CLK and SP_PLL_CLK are SYS_CLK.



FMPLL_0_PCS_CLK—64 MHz, 50%
FMPLL_0_CLK—64 MHz, 50%
SYS_CLK = System Clock

NOTE: FlexRay protocol clock does not support IRC as a clock source.

Figure 9. SPC560P40/34 system clock generation

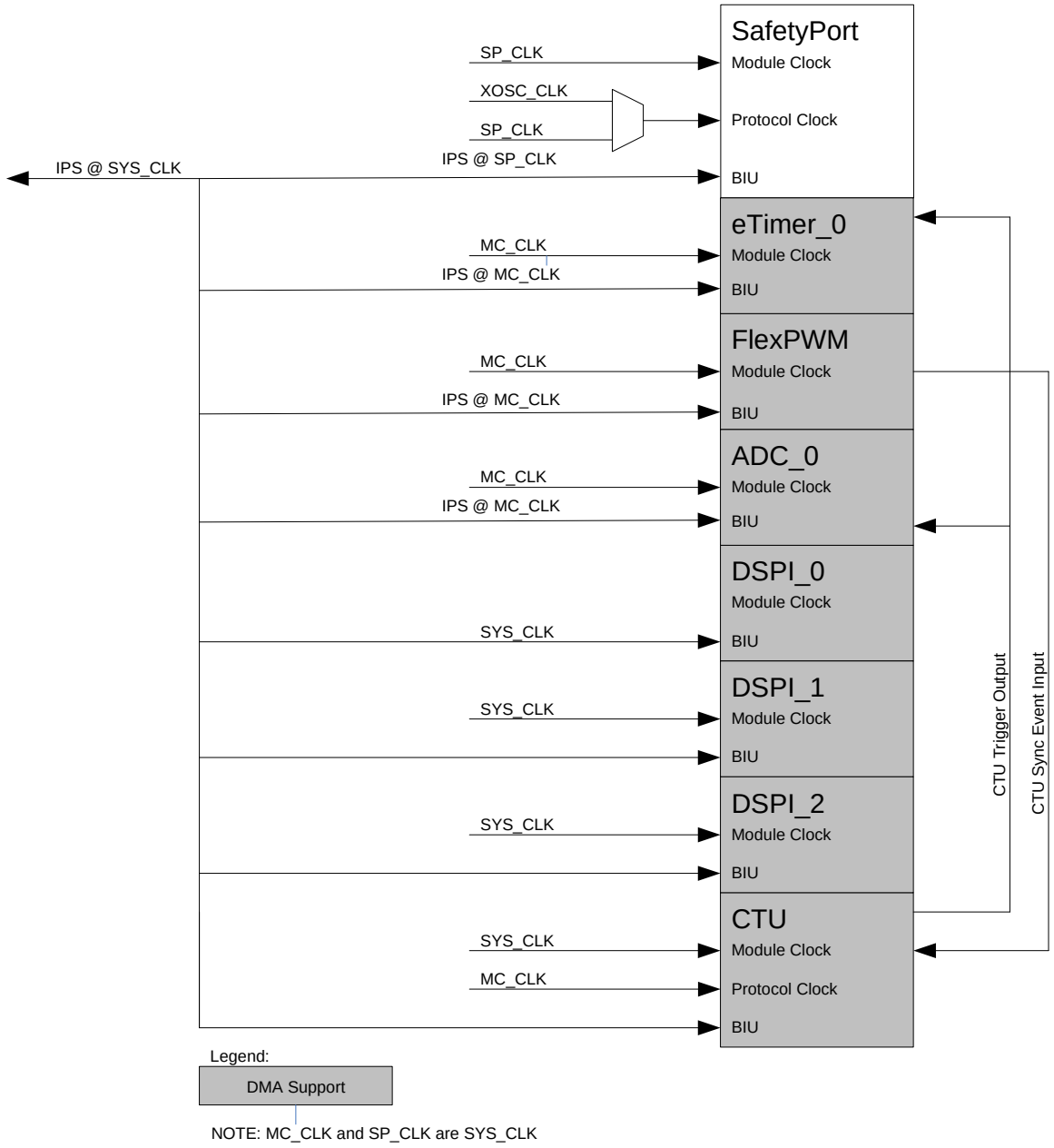


Figure 10. SPC560P40/34 system clock distribution Part A

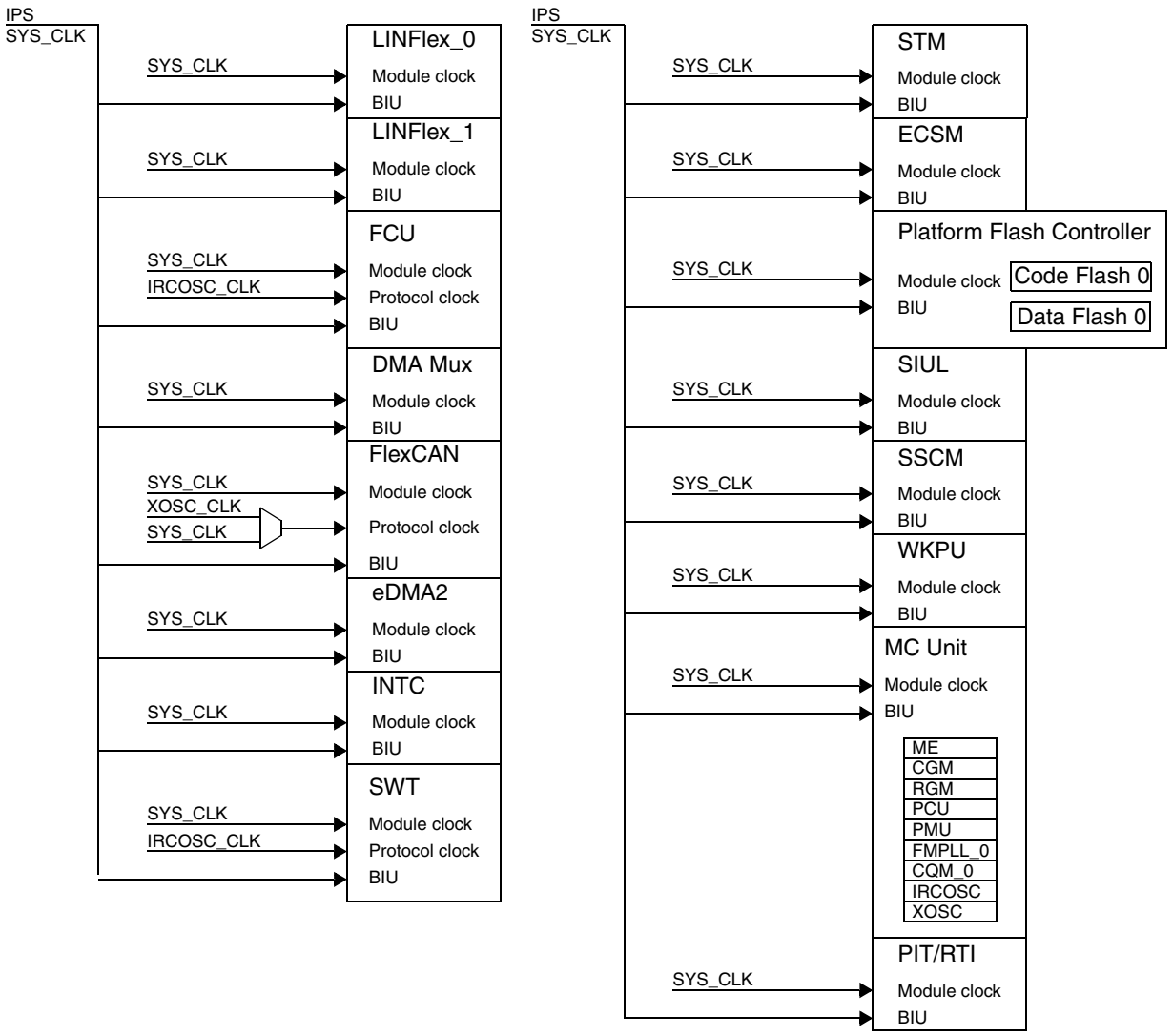


Figure 11. SPC560P40/34 system clock distribution Part B

4.2 Available clock domains

This section describes the various clock domains available on SPC560P40/34.

4.2.1 FMPLL input reference clock

The input reference clock for FMPLL_0 is always the external crystal oscillator clock (XOSC).

4.2.2 Clock selectors

System clock selector 0 for SYS_CLK

The system clock selector 0 selects the clock source for the system clock (SYS_CLK) from clock signals:

- Internal RC oscillator clock (IRC)
- Progressive output clock of FMPLL_0
- Directly from the oscillator clock (XOSC)

Its behavior is configured via software through ME_*x*_MC register of the ME module.

When the standard boot from internal flash is selected via the boot configuration pins, the clock source for the system clock (SYS_CLK) after reset (DRUN mode) is the internal RC oscillator (IRC).

4.2.3 Auxiliary Clock Selector 0

There is no Auxiliary Clock present on SPC560P40/34 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC0_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

4.2.4 Auxiliary Clock Selector 1

There is no Auxiliary Clock present on SPC560P40/34 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC1_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

4.2.5 Auxiliary Clock Selector 2

There is no Auxiliary Clock present on SPC560P40/34 device, but to maintain the software compatibility, corresponding register in MC_CGM (CGM_AC2_SC) has been implemented through which user can select any clock source from the given auxiliary clock sources. As there is no auxiliary clock, all the auxiliary clock sources have been tied to ‘0’.

4.2.6 Auxiliary clock dividers

As there is no auxiliary clock present on SPC560P40/34, there is no point in having the auxiliary clock dividers. To maintain the software compatibility, one divider corresponding to every auxiliary clock has been implemented. Corresponding registers have been implemented in MC_CGM which can be accessed by user but have no impact in device. These registers are CGM_AC0_DC0, CGM_AC1_DC0, and CGM_AC2_DC0

4.2.7 External clock divider

The output clock divider provides a nominal 50% duty cycle clock and allows the selected output clock source to be divided with these divide options:

- ÷ 1, ÷ 2, ÷ 4, ÷ 8

4.3 Alternate module clock domains

This section lists the different clock domains for each module. If not otherwise noted, all modules on the SPC560P40/34 device are clocked on the SYS_CLK.

4.3.1 FlexCAN clock domains

The FlexCAN modules have two distinct software controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be either the system clock (SYS_CLK) or a direct feed from the oscillator pin XOSC_CLK. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures very low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR. *Figure 262* shows the two clock domains in the FlexCAN modules.

Refer to *22, "FlexCAN* for more information on the FlexCAN modules.

4.3.2 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) is always the IRC generated by the internal RC oscillator.

4.3.3 Cross Triggering Unit (CTU) clock domains

The 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 MC_PLL_CLK. The logic in the Protocol Clock domain controls the CTU interface pins to the eTimer module and the ADC module.

4.3.4 Peripherals behind the IPS bus clock sync bridge

FlexPWM clock domain

The FlexPWM module has only one clock domain. The FlexPWM module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

eTimer_0 clock domain

The eTimer_0 module has only one clock domain. The eTimer_0 module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

ADC_0 clock domain

The ADC_0 module has only one clock domain. The ADC_0 module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

Safety Port clock domains

The Safety Port module has two distinct software-controlled clock domains. The first clock domain (Module Clock) is always supplied from the SP_PLL_CLK. The source for the second clock domain (Protocol Clock) can either be the SP_PLL_CLK or the XOSC_CLK.

The user must ensure that the frequency of the first clock domain (Module Clock) clocked from the MC_PLL_CLK is always the same or greater than the clock selected for the second clock domain (Protocol Clock).

4.4 Clock behavior in STOP and HALT mode

In this section the term “resume” is used to describe the transition from STOP and HALT mode back to a RUN mode.

The SPC560P40/34 supports the STOP and the HALT modes. These two modes allow to put the device into a power saving mode with the configuration options defined in the ME module.

The following constraints are applied on SPC560P40/34 to guarantee that in all modes of operation a resume from STOP or HALT mode is always possible without the need to reset:

- STOP and HALT mode:
  - SIUL clock is not gateable
  - SIUL filter for external interrupt capable pins is always clocked with IRC
  - 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:
  - IRC can NOT be switched off
  - The System Clock Selector 0 is switched to the IRC and therefore the SYS_CLK is feed by the IRC signal
  - Resume via external interrupt pin is always possible (if not masked)
- HALT mode:
  - 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) and IRC is not switched off

4.5 System clock functional safety

This section shows the SPC560P40/34 modules used to detect clock failures:

- The Clock Monitoring Unit (CMU_0) monitors the clock frequency of the FMPLL_0 and the XOSC signal against the IRC and provides clock out of range information about the monitored clock signals.
- FMPLL_0 provides a signal that indicates a loss of lock. Each loss of lock signal is sent to the CGM module.

Upon the detection of one of the above mentioned failures, the SPC560P40/34 device either asserts a reset, generates an interrupt, or sends the device into the SAFE state.

The reaction to each of the clock failures and system parameters (like active clocks and SYS_CLK clock source) that become active in SAFE state are under software control and can be configured in the ME module.

4.6 IRC 16 MHz internal RC oscillator (RC_CTL)

The IRC output frequency can be trimmed using RCTRIM bits. After a power-on reset, the IRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at RC_CTL[RCTRIM], and this field shows a value of zero. Therefore, be aware that the RC_CTL[RCTRIM] field does not reflect the current trim value until you have written to it. Pay particular attention to this feature when you initiate a read-modify-write operation on RC_CTL, because a RCTRIM value of zero may be unintentionally written back and this may alter the IRC frequency. In this case, you should calibrate the IRC using the CMU.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -32 to 31. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to device datasheet for average frequency variation of the trimming step.

Figure 12. RC Control register (RC_CTL)

Address: 0xC3FE_0060 (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	0	0	0	0	0	0	0	0	0	0	RCTRIM[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	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

Table 8. RC_CTL field descriptions

Field	Description
RCTRIM[5:0]	Main RC trimming bits

4.7 XOSC external crystal oscillator

The external crystal oscillator (XOSC) operates in the range of 4 MHz to 40 MHz. The XOSC digital interface contains the control and status registers accessible for the external crystal oscillator.

Main features are:

- Oscillator clock available interrupt
- Oscillator bypass mode

4.7.1 Functional description

The crystal oscillator circuit includes an internal oscillator driver and an external crystal circuitry. The XOSC provides an output clock to the PLL or it is used as a reference clock to specific modules depending on system needs.

The crystal oscillator can be controlled by the ME:

- Control by ME module. The OSCON bit of the ME_xxx_MCRs controls the powerdown of oscillator based on the current device mode while S_OSC of ME_GS register provides the oscillator clock available status.

After system reset, the oscillator is put to power down state and software has to switch on when required. Whenever the crystal oscillator is switched on from off state, OSCCNT counter starts and when it reaches the value EOCV[7:0] × 512, oscillator clock is made available to the system. Also an interrupt pending bit I_OSC of OSC_CTL register is set. An interrupt will be 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 reset this bit. In this bypass mode, the output clock has the same polarity as external clock applied on EXTAL pin and the oscillator status is forced to '1'. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 9 shows the truth table of different configurations of oscillator.

Table 9. Crystal oscillator truth table

ENABLE	BYP	XTALIN	EXTAL	CK_OSCM	XOSC Mode
0	0	No crystal, High Z	No crystal, High Z	0	Power down, IDDQ
x	1	x	Ext clock	EXTAL	Bypass, XOSC disabled
1	0	Crystal	Crystal	EXTAL	Normal, XOSC enabled

4.7.2 Register description

Table 10. OSC_CTL memory map

Offset from OSC_CTL_BASE (0xC3FE_0000)	Register	Access	Reset value	Location
0x0000	OSC_CTL—Oscillator control register	R/W	0x0080_0000	*on page 4-99*
0x0004–0x000F	Reserved			

Figure 13. Crystal Oscillator Control register (OSC_CTL)

Address: 0xC3FE_0000 (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	OSC BYP	0	0	0	0	0	0	0	EOCV[7:0]							
W																
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	M_ OSC	0	0	0	0	0	0	0	I_ OSC	0	0	0	0	0	0	0
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11. 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 reset this bit. 0: Oscillator output is used as root clock. 1: EXTAL is used as root clock.
EOCV[7:0]	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 external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV[7:0]*512, oscillator available interrupt request is generated. The reset value of this field depends on the device specification. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask 0: Crystal oscillator clock interrupt masked 1: Crystal oscillator clock interrupt enabled
I_OSC	Crystal oscillator clock interrupt This bit is set by hardware when OSCCNT counter reaches the count value EOCV[7:0]*512. It is cleared by software by writing 1. 0: No oscillator clock interrupt occurred 1: Oscillator clock interrupt pending

4.8 Frequency Modulated Phase Locked Loop (FMPLL)

4.8.1 Introduction

This section describes the features and functions of the FMPLL module implemented in SPC560P40/34.

4.8.2 Overview

The FMPLL enables the generation of high speed system clocks from a common 4–40 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor and output clock divider ratio are all software configurable.

The FMPLL block diagram is shown in *Figure 14*.

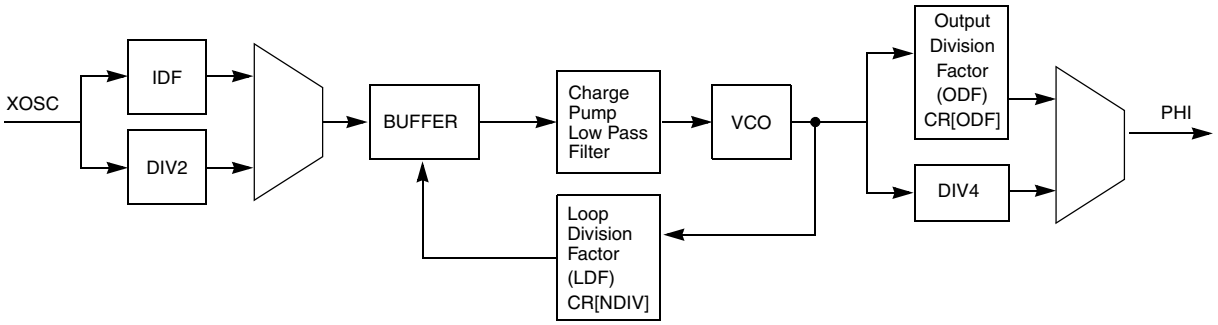


Figure 14. FMPLL block diagram

4.8.3 Features

The FMPLL has the following major features:

- Input clock frequency 4–40 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated PLL
  - 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
- Self-clocked mode (SCM) operation
- 4 available modes
  - Normal mode
  - Progressive clock switching
  - Normal Mode with SSCG
  - Powerdown mode

4.8.4 Memory map

Table 12 shows the memory map locations. Addresses are given as offsets of the module base address.

Table 12. FMPLL memory map

Offset from ME_CGM_BASE[1] FMPLL_0: 0xC3FE_00A0	Register	Access	Reset value	Location
0x0000	CR—Control Register	R/W	0x0080_0000	*on page 4-102*
0x0004	MR—Modulation register	R/W	0x0080_0000	*on page 4-104*
0x0004–0x000F	Reserved			

1. FMPLL_x are mapped through the ME_CGM Register Slot

4.8.5 Register description

The PLL operation is controlled by two registers. Those registers can only be written in supervisor mode.

Control Register (CR)

Figure 15. Control Register (CR)

Address: Base + 0x0000
FMPLL_0 = 0xC3FE_00A0

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	IDF[3:0]				ODF[1:0]		0	NDIV[6:0]						
W																
Reset	0	0	0	0	0	1	0	1	0	1	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	en_pll_sw	0	unlock_once	0	i_lock	s_lock	pll_fail_mask	pll_fail_flag	1
W												w1c			w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 13. CR field descriptions

Field	Description
IDF[3:0]	Input Division Factor The value of this field sets the PLL input division factor. 0000: Divide by 1 0001: Divide by 2 0010: Divide by 3 0011: Divide by 4 0100: Divide by 5 0101: Divide by 6 0110: Divide by 7 0111: Divide by 8 1000: Divide by 9 1001: Divide by 10 1010: Divide by 11 1011: Divide by 12 1100: Divide by 13 1101: Divide by 14 1110: Divide by 15 1111: Clock Inhibit
ODF[1:0]	Output Division Factor The value of this field sets the PLL output division factor. 00: Divide by 2 01: Divide by 4 10: Divide by 8 11: Divide by 16
NDIV[6:0]	Loop Division Factor The value of this field sets the PLL loop division factor. 0000000–0011111: Reserved 0100000: Divide by 32 0100001: Divide by 33 0100010: Divide by 34 ... 1011111: Divide by 95 1100000: Divide by 96 1100001–1111111: Reserved
en_pll_sw	This bit is used to enable progressive clock switching. After the PLL locks, the PLL output initially is divided by 8, and then progressively decreases until it reaches divide-by-1. Note: The PLL output should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished. 0: Progressive clock switching disabled 1: Progressive clock switching enabled
unlock_once	This bit is a sticky indication of PLL loss of lock condition. Unlock_once is set when the PLL loses lock. Whenever the PLL reacquires lock, unlock_once remains set. unlock_once is cleared after a POR event.

Table 13. CR field descriptions (continued)

Field	Description
i_lock	This bit is set by hardware whenever there is a lock/unlock event.It is cleared by software, writing 1.
s_lock	This bit indicates 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 software, writing 1.

Modulation Register (MR)

Figure 16. Modulation Register (MR)

Address: Base + 0x0004

Access: Supervisor read/write
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STRB_BYPASS	0	SPRD_SEL	MOD_PERIOD												
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	FM_EN	INC_STEP														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14. MR field descriptions

Field	Description
STRB_BYPASS	Strobe bypass The STRB_BYPASS signal bypasses the STRB signal used inside the PLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0: STRB latches the PLL modulation control bits. 1: STRB is bypassed. In this case, the 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 The SPRD_SEL bit selects the spread type in Frequency Modulation mode. 0: Center spread 1: Down spread

Table 14. MR field descriptions (continued)

Field	Description
MOD_PERIOD	Modulation period The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $modperiod = \frac{f_{ref}}{4 \times f_{mod}}$ where: *fref:* represents the frequency of the feedback divider *fmod*: represents the modulation frequency
FM_EN	Frequency modulation enable The FM_EN bit 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: $incstep = round\left(\frac{(2^{15} - 1) \times md \times MDF}{100 \times 5 \times MODPERIOD}\right)$ 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).

4.8.6 Functional description

Normal mode

In Normal mode, the PLL inputs are driven by the Control Register (CR). This means that when the PLL is locked, the PLL output clock (PHI) is derived from the reference clock (XOSC) through this relationship:

Equation 1

$$phi = \frac{xosc \cdot ldf}{idf \cdot odf}$$

where the value of *idf* (Input Division Factor), *ldf* (Loop Division Factor), and *odf* (Output Division Factor) are set in the CR as shown in *Table 13*. *idf* and *odf* are specified in the IDF and ODF bitfields, respectively; *ldf* is specified in the NDIV bitfield.

Progressive clock switching

Progressive clock switching allows to switch system clock to PLL output clock stepping through different division factors. This means that the current consumption gradually increases and, in turn, voltage regulator response is improved.

This feature can be enabled by programming bit en_pll_sw in the CR. Then, when the PLL is selected as the system clock, the output clock progressively increases its frequency as shown in *Table 15*.

Table 15. Progressive clock switching on pll_select rising edge

Number of PLL output clock cycles	ck_pll_div frequency (PLL output frequency)
8	(ck_pll_out frequency) ÷ 8
16	(ck_pll_out frequency) ÷ 4
32	(ck_pll_out frequency) ÷ 2
onward	(ck_pll_out frequency)

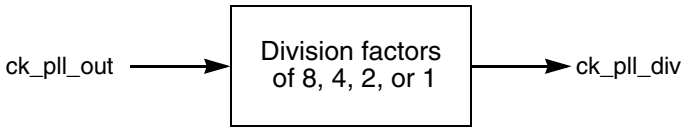


Figure 17. Progressive clock switching scheme

Normal Mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. When frequency modulation is enabled, however, 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.

Frequency modulation is activated as follows:

1. Configure the FM modulation characteristics: MOD_PERIOD, INC_STEP.
2. Enable the FM modulation by programming bit SSCG_EN of the MR to '1'. FM modulated mode can be enabled only when PLL is in lock state.

There are two ways to latch these values inside the FMPLL, depending on the value of bit STRB_BYPASS in the MR.

If STRB_BYPASS is low, the modulation parameters are latched in the PLL only when the strobe signal goes high. The strobe 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 strobe 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

Equation 2

$$\text{ModulationDepth} = \left(\frac{100 \times 5 \times \text{INCSTEPxMODPERIOD}}{(2^{15} - 1) \times \text{MDF}}\right)$$

Note: The user must ensure that the product of INCSTEP and MODPERIOD is less than ($2^{15} - 1$).

The following values show the input setting for one possible configuration of the PLL:

- PLL input frequency: 4 MHz
- Loop divider (LDF): 64
- Input divider (IDF): 1
- VCO frequency = 4 MHz × 64 = 256 MHz
- PLL output frequency = 256 MHz/ODF
- Spread: Center spread (SPREAD_CONTROL = 0)
- Modulation frequency = 24 kHz
- Modulation depth = ±2.0% (4% pk-pk)

Using the formulae for MODPERIOD and INCSTEP:

Equation 3

MODPERIOD = Round [(4e06) / (4 × 24e03)] = Round [41.66] = 42

Equation 4

INCSTEP = Round [$((2^{15} - 1) \times 2 \times 64) / (100 \times 5 \times 42)$] = Round [199.722] = 200

Equation 5

MODPERIOD × INCSTEP = 42 × 200 = 8400 (which is less than 2^{15})

Equation 6

md(quantized)% = ((42*200*100*5) / ((2^15-1)*64) = 2.00278% (peak)

Equation 7

Error in modulation depth = 2.00278 - 2.0 = 0.00278%

If we choose MODPERIOD = 41,

Equation 8

INCSTEP = Round [$((2^{15} - 1) \times 2 \times 64) / (100 \times 5 \times 41)$] = Round [204.878] = 205

Equation 9

MODPERIOD × INCSTEP = 41 × 205 = 8405 (which is less than 2^{15})

Equation 10

md(quantized)% = $((41 \times 205 \times 100 \times 5) / ((2^{15} - 1) \times 64)$ = 2.00397% (peak)

Equation 11

Error in modulation depth = 2.00397 – 2.0 = 0.00397%

The above calculations show that the quantization error in the modulation depth depends on the flooring and rounding of MODPERIOD and INCSTEP. For this reason, the MODPERIOD and INCSTEP should be judiciously rounded/floored to minimize the quantization error in the modulation depth.

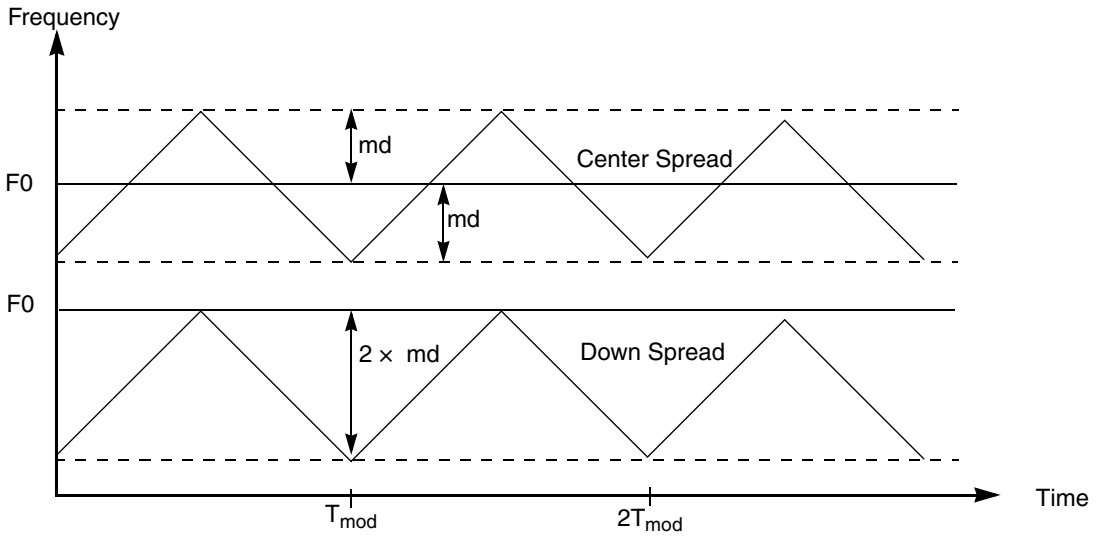


Figure 18. Frequency modulation depth spreads

Powerdown mode

To reduce consumption, the FMPLL can be switched off when not required by programming the registers ME_x_MC on the ME module.

4.8.7 Recommendations

To avoid any unpredictable behavior of the PLL clock, it is recommended to follow these guidelines:

- The PLL VCO frequency should reside in the range 256 MHz to 512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- The user must change the multiplication, division factors only when the PLL output clock is not selected as system clock. MOD_PERIOD, INC_STEP, 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.

4.9 Clock Monitor Unit (CMU)

4.9.1 Overview

The Clock Monitor Unit (CMU) serves three purposes:

- PLL clock monitoring: detects if PLL leaves an upper or lower frequency boundary
- XOSC clock monitoring: monitor the XOSC clock, which must be greater than the IRCOSC clock divided by a division factor given by CMU_CSR[RCDIV]
- Frequency meter: measure the frequency of the IRCOSC clock versus the reference XOSC clock frequency

When mismatch occurs in the CMU either with the PLL monitor or the XOSC monitor, the CMU notifies the RGM, ME and the FCU (Fault Collection Unit) modules. The default behavior is such that a reset occurs and a status bit is set in the RGM. The user also has the option to change the behavior of the action by disabling the reset and selecting an alternate action. The alternate action can be either entering safe mode or generating an interrupt.

Table 16. CMU module summary

Module	Monitored clocks
CMU_0	– XOSC integrity supervisor – FMPLL_0 integrity supervisor – IRCOSC frequency meter

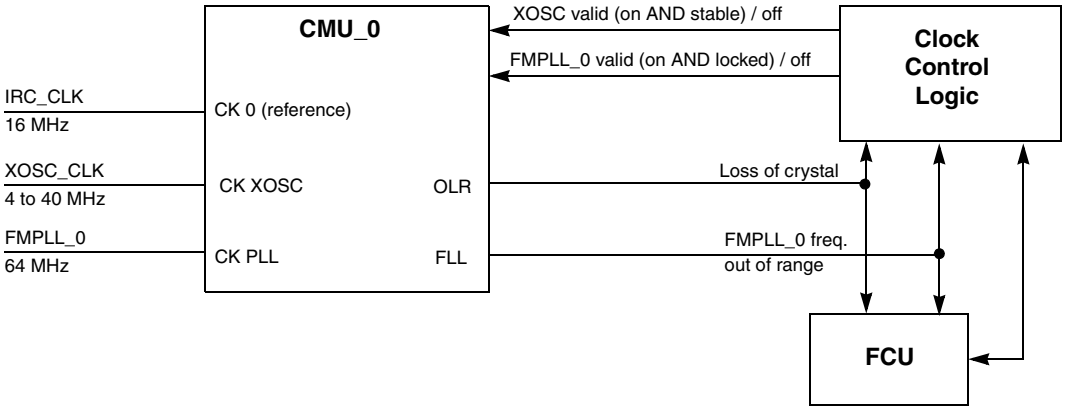


Figure 19. SPC560P40/34CMU

4.9.2 Main features

- RC oscillator frequency measurement
- External oscillator clock monitoring with respect to CK_IRC/*n* clock
- PLL clock frequency monitoring with respect to CK_IRC/4 clock
- Event generation for various failures detected inside monitoring unit

4.9.3 Functional description

The clock and frequency names referenced in this block are defined as follows:

- CK_XOSC: clock coming from the external crystal oscillator
- CK_IRC: clock coming from the low frequency internal RC oscillator
- CK_PLL: clock coming from the PLL
- f_{XOSC}: frequency of external crystal oscillator clock
- f_{RC}: frequency of low frequency internal RC oscillator
- f_{PLL}: frequency of FMPLL clock

Crystal clock monitor

If f_{XOSC} is smaller than f_{RC} divided by 2^{RCDIV} bits of CMU_0_CSR and the CK_XOSC is 'ON' and stable as signaled by the ME, then:

- An event pending bit OLRI in CMU_0_ISR is set
- A failure event OLR is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

PLL clock monitor

The PLL clock CK_PLL frequency can be monitored by programming bit CME_0 of the CMU_0_CSR to '1'. The CK_PLL monitor starts as soon as bit CME_0 is set. This monitor can be disabled at any time by writing bit CME_0 to '0'.

If the CK_PLL frequency (f_{PLL}) is greater than a reference value determined by bits HFREF[11:0] of the CMU_HFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME then:

- An event pending bit FHHI_0 in the CMU_0_ISR is set.
- A failure event FHH is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

If f_{PLL} is less than a reference clock frequency ($f_{RC}/4$) and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then:

- An event pending bit FLCI_0 in the CMU_0_ISR is set.
- A failure event FLC is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

If f_{PLL} is less than a reference value determined by bits LFREF[11:0] of the CMU_LFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then:

- An event pending bit FLLI_0 in the CMU_0_ISR is set.
- A failure event FLL is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

Note: It is possible for either the XOSC or PLL monitors to produce a false event when the XOSC or PLL frequency is too close to RC/2^{RCDIV} frequency due to an accuracy limitation of the compare circuitry.

System clock monitor

The system clock is monitored by CMU_1. The F_{SYS_CLK} frequency can be monitored by programming CMU_1_CSR[CME] = 1. SYS_CLK monitoring starts as soon as CMU_1_CSR[CME] = 1. This monitor can be disabled at any time by writing CME bit to 0.

If F_{SYS_CLK} is greater than a reference value determined by the CMU_1_HFREFR_A[HFREF_A] bits and the system clock is enabled, then:

- CMU_1_ISR[FHHI] is set
- A failure event is signaled to the MC_RGM and FCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

If F_{SYS_CLK} is less than a reference clock frequency (F_{IRCOSC_CLK}÷4) and the system clock is enabled, then:

- CMU_1_ISR[FLCI] is set
- A failure event FLC is signaled to the MC_RGM and Fault Collection Unit, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

If F_{SYS_CLK} is less than a reference value determined by the CMU_1_LFREFR_A[LFREF_A] bits and the system clock is enabled, then:

- CMU_1_ISR[FLLI] is set
- A failure event is signaled to the MC_RGM and FCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

Note: The system clock monitor may produce a false event when F_{SYS_CLK} is less than $2 \times F_{IRCOSC\_CLK}/2^{CMU\_1\_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

Frequency meter

The frequency meter calibrates the internal RC oscillator (CK_IRC) using a known frequency.

Note: This value can then be stored into the flash so that application software can reuse it later on.

The reference clock will be always the XOSC. A simple frequency meter returns a draft value of CK_IRC. The measure starts when bit SFM (Start Frequency Measure) in the CMU_CSR is set to '1'. The measurement duration is given by the CMU_MDR in numbers of IRC clock cycles with a width of 20 bits. Bit SFM is reset to '0' by hardware once the frequency measurement is done and the count is loaded in the CMU_FDR. The frequency f_{RC} can be derived from the value loaded in the CMU_FDR as follows:

Equation 12

$$f_{RC} = (f_{OSC} \times MD) / n$$

where *n* is the value in the CMU_FDR and MD is the value in the CMU_MDR.

4.9.4 Memory map and register description

Table 17 shows the memory map of the CMU.

Table 17. CMU memory map

Offset from CMU_BASE (0xC3FE_0100)	Register	Access	Reset value	Location
0x0000	Control Status Register (CMU_0_CSR)	R/W	0x0000_0006	*on page 4-112*
0x0004	Frequency Display Register (CMU_0_FDISP)	R	0x0000_0000	*on page 4-113*
0x0008	High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_A)	R/W	0x0000_0FFF	*on page 4-113*

Table 17. CMU memory map (continued)

Offset from CMU_BASE (0xC3FE_0100)	Register	Access	Reset value	Location
0x000C	Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)	R/W	0x0000_0000	*on page 4-114*
0x0010	Interrupt Status Register (CMU_0_ISR)	R/W	0x0000_0000	*on page 4-114*
0x0014	Reserved			
0x0018	Measurement Duration Register (CMU_0_MDR)	R/W	0x0000_0000	*on page 4-115*
0x001C–0x3FFF	Reserved			

Control Status Register (CMU_0_CSR)

Figure 20. Control Status Register (CMU_0_CSR)

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	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	0	0	0	0	0	0	0	RCDIV[1:0]		CME_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 18. CMU_0_CSR field descriptions

Field	Description
SFM	Start frequency measure The software can only set this bit to start a clock frequency measure. It is reset by hardware when the measure is ready in the CMU_FDR. 0: Frequency measurement completed or not yet started 1: Frequency measurement not completed
RCDIV[1:0]	RC clock division factor These bits specify the RC clock division factor. The output clock is CK_IRC 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
CME_0	FMPLL_0 clock monitor enable 0: FMPLL_0 monitor disabled 1: FMPLL_0 monitor enabled

Frequency Display Register (CMU_0_FDR)

Figure 21. Frequency Display Register (CMU_0_FDR)

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	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

Table 19. CMU_0_FDR field descriptions

Field	Description
FD[19:0]	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 CMU_FDR.

High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_A)

Figure 22. High Frequency Reference register FMPLL_0 (CMU_0_HFREFR_A)

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	HFREF[11:0]											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 20. CMU_0_HFREFR_A field descriptions

Field	Description
HFREF_A	High Frequency reference value These bits determine the high reference value for the FMPLL_0 clock. The reference value is given by: (HFREF_A[11:0]/16) × (f_{RC}/4).

Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)

Figure 23. Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)

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	0	0	0	0	LFREF[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 21. CMU_0_LFREFR_A fields descriptions

Field	Description
LFREF_A	Low Frequency reference value These bits determine the low reference value for the FMPLL_0. The reference value is given by: (LFREF_A[11:0]/16) * (f_{RC}/4).

Interrupt Status Register (CMU_0_ISR)

Figure 24. Interrupt Status Register (CMU_0_ISR)

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	FLCI_0	FHHI_0	FLLI_0	OLRI
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22. CMU_0_ISR field descriptions

Field	Description
FLCI_0	FMPLL_0 Clock frequency less than reference clock interrupt This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than reference clock frequency (f_{RC}/4) value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLC event 1: FLC event pending

Table 22. CMU_0_ISR field descriptions (continued)

Field	Description
FHHI_0	FMPLL_0 Clock frequency higher than high reference interrupt This bit is set by hardware when CK_FMPLL_ 0 frequency becomes higher than HFREF_A value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FHH event 1: FHH event pending
FLLI_0	FMPLL_0 Clock frequency less than low reference event This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than LFREF_A value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLL event 1: FLL event pending
OLRI	Oscillator frequency less than RC frequency event This bit is set by hardware when the frequency of CK_XOSC is less than CK_IRC/2^{RCDIV} frequency and CK_XOSC is 'ON' and stable as signaled by the ME. It can be cleared by software by writing 1. 0: No OLR event 1: OLR event pending

Measurement Duration Register (CMU_0_MDR)

Figure 25. Measurement Duration Register (CMU_0_MDR)

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	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

Table 23. CMU_0_MDR field descriptions

Field	Description
MD[19:0]	Measurement duration bits This register displays the measured duration in term of IRC clock cycles. This value is loaded in the frequency meter downcounter. When SFM bit is set to '1', downcounter starts counting.

5 Clock Generation Module (MC_CGM)

5.1 Overview

The clock generation module (MC_CGM) generates reference clocks for all the SoC blocks. The MC_CGM selects one of the system clock sources to supply the system clock. The MC_ME controls the system clock selection (see the MC_ME chapter for more details). Peripheral clock selection is controlled by MC_CGM control registers. A set of MC_CGM registers controls the clock dividers which are used for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources which have addressable memory spaces are accessed through the MC_CGM memory space. The MC_CGM also selects and generates an output clock.

Figure 26 depicts the MC_CGM Block Diagram.

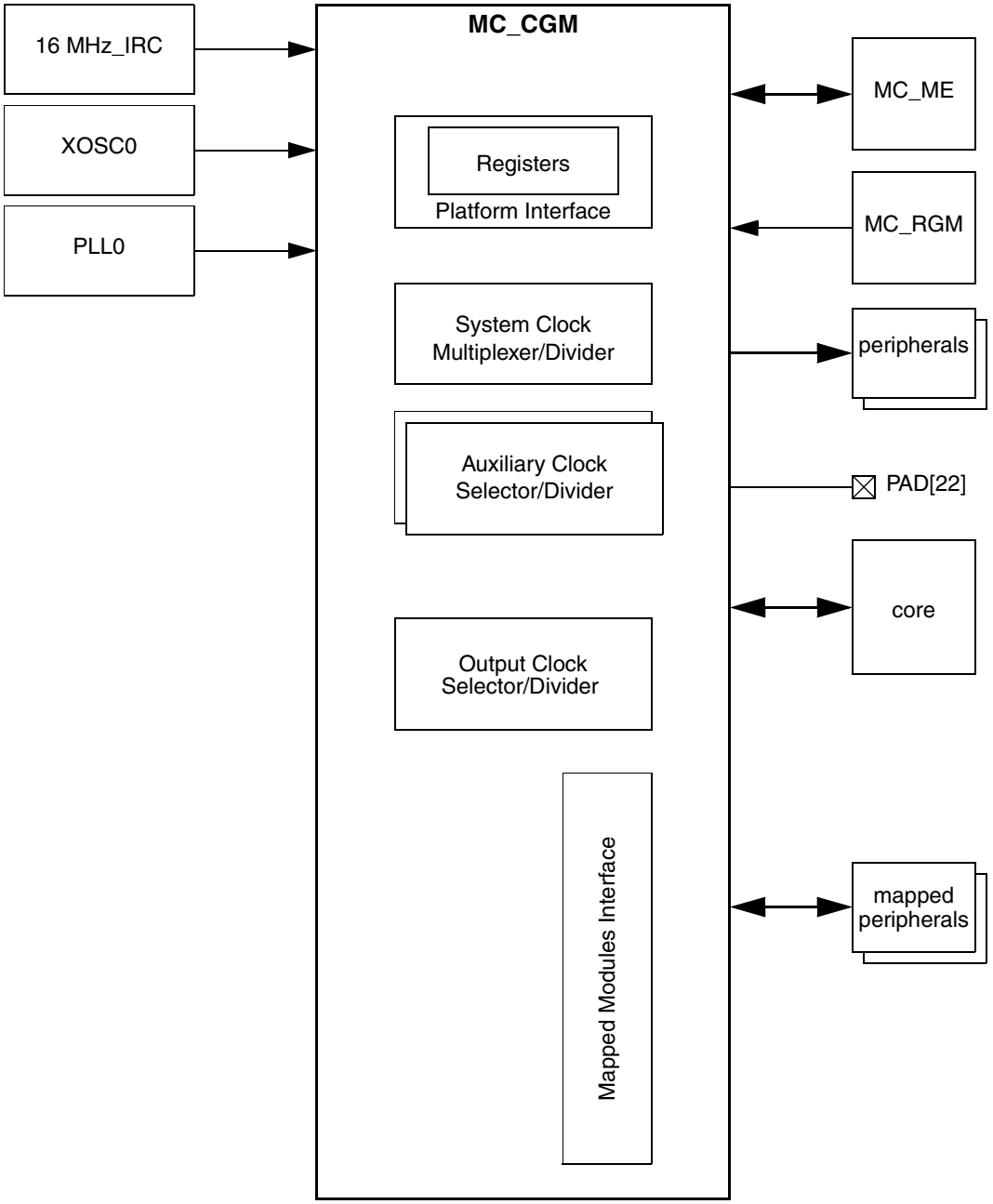


Figure 26. MC_CGM Block Diagram

5.2 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
- contains a set of registers to control peripheral clock selection
- supports multiple clock sources and maps their address spaces to its memory map
- generates an output clock
- guarantees glitch-less clock transitions when changing the system clock selection
- supports 8, 16 and 32-bit wide read/write accesses

5.3 External Signal Description

The MC_CGM delivers an output clock to the PAD[22] pin for off-chip use and/or observation.

5.4 Memory Map and Register Definition

Table 24. 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 5-124*
0xC3FE_0374	CGM_OCDS_SC	Output Clock Division Select	byte	read	read/write	read/write	*on page 5-124*
0xC3FE_0378	CGM_SC_SS	System Clock Select Status	byte	read	read	read	*on page 5-125*
0xC3FE_037C	CGM_SC_DC0	System Clock Divider Configuration 0	byte	read	read/write	read/write	*on page 5-126*
0xC3FE_0380	CGM_AC0_SC	Aux Clock 0 Select Control	word	read	read/write	read/write	*on page 5-127*
0xC3FE_0384	CGM_AC0_DC0	Aux Clock 0 Divider Configuration 0	byte	read	read/write	read/write	*on page 5-128*
0xC3FE_0388	CGM_AC1_SC	Aux Clock 1 Select Control	word	read	read/write	read/write	*on page 5-128*
0xC3FE_038C	CGM_AC1_DC0	Aux Clock 1 Divider Configuration 0	byte	read	read/write	read/write	*on page 5-129*
0xC3FE_0390	CGM_AC2_SC	Aux Clock 2 Select Control	word	read	read/write	read/write	*on page 5-130*
0xC3FE_0394	CGM_AC2_DC0	Aux Clock 2 Divider Configuration 0	byte	read	read/write	read/write	*on page 5-131*

Note: Any access to unused registers as well as write accesses to read-only registers will not change register content, and cause a transfer error.

Table 25. 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																
0xC3FE_0020 … 0xC3FE_003C	reserved																
0xC3FE_0040 … 0xC3FE_005C	reserved																
0xC3FE_0060 … 0xC3FE_007C	IRCOSC registers																
0xC3FE_0080 … 0xC3FE_009C	reserved																
0xC3FE_00A0 … 0xC3FE_00BC	PLL0 registers																
0xC3FE_00C0 … 0xC3FE_00DC	reserved																
0xC3FE_00E0 … 0xC3FE_00FC	reserved																

Table 25. 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_0100 … 0xC3FE_011C	CMU0 registers																
0xC3FE_0120 … 0xC3FE_013C	reserved																
0xC3FE_0140 … 0xC3FE_015C	reserved																
0xC3FE_0160 … 0xC3FE_017C	reserved																
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																

Table 25. 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_0220 … 0xC3FE_023C	reserved																
0xC3FE_0240 … 0xC3FE_025C	reserved																
0xC3FE_0260 … 0xC3FD_C27C	reserved																
0xC3FE_0280 … 0xC3FE_029C	reserved																
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																

Table 25. 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_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																
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	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_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																

Table 25. 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_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																
0xC3FE_0398 … 0xC3FE_3FFC	reserved																	

5.5 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.

5.5.1 Output Clock Enable Register (CGM_OC_EN)

Figure 27. Output Clock Enable Register (CGM_OC_EN)

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

This register is used to enable and disable the output clock.

Table 26. Output Clock Enable Register (CGM_OC_EN) Field Descriptions

Field	Description
EN	**Output Clock Enable control** 0 Output Clock is disabled 1 Output Clock is enabled

5.5.2 Output Clock Division Select Register (CGM_OCDS_SC)

Figure 28. Output Clock Division Select Register (CGM_OCDS_SC)

Address 0xC3FE_0374 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	SELDIV		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

This register is used to select the current output clock source and by which factor it is divided before being delivered at the output clock.

Table 27. Output Clock Division Select Register (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 MHz crystal osc. 0010 system PLL 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

5.5.3 System Clock Select Status Register (CGM_SC_SS)

Figure 29. System Clock Select Status Register (CGM_SC_SS)

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

This register provides the current system clock source selection.

Table 28. System Clock Select Status Register (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 4 MHz crystal osc. 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

5.5.4 System Clock Divider Configuration Register (CGM_SC_DC0)

Figure 30. System Clock Divider Configuration Register (CGM_SC_DC0)

Address 0xC3FE_037C Access: User read, Supervisor read/write, Test read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
Reset	1	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
	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

This register controls the system clock divider.

Table 29. System Clock Divider Configuration Register (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 divided system clock 0 will have 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 divided system clock 0 remains disabled.

5.5.5 Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

Figure 31. Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)

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

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 0 divider 0: (unused)

See *Figure 40* for details.

Table 30. Auxiliary Clock 0 Select Control Register (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 (no clock) 0001 reserved 0010 (no clock) 0011 reserved 0100 (no clock) 0101 (no clock) 0110 reserved 0111 reserved 1000 (no clock) 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

5.5.6 Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)

Figure 32. Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)

Address 0xC3FE_0384 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	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
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	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

This register controls the auxiliary clock 0 divider.

Table 31. Auxiliary Clock 0 Divider Configuration Register (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 (unused) will have 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 (unused) remains disabled.

5.5.7 Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

Figure 33. Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)

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	0	0	0	0	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

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 1 divider 0: (unused)

Table 32. Auxiliary Clock 1 Select Control Register (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 (no clock) 0001 reserved 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

5.5.8 Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)

Figure 34. Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)

Address 0xC3FE_038C 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	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
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	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

This register controls the auxiliary clock 1 divider.

Table 33. Auxiliary Clock 1 Divider Configuration Register (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 (unused) will have 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 (unused) remains disabled.

5.5.9 Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

Figure 35. Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)

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	0	0	0	0	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

This register is used to select the current clock source for the following clocks:

- undivided: (unused)
- divided by auxiliary clock 2 divider 0: (unused)

See *Figure 38* for details.

Table 34. Auxiliary Clock 2 Select Control Register (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 (no clock) 0001 reserved 0010 (no clock) 0011 reserved 0100 (no clock) 0101 (no clock) 0110 reserved 0111 reserved 1000 (no clock) 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

5.5.10 Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

Figure 36. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

Address 0xC3FE_0394 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	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
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	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

This register controls the auxiliary clock 2 divider.

Table 35. Auxiliary Clock 2 Divider Configuration Register (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 (unused) will have 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 (unused) remains disabled.

5.6 Functional Description

5.7 System Clock Generation

Figure 37 shows the block diagram of the system clock generation logic. The MC_ME provides the system clock select and switch mask (see MC_ME chapter for more details), and the MC_RGM provides the safe clock request (see MC_RGM chapter 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.

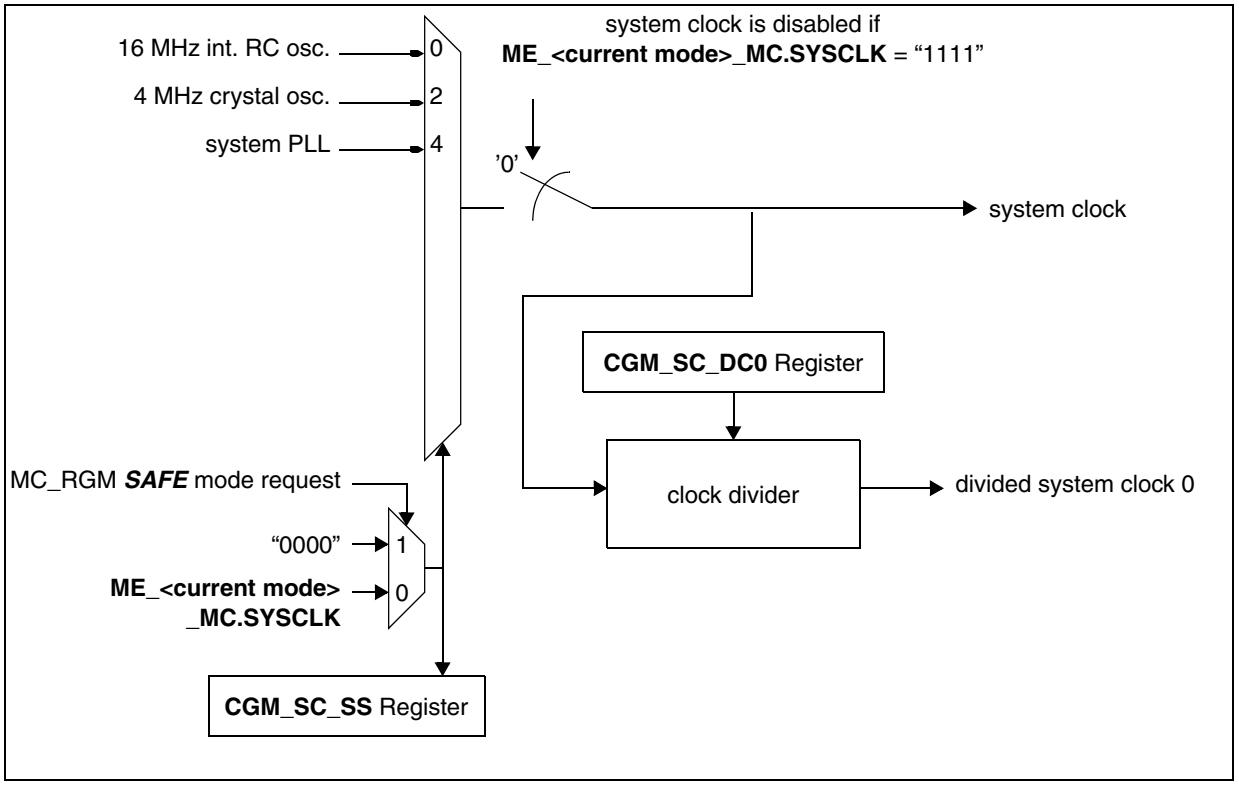


Figure 37. MC_CGM System Clock Generation Overview

5.7.1 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

5.7.2 System Clock Disable

During the TEST mode, the system clock can be disabled by the MC_ME.

5.7.3 System Clock Dividers

The MC_CGM generates the divided system clock 0 - controlled by the CGM_SC_DC0 register.

5.8 Auxiliary Clock Generation

Figure 38 shows the block diagram of the auxiliary clock generation logic. See *Section 5.5.5, “Auxiliary Clock 0 Select Control Register (CGM_AC0_SC)*, *Section 5.5.7, “Auxiliary Clock 1 Select Control Register (CGM_AC1_SC)*, and *Section 5.5.9, “Auxiliary Clock 2 Select Control Register (CGM_AC2_SC)* for auxiliary clock selection control.

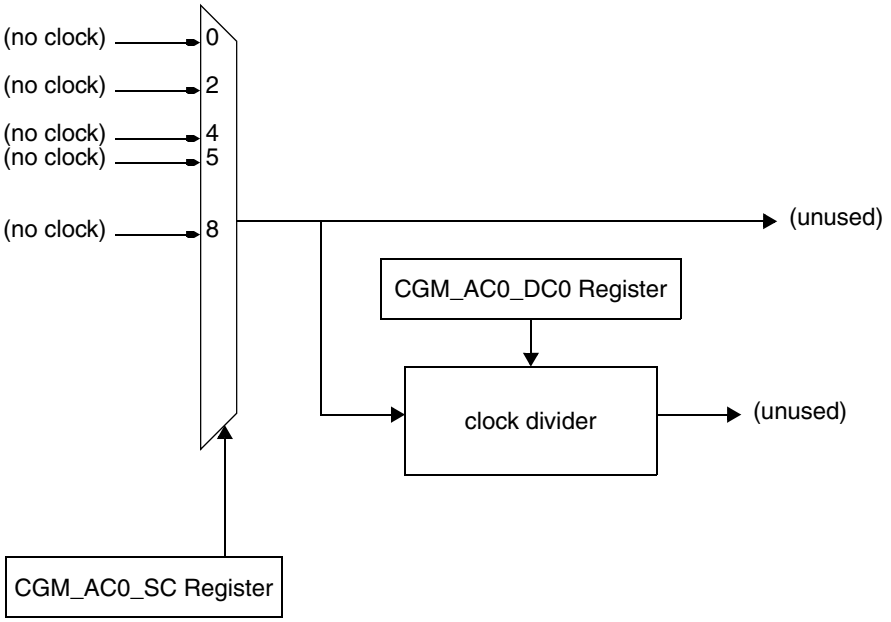


Figure 38. MC_CGM Auxiliary Clock 0 Generation Overview

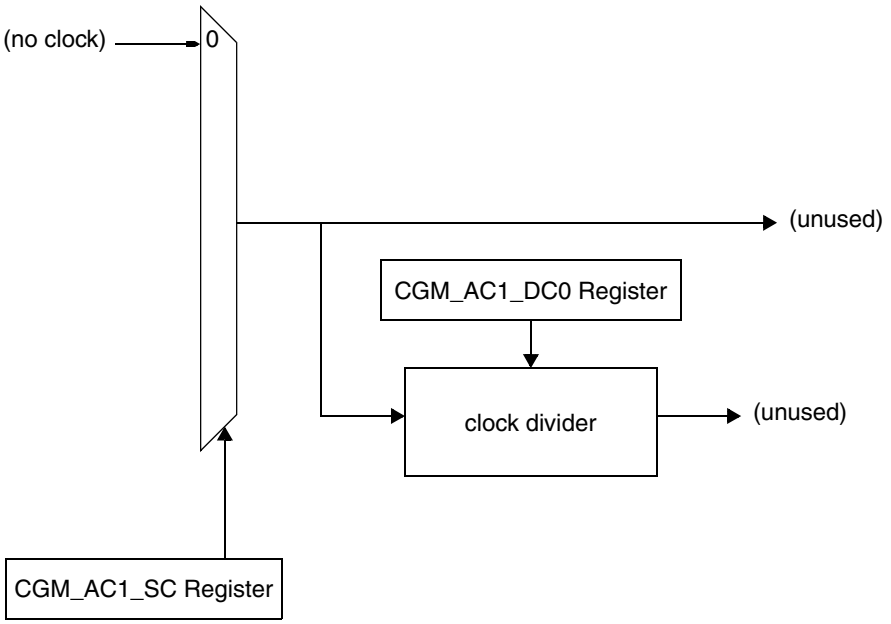


Figure 39. MC_CGM Auxiliary Clock 1 Generation Overview

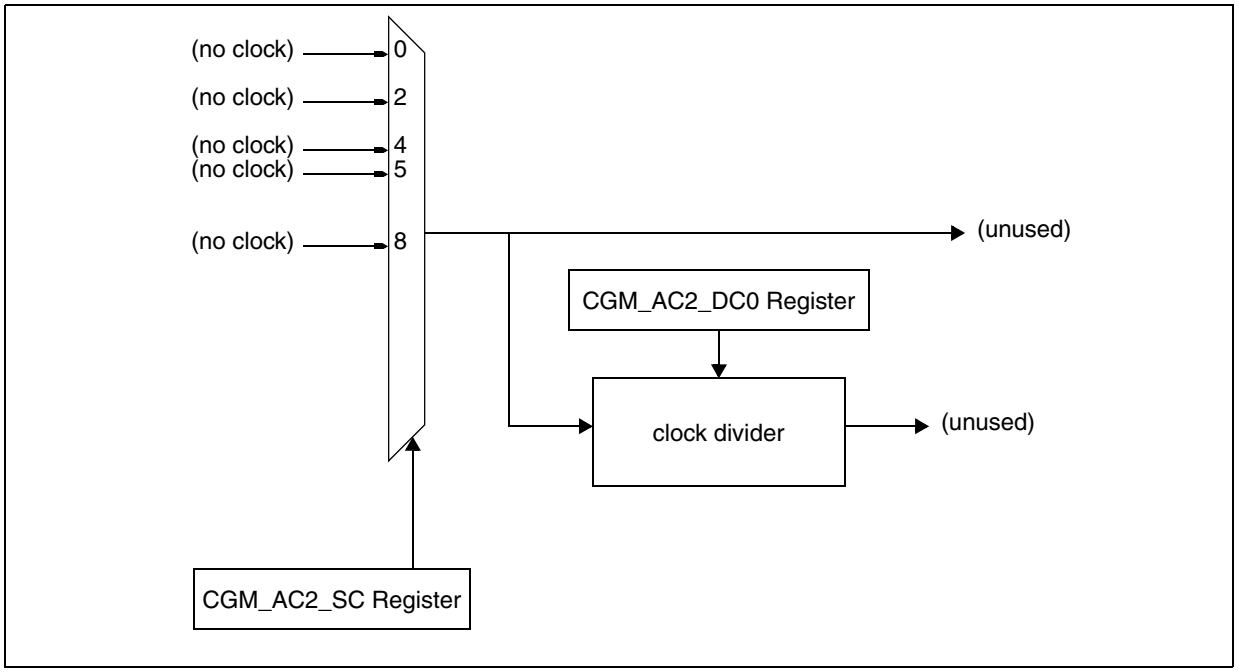


Figure 40. MC_CGM Auxiliary Clock 2 Generation Overview

5.8.1 Auxiliary Clock Source Selection

During normal operation, the auxiliary clock selection is done via the CGM_AC0…2_SC registers. If software selects an 'unavailable' source, the old selection remains, and the register content does not change.

5.8.2 Auxiliary Clock Dividers

The MC_CGM generates the following derived clocks:

- (unused) - controlled by the CGM_AC0_DC0 register
- (unused) - controlled by the CGM_AC1_DC0 register
- (unused) - controlled by the CGM_AC2_DC0 register

5.9 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 5.5.4, "System Clock Divider Configuration Register (CGM_SC_DC0)*
- *Section 5.5.6, "Auxiliary Clock 0 Divider Configuration Register (CGM_AC0_DC0)*
- *Section 5.5.8, "Auxiliary Clock 1 Divider Configuration Register (CGM_AC1_DC0)*
- *Section 5.5.10, "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 DIVn field is ignored.

5.10 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.

5.11 Output Clock Division Selection

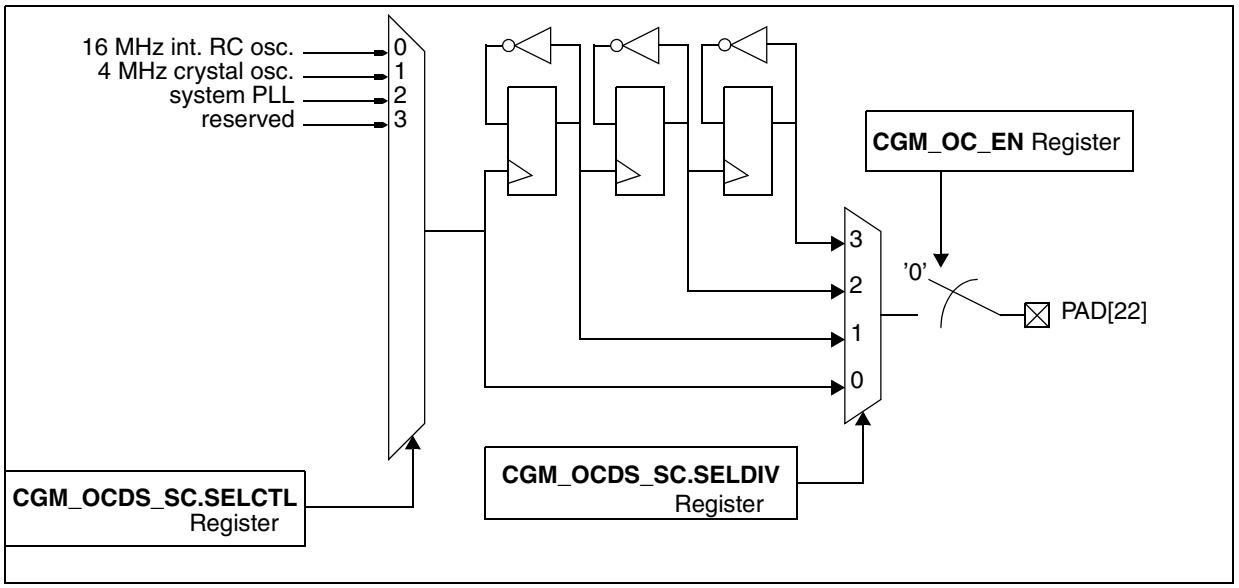


Figure 41. MC_CGM Output Clock Multiplexer and PAD[22] Generation

The MC_CGM provides the following output signals for the output clock generation:

- PAD[22] (see *Figure 41*). 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 5.5.1, "Output Clock Enable Register (CGM_OC_EN)"*) which contains the output clock enable/disable control bit.

6 Mode Entry Module (MC_ME)

6.1 Introduction

6.1.1 Overview

The MC_ME controls the SoC mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

Figure 42 depicts the MC_ME Block Diagram.

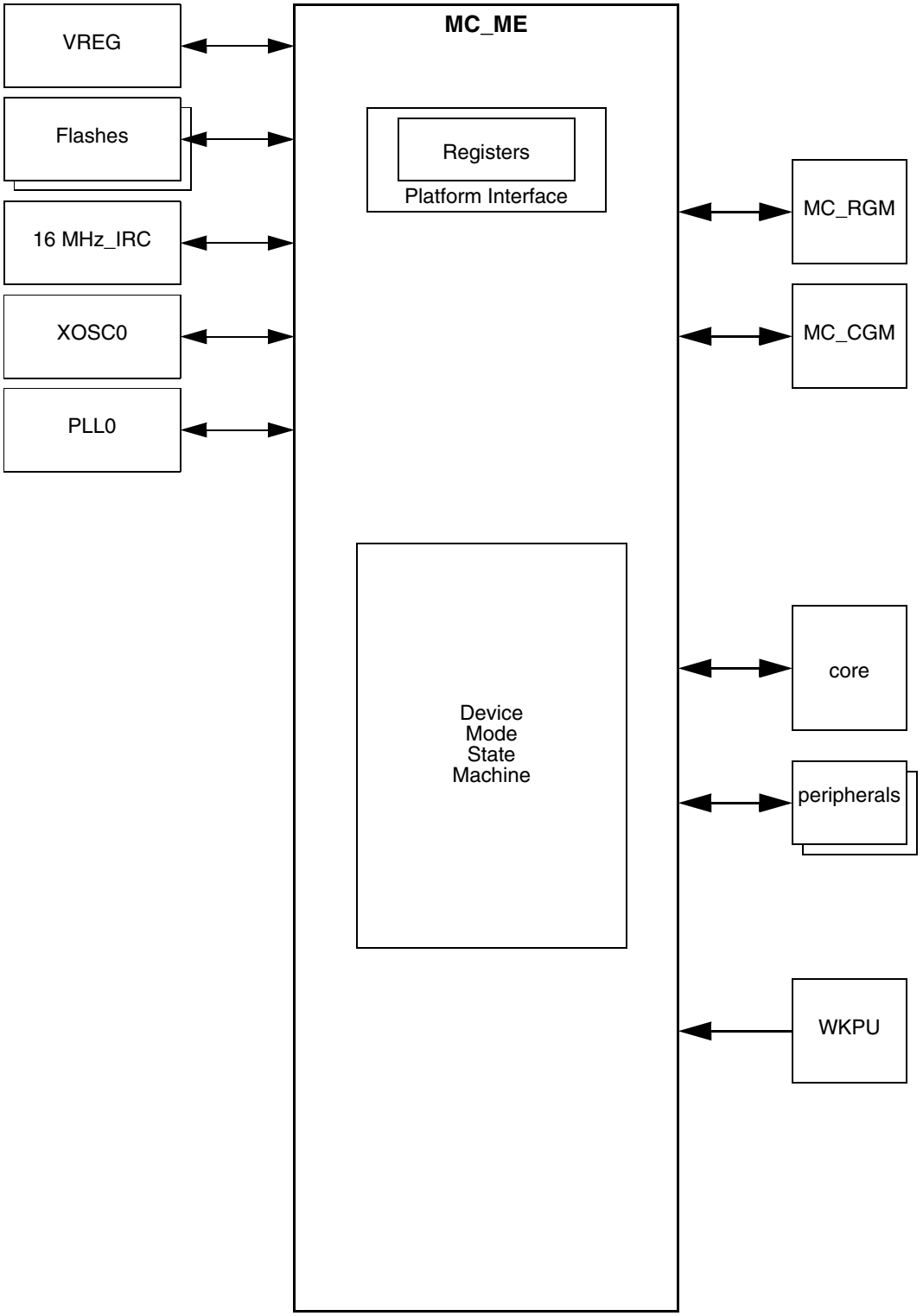


Figure 42. MC_ME Block Diagram

6.1.2 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 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_PCTL0…143 registers
- capture of current peripheral clock gated/enabled status

6.1.3 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 36 describes the MC_ME modes.

Table 36. 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 which 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, software request from DRUN, TEST, and RUN0…3	system reset assertion, DRUN via software
TEST	This is a chip-wide service mode which is intended to provide a control environment for device software teting.	software request from DRUN	system reset assertion, DRUN via software

Table 36. 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 to enable different clock & 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, 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

6.2 External Signal Description

The MC_ME has no connections to any external pins.

6.3 Memory Map and Register Definition

The MC_ME contains registers for:

- mode selection and status reporting
- mode configuration
- mode transition interrupts status and mask control
- scalable number of peripheral sub-mode selection and status reporting

6.3.1 Memory Map

Table 37. 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 6-147*
0xC3FD_C004	ME_MCTL	Mode Control	word	read	read/write	read/write	*on page 6-149*
0xC3FD_C008	ME_ME	Mode Enable	word	read	read/write	read/write	*on page 6-150*

Table 37. MC_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C00C	ME_IS	Interrupt Status	word	read	read/write	read/write	on page 6-152
0xC3FD_C010	ME_IM	Interrupt Mask	word	read	read/write	read/write	on page 6-153
0xC3FD_C014	ME_IMTS	Invalid Mode Transition Status	word	read	read/write	read/write	on page 6-154
0xC3FD_C018	ME_DMTS	Debug Mode Transition Status	word	read	read	read	on page 6-155
0xC3FD_C020	ME_RESET_MC	RESET Mode Configuration	word	read	read	read	on page 6-158
0xC3FD_C024	ME_TEST_MC	TEST Mode Configuration	word	read	read/write	read/write	on page 6-158
0xC3FD_C028	ME_SAFE_MC	SAFE Mode Configuration	word	read	read/write	read/write	on page 6-159
0xC3FD_C02C	ME_DRUN_MC	DRUN Mode Configuration	word	read	read/write	read/write	on page 6-160
0xC3FD_C030	ME_RUN0_MC	RUN0 Mode Configuration	word	read	read/write	read/write	on page 6-161
0xC3FD_C034	ME_RUN1_MC	RUN1 Mode Configuration	word	read	read/write	read/write	on page 6-161
0xC3FD_C038	ME_RUN2_MC	RUN2 Mode Configuration	word	read	read/write	read/write	on page 6-161
0xC3FD_C03C	ME_RUN3_MC	RUN3 Mode Configuration	word	read	read/write	read/write	on page 6-161
0xC3FD_C040	ME_HALT0_MC	HALT0 Mode Configuration	word	read	read/write	read/write	on page 6-161
0xC3FD_C048	ME_STOP0_MC	STOP0 Mode Configuration	word	read	read/write	read/write	on page 6-162
0xC3FD_C060	ME_PS0	Peripheral Status 0	word	read	read	read	on page 6-164
0xC3FD_C064	ME_PS1	Peripheral Status 1	word	read	read	read	on page 6-164
0xC3FD_C068	ME_PS2	Peripheral Status 2	word	read	read	read	on page 6-165
0xC3FD_C080	ME_RUN_PC0	Run Peripheral Configuration 0	word	read	read/write	read/write	on page 6-166
0xC3FD_C084	ME_RUN_PC1	Run Peripheral Configuration 1	word	read	read/write	read/write	on page 6-166
...							

Table 37. MC_ME Register Description (continued)

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FD_C09C	ME_RUN_PC7	Run Peripheral Configuration 7	word	read	read/write	read/write	on page 6-166
0xC3FD_C0A0	ME_LP_PC0	Low-Power Peripheral Configuration 0	word	read	read/write	read/write	on page 6-167
0xC3FD_C0A4	ME_LP_PC1	Low-Power Peripheral Configuration 1	word	read	read/write	read/write	on page 6-167
...							
0xC3FD_C0BC	ME_LP_PC7	Low-Power Peripheral Configuration 7	word	read	read/write	read/write	on page 6-167
0xC3FD_C0C4	ME_PCTL4	DSPI_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0C5	ME_PCTL5	DSPI_1 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0C6	ME_PCTL6	DSPI_2 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0D0	ME_PCTL16	FlexCAN_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0DA	ME_PCTL26	SafetyPort Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0E0	ME_PCTL32	ADC_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0E3	ME_PCTL35	CTU Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0E6	ME_PCTL38	eTimer_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0E9	ME_PCTL41	FlexPWM_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0F0	ME_PCTL48	LIN_FLEX_0 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C0F1	ME_PCTL49	LIN_FLEX_1 Control	byte	read	read/write	read/write	on page 6-167
0xC3FD_C11C	ME_PCTL92	PIT Control	byte	read	read/write	read/write	on page 6-167

Note: Any access to unused registers as well as write accesses to read-only registers will not change register content, and cause a transfer error.

Table 38. MC_ME 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
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	0	S_PLL0	S_XOSC0	S_16 MHz_IRC	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	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		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	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W													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	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																
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

Table 38. MC_ME 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
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	16 MHz_IRC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	0	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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
		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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																

Table 38. MC_ME 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
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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																

Table 38. MC_ME 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
0xC3FD_C044	reserved																	
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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C04C … 0xC3FD_C05C	reserved																	
0xC3FD_C060	ME_PS0	R	0	0	0	0	0	S_SafetyPort	0	0	0	0	0	0	0	0	0	S_FlexCAN_0
		W																
		R	0	0	0	0	0	0	0	0	0	S_DSPI_2	S_DSPI_1	S_DSPI_0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	S_LIN_FLEX_1	S_LIN_FLEX_0
		W																
		R	0	0	0	0	0	0	S_FlexPWM_0	0	0	S_eTimer_0	0	0	S_CTU	0	0	S_ADC_0
		W																

Table 38. MC_ME Memory Map (continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	4 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT	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	0
		W																
0xC3FD_C06C	reserved																	
0xC3FD_C070	reserved																	
0xC3FD_C074 … 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																
0xC3FD_C0C0 … 0xC3FD_C14C	ME_PCTL0…143	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																	

6.3.2 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 according to big endian. 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.

Global Status Register (ME_GS)

Figure 43. 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	0	S_PLL0	S_XOSC0	S_16 MHz_IRC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register contains global mode status.

Table 39. 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

Table 39. Global Status Register (ME_GS) Field Descriptions (continued)

Field	Description
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_PLL0	**system PLL status** 0 system PLL is not stable 1 system PLL is providing a stable clock
S_XOSC0	**4 MHz crystal oscillator status** 0 4 MHz crystal oscillator is not stable 1 4 MHz crystal oscillator is providing a stable clock
S_16 MHz_IRC	**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 int. RC osc. 0001 reserved 0010 4 MHz crystal osc. 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

Mode Control Register (ME_MCTL)

Figure 44. 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

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 as a predefined key is required to change its value.

Table 40. 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: first time with key, and second time with 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 reserved
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)

Mode Enable Register (ME_ME)

Figure 45. 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	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

This register allows a way to disable the device modes which are not required for a given device. RESET, SAFE, DRUN, and RUN0 modes are always enabled.

Table 41. Mode Enable Register (ME_ME) Field Descriptions

Field	Description
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	**RESET mode enable** 0 RESET mode is disabled 1 RESET mode is enabled

Interrupt Status Register (ME_IS)

Figure 46. 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	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the current interrupt status.

Table 42. Interrupt Status Register (ME_IS) Field Descriptions

Field	Description
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

Interrupt Mask Register (ME_IM)

Figure 47. 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	0	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

This register controls whether an event generates an interrupt or not.

Table 43. Interrupt Mask Register (ME_IM) Field Descriptions

Field	Description
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

Invalid Mode Transition Status Register (ME_IMTS)

Figure 48. 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

This register provides the status bits for the possible causes of an invalid mode interrupt.

Table 44. 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 6.4.5, "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, SAFE event bit is pending and a new mode 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

Debug Mode Transition Status Register (ME_DMTS)

Figure 49. 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	16 MHz_IRC_SC	SCSRC_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	0	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

This register provides the status of different factors which 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 45. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions

Field	Description
PREVIOUS_MODE	Previous **device mode** — These bits show the mode in which the device was prior to the latest 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/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 which 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 which 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 45. Debug Mode Transition Status Register (ME_DMTS) Field Descriptions (continued)

Field	Description
16 MHz_IRC_SC	16 MHz_IRC 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 1A state change is taking place
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending 1A 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 1A 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 1A 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 which 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 1Clock 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 which 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 1Clock disabling is pending for at least one peripheral
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 which 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 1Clock 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 which 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 1Clock disabling is pending for at least one peripheral

RESET Mode Configuration Register (ME_RESET_MC)

Figure 50. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during RESET mode. Please refer to *Table 46* for details.

TEST Mode Configuration Register (ME_TEST_MC)

Figure 51. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during TEST mode. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

SAFE Mode Configuration Register (ME_SAFE_MC)

Figure 52. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during SAFE mode. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

DRUN Mode Configuration Register (ME_DRUN_MC)

Figure 53. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during DRUN mode. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

RUN0…3 Mode Configuration Registers (ME_RUN0…3_MC)

Figure 54. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during RUN0…3 modes. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

HALT0 Mode Configuration Register (ME_HALT0_MC)

Figure 55. 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	0	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	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during HALT0 mode. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

Note: The reset value of the DFLAON field in the ME_HALT0_MC register is "10". This reset value is illegal for the data flash. Thus, the reset value for the HALT0 mode configuration cannot be used as is and must be set to a legal value before requesting the entry of the HALT0 mode.

STOP0 Mode Configuration Register (ME_STOP0_MC)

Figure 56. 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	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

This register configures system behavior during STOP0 mode. Please refer to *Table 46* for details.

Note: Byte write accesses are not allowed to this register.

Table 46. 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. 1 Main voltage regulator is switched on

Table 46. Mode Configuration Registers (ME_<*mode*>_MC) Field Descriptions (continued)

Field	Description
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. 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
PLL0ON	**system PLL control** 0 system PLL is switched off 1 system PLL is switched on
XOSC0ON	**4 MHz crystal oscillator control** 0 4 MHz crystal oscillator is switched off 1 4 MHz crystal oscillator is switched on
16 MHz_IRCON	**16 MHz internal RC oscillator control** 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 int. RC osc. 0001 reserved 0010 4 MHz crystal osc. 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

Peripheral Status Register 0 (ME_PS0)

Figure 57. 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	S_SafetyPort	0	0	0	0	0	0	0	0	0	S_FlexCAN_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_DSPI_2	S_DSPI_1	S_DSPI_0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 47* for details.

Peripheral Status Register 1 (ME_PS1)

Figure 58. 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	0	0	0	0	0	0	0	0	0	S_LIN_FLEX_1	S_LIN_FLEX_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	S_FlexPWM_0	0	0	S_eTimer_0	0	0	S_CTU	0	0	S_ADC_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 47* for details.

Peripheral Status Register 2 (ME_PS2)

Figure 59. 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	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides the status of the peripherals. Please refer to *Table 47* for details.

Table 47. 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. If no peripheral is mapped on a particular position (i.e., the corresponding *MODS* bit is ‘0’), the corresponding bit is always read as ‘0’. 0 Peripheral is frozen 1 Peripheral is active

Run Peripheral Configuration Registers (ME_RUN_PC0…7)

Figure 60. 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

These registers configure eight different types of peripheral behavior during run modes.

Table 48. 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

Low-Power Peripheral Configuration Registers (ME_LP_PC0…7)

Figure 61. 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

These registers configure eight different types of peripheral behavior during non-run modes.

Table 49. 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

Peripheral Control Registers (ME_PCTL0…143)

Figure 62. Peripheral Control Registers (ME_PCTL0…143)

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

These registers select the configurations during run and non-run modes for each peripheral.

Table 50. Peripheral Control Registers (ME_PCTL0…143) 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 accesses.*
LP_CFG	**Peripheral configuration select for non-run 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 run 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

6.4 Functional Description

6.4.1 Mode Transition Request

The transition from one mode to another mode is normally handled by software by accessing the mode control register ME_MCTL. But the in 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 writing

- the first time with the value of the key (0x5AF0) into the KEY bit field and the required target mode into the TARGET_MODE bit field,
- and the second time 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. The mode transition request may require a number of cycles depending on the programmed configuration, 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 6.4.5, "Mode Transition Interrupts"*.

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means that transition requests such as RUN0…3 Æ RUN0…3, DRUN Æ DRUN, SAFE Æ SAFE, and TEST Æ TEST are considered valid mode transition requests. As soon as the mode request is accepted as valid, the S_MTRANS bit is set till 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.

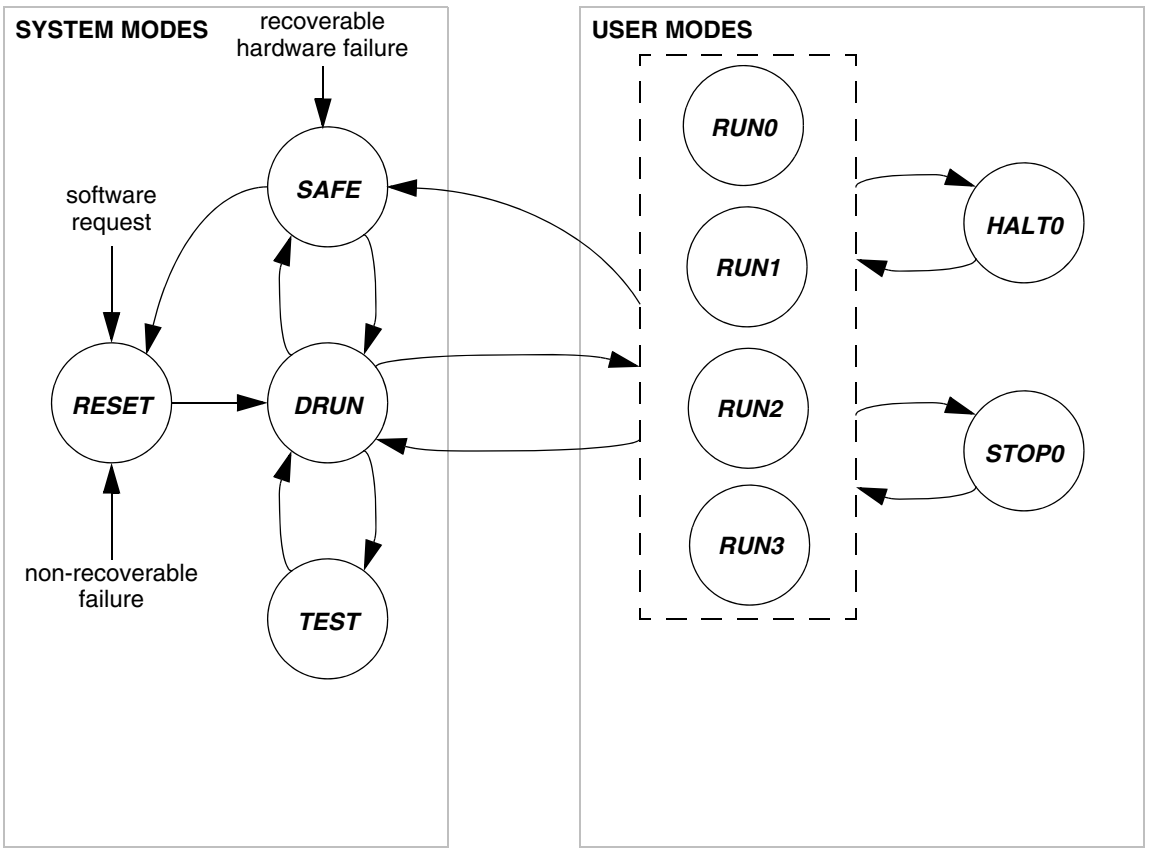


Figure 63. MC_ME Mode Diagram

6.4.2 Modes Details

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 int. RC osc. is selected as the system clock.

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 int. RC osc. 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 to be in the low-power or power-down state in this mode.

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 the MC_RGM chapter 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 int. RC osc. 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 to change to the 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 the SAFE mode.

As long as a SAFE event is active, the system remains in the 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 the DRUN mode, or
  - completely reset the device via the 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.

TEST Mode

The device enters this mode on the following events:

- from the DRUN mode when the TARGET_MODE bit field of the ME_MCTL register is written with “0001”

As soon as any of the above events 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 to be in the low-power or power-down state in this mode.

RUN0…3 Modes

The device enters one of these modes on the following events:

- from the DRUN, SAFE, 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 to be in the low-power or power-down state in this mode.

HALT0 Mode

The device enters this mode on the following events:

- 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 any of the above events has occurred, a HALT0 mode transition request is generated. The mode configuration information for this mode is provided by ME_HALT0_MC register. This mode is quite configurable, and the ME_HALT0_MC register should be programmed according to the system needs. The flashes can be put in low-power or power-down mode as needed. If there is a HALT0 mode request while an interrupt request is active, the transition to HALT0 is aborted with the resultant mode being 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

and to be used by software

- to wait until it is required to do something and then to react quickly (i.e., within a few system clock cycles of an interrupt event)

STOP0 Mode

The device enters this mode on the following events:

- 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 any of the above events 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 fully configurable, and the ME_STOP0_MC register should be programmed according to the system needs. The following clock sources are switched off in this mode:

- the system PLL

The flashes can be put in power-down mode as needed. If there is a STOP0 mode request 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), and an invalid mode interrupt is not generated.

This can be used as an advanced low-power mode with the core clock frozen and almost all peripherals stopped.

This mode is intended as an advanced low-power mode with

- the core clock frozen
- almost all peripherals stopped

and to be used by software

- to wait until it is required to do something with no need to react quickly (e.g., 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.

6.4.3 Mode Transition Process

The process of mode transition follows 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.

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 6.4.5, "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.

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 . A '√' indicates that a given resource is configurable for a given mode.

Table 51. MC_ME Resource Control Overview

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0…3***	***HALT0***	***STOP0***
16 MHz_IRC	on	√ on	on	on	on	√ on	√ on
XOSC0	off	√ off	off	√ off	√ off	√ off	√ off
PLL0	off	√ off	off	√ off	√ off	√ off	off
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
MVREG	on	on	on	on	on	√ on	√ on

Table 51. MC_ME Resource Control Overview (continued)

Resource	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0...3***	***HALT0***	***STOP0***
PDO	off	√ off	√ on	off	off	off	√ off

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, and the peripheral control registers ME_PCTL0…143

Warning: 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 regardless of the status of the peripherals' stop acknowledges.

Please refer to *Section 6.4.6, "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.

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.

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.

Warning: Clocks to the whole device including the processor and system memory can be disabled in TEST mode.

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:

- the 16 MHz internal RC oscillator
- the 4 MHz crystal oscillator
- the system PLL

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 which 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.

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 or power-down mode 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.

Warning: 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.

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

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_PCTL0…143, the MC_ME enables the clocks for selected modules as required. This step is executed only after the process is completed.

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.

Processor Low-Power Mode Exit

If the mode transition is from any of the low-power modes HALT0 orSTOP0 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.

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 int. RC osc. takes effect only after the S_16 MHz_IRC bit of the ME_GS register is set by hardware (i.e., the 16 MHz internal RC oscillator has stabilized).
- The target clock configuration for the 4 MHz crystal osc. takes effect only after the S_XOSC0 bit of the ME_GS register is set by hardware (i.e the 4 MHz crystal 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 (i.e., 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 theTEST 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
- 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 52*. A ‘√’ indicates that a given clock source is selectable for a given mode.

Table 52. MC_ME System Clock Selection Overview

System Clock Source	Mode						
	RESET	***TEST***	***SAFE***	***DRUN***	***RUN0...3***	***HALT0***	***STOP0***
16 MHz int. RC osc.	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)
4 MHz crystal osc.		√		√	√	√	√
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

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.

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 switched off at this step:

- the 4 MHz crystal oscillator
- the system PLL

This step is executed only after the *System Clock Switching* process has completed.

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 which 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 switched off at this step:

- the 16 MHz internal RC 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

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 its low-power or power-down mode, 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.

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 the 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 is expected, the ME_DMTS register can indicate which process is still in progress.

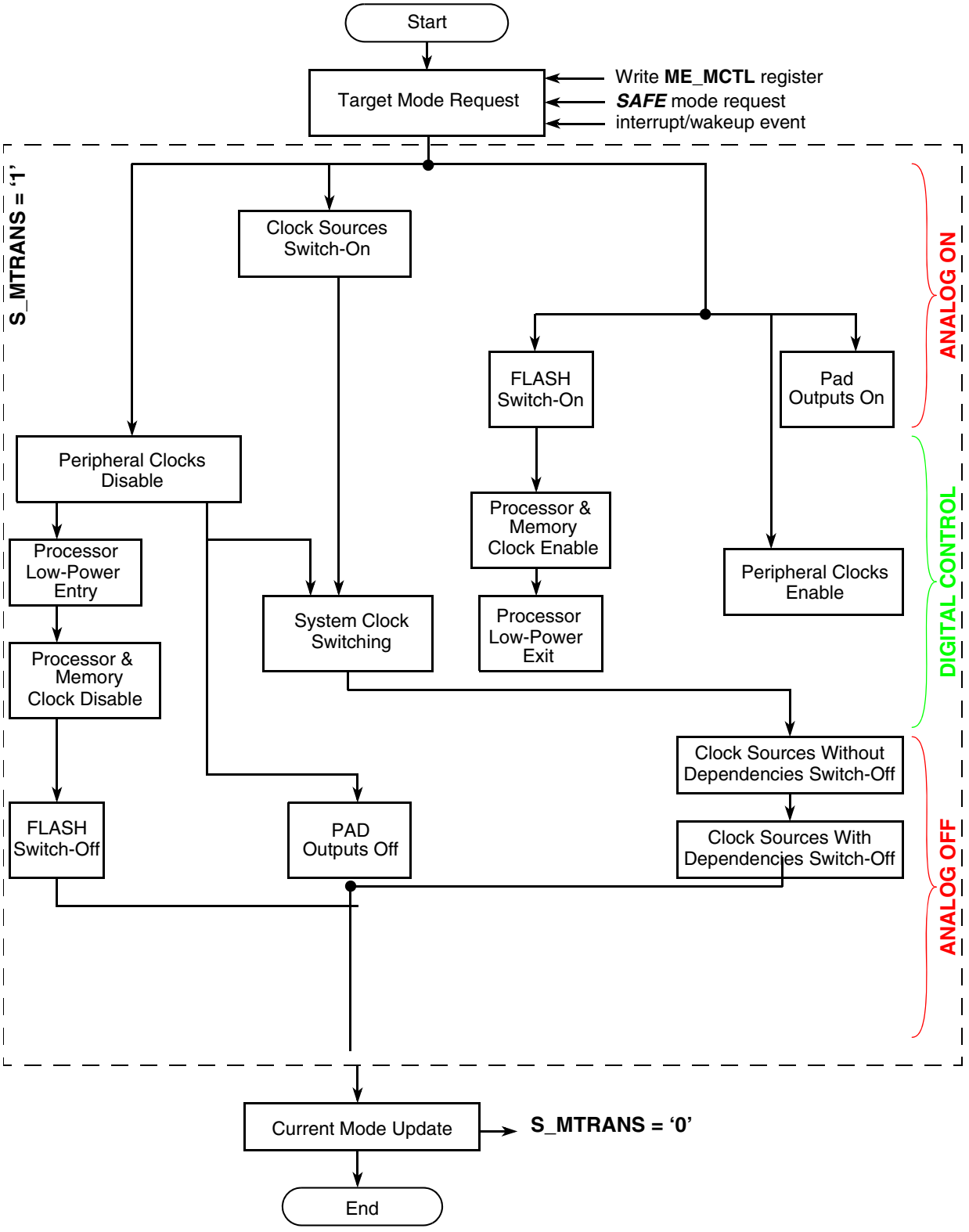


Figure 64. MC_ME Transition Diagram

6.4.4 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 int. RC osc. is selected as the system clock, 16 MHz_IRC must be on.
- If the 4 MHz crystal osc. clock is selected as the system clock, OSC must be on.
- If the system PLL clock is selected as the system clock, PLL must be on.
- The 4 MHz crystal oscillator must be on if the system PLL is on. Therefore, when writing a '1' to PLL0ON, a '1' must also be written to XOSC0ON.

Note: Software must ensure that clock sources with dependencies other than those mentioned above are swithced 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 theTEST mode, and only in this case may all system clock sources be turned off.

Warning: If the system clock is stopped during TEST mode, the device can exit only via a system reset.

6.4.5 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, and indicating when a mode transition has completed.

Invalid Mode Configuration Interrupt

Whenever a write operation is attempted to the ME_<mode>_MC registers violating the protection rules mentioned in the *Section 6.4.4, "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'.

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 (i.e., 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: As 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 the 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 whatever reason. The parent mode is the device mode before a valid mode request was made.
- Self-transition requests (e.g., RUN0 Æ RUN0) are not considered as invalid even when the mode transition process is active (i.e., 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 (i.e., all status bits of the ME_GS register match with configuration bits in the ME_<mode>_MC register), then software can only request the 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'.

SAFE Mode Transition Interrupt

Whenever the system enters the 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 (i.e., programming of ME_MCTL register).

Mode Transition Complete interrupt

Whenever the system fully completes a mode transition (i.e., 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.

6.4.6 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 which 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_PCTL0…143 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 which 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_PCTL0…143 registers.

Any modifications to the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL0…143 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_PCTL0…143 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_PCTL0…143 registers during a debug session will take affect immediately without requiring any new mode request.

6.4.7 Application Example

Figure 65 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

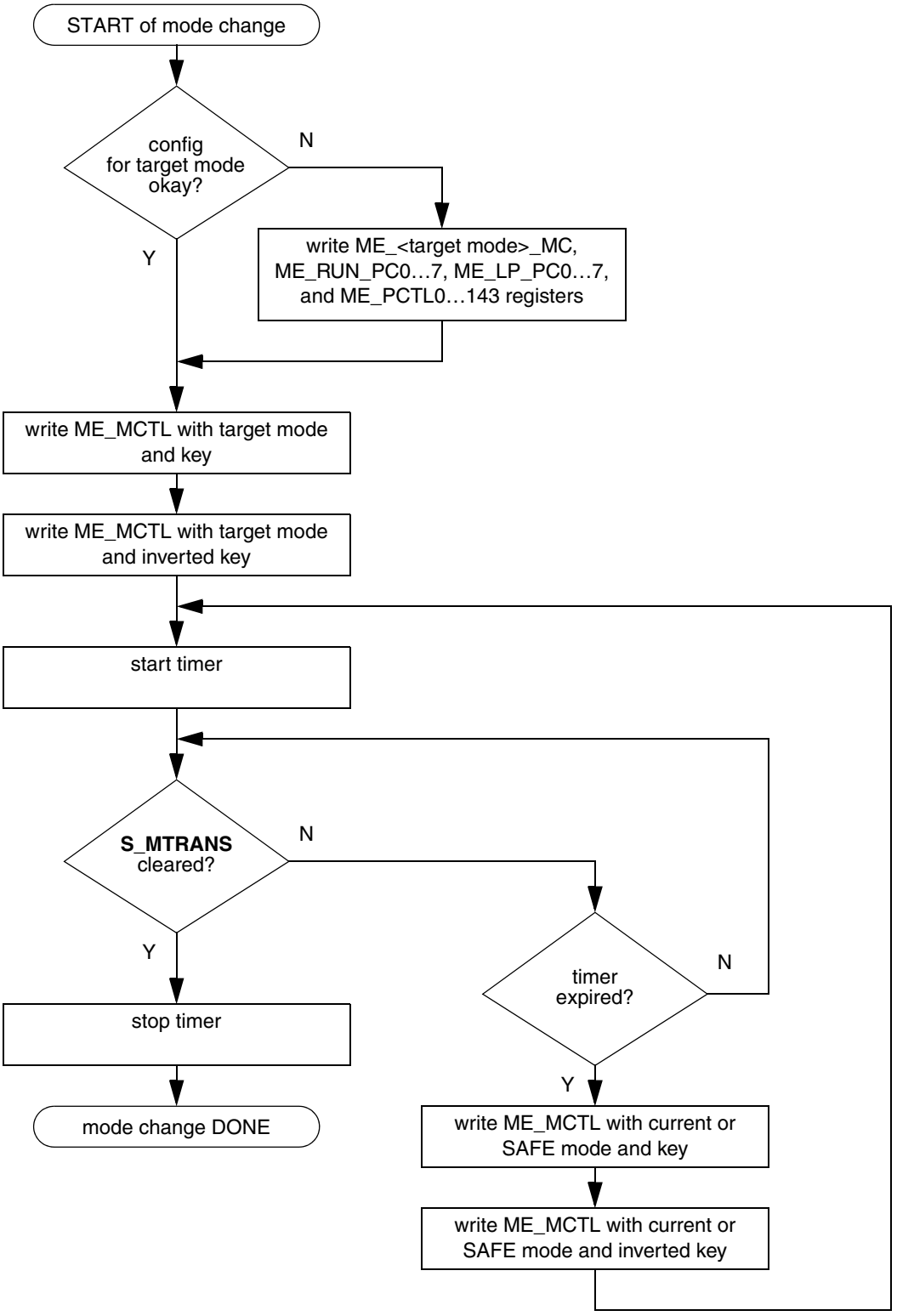


Figure 65. MC_ME Application Example Flow Diagram

7 Power Control Unit (MC_PCU)

7.1 Introduction

7.1.1 Overview

The power control unit (MC_PCU) acts as a bridge for mapping the PMU peripheral to the MC_PCU address space.

Figure 66 depicts the MC_PCU block diagram.

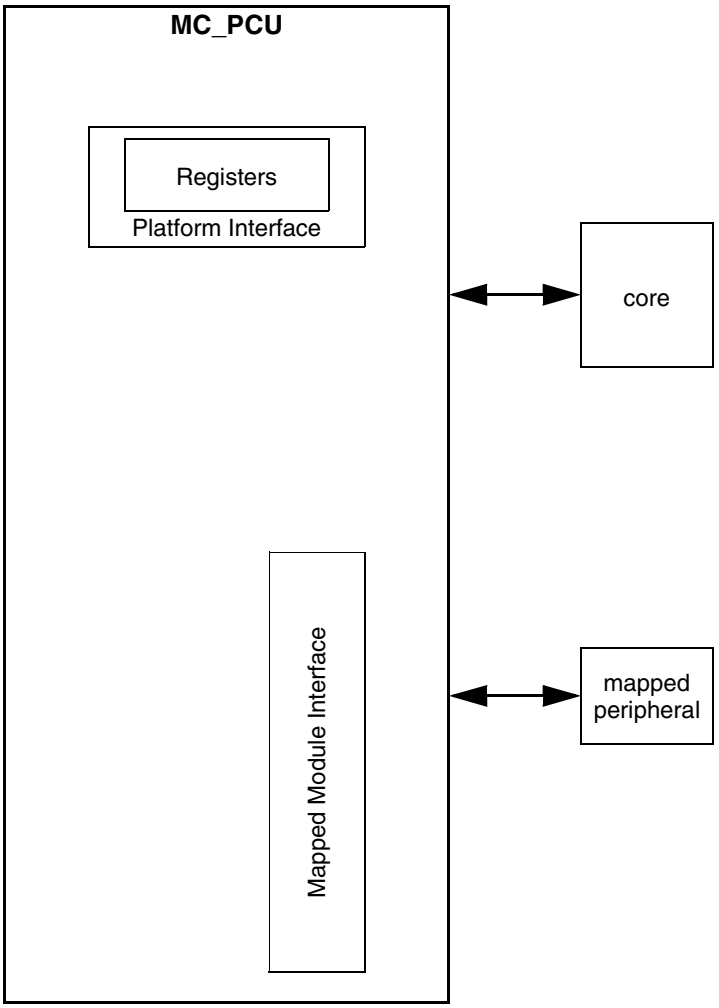


Figure 66. MC_PCU Block Diagram

7.1.2 Features

The MC_PCU includes the following features:

- maps the PMU registers to the MC_PCU address space

7.2 External Signal Description

The MC_PCU has no connections to any external pins.

7.3 Memory Map and Register Definition

7.3.1 Memory Map

Table 53. 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 7-186*

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 54. 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																	
0xC3FE_8080 … 0xC3FE_80FC	PMU registers																	
0xC3FE_8100 … 0xC3FE_BFFC	reserved																	

7.3.2 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 **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.

Power Domain Status Register (PCU_PSTAT)

Figure 67. 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

This register reflects the power status of all available power domains.

Table 55. Power Domain Status Register (PCU_PSTAT) Field Descriptions

Field	Description
PD*n*	Power status for power domain #*n* 0Power domain is inoperable 1Power domain is operable

8 Reset Generation Module (MC_RGM)

8.1 Introduction

8.1.1 Overview

The reset generation module (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 which controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and controls the reset signals generated in the system.

Figure 68 depicts the MC_RGM block diagram.

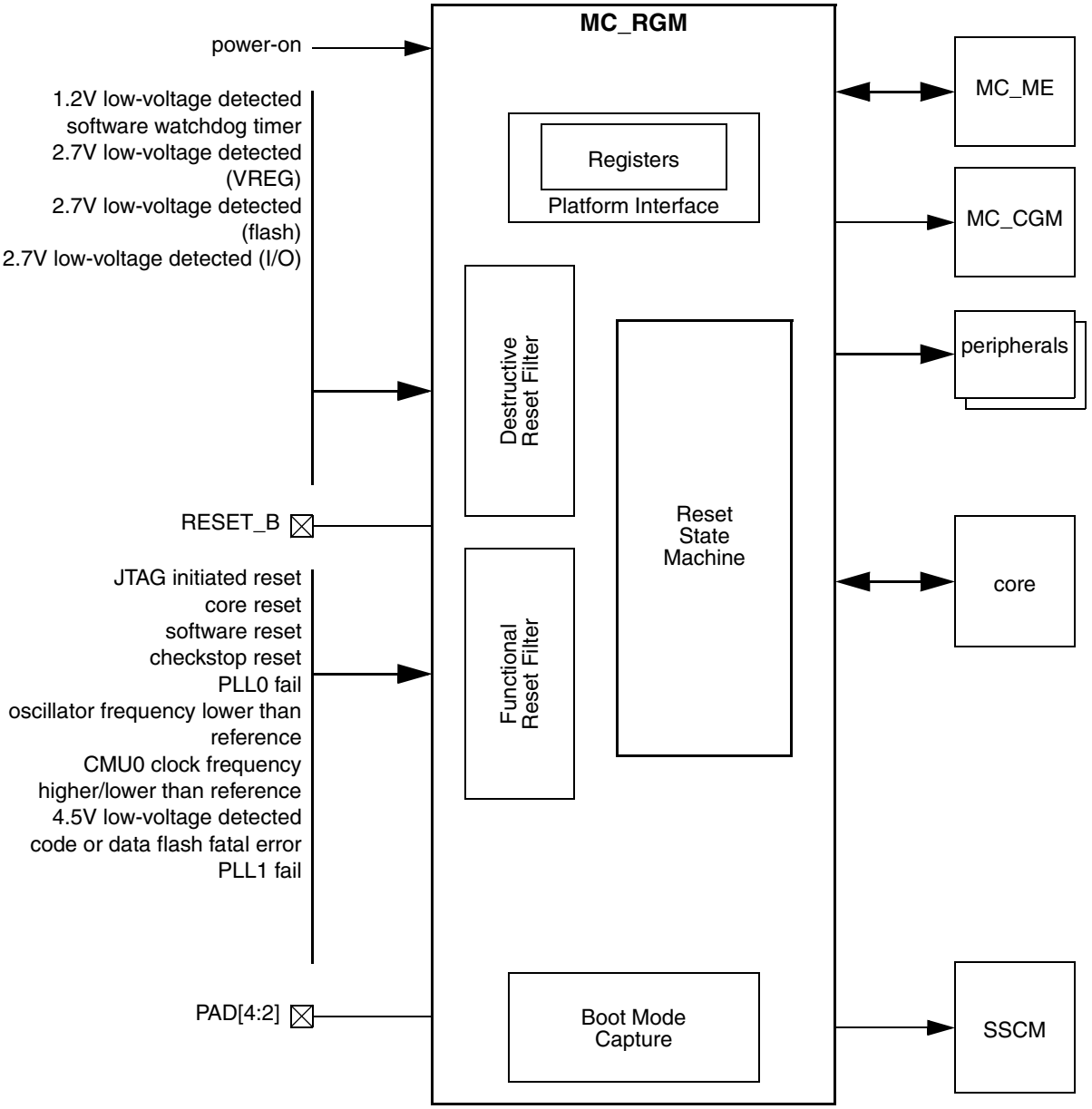


Figure 68. MC_RGM Block Diagram

8.1.2 Features

The MC_RGM contains the functionality for the following features:

- ‘destructive’ resets management
- ‘functional’ resets management
- signalling 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 RESET_B deassertion

8.1.3 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 start-up state for both digital and analog modules. ‘Destructive’ resets are
  - power-on reset
  - 1.2V low-voltage detected
  - software watchdog timer
  - 2.7V low-voltage detected (VREG)
  - 2.7V low-voltage detected (flash)
  - 2.7V low-voltage detected (I/O)
  - comparator error
- 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, while analog modules or specific digital modules’ (e.g., debug modules, flash modules) state is preserved. ‘Functional’ resets are
  - external reset
  - JTAG initiated reset
  - core reset
  - software reset
  - checkstop reset
  - PLL0 fail
  - oscillator frequency lower than reference
  - CMU0 clock frequency higher/lower than reference
  - 4.5V low-voltage detected
  - code or data flash fatal error
  - PLL1 fail

When a reset is triggered, the MC_RGM state machine is activated and proceeds through the different phases (i.e., PHASEn 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 from either 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 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 , "Functional Event Reset Disable Register (RGM_FERD)* and *Section , "Functional Event Alternate Request Register (RGM_FEAR)* for 'functional' resets).

8.2 External Signal Description

The MC_RGM interfaces to the bidirectional reset pin RESET_B and the boot mode pins PAD[4:2].

8.3 Memory Map and Register Definition

Table 56. MC_RGM Register Description

Address	Name	Description	Size	Access			Location
				User	Supervisor	Test	
0xC3FE_4000	RGM_FES	Functional Event Status	half-word	read	read/write(1)	read/write (1)	*on page 8-192*
0xC3FE_4002	RGM_DES	Destructive Event Status	half-word	read	read/write(1)	read/write (1)	*on page 8-194*
0xC3FE_4004	RGM_FERD	Functional Event Reset Disable	half-word	read	read/write(2)	read/write (2)	*on page 8-195*
0xC3FE_4006	RGM_DERD	Destructive Event Reset Disable	half-word	read	read	read	*on page 8-197*
0xC3FE_4010	RGM_FEAR	Functional Event Alternate Request	half-word	read	read/write	read/write	*on page 8-198*
0xC3FE_4018	RGM_FESS	Functional Event Short Sequence	half-word	read	read/write	read/write	*on page 8-199*
0xC3FE_401C	RGM_FBRE	Functional Bidirectional Reset Enable	half-word	read	read/write	read/write	*on page 8-200*

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, and cause a transfer error.

Table 57. 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	0	0	0	0	0	F_PLL1	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
		W	w1c						w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	F_SWT	0	F_LVD12
		W	w1c									w1c	w1c	w1c		w1c		w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	0	0	0	0	0	D_PLL1	D_FLASH	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	D_SWT	0	D_LVD12
		W																
0xC3FE_4008 … 0xC3FE_400C	reserved																	
0xC3FE_4010	RGM_FEAR	R	0	0	0	0	0	0	AR_PLL1	0	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	0	0	AR_CORE	AR_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_4014	reserved																	

Table 57. 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_4018	RGM_FESS	R	SS_EXR	0	0	0	0	0	SS_PLL1	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	BE_EXR	0	0	0	0	0	BE_PLL1	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	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																	

8.3.1 Register Descriptions

Unless otherwise noted, 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 RGM_DES[8:15] register bits may be accessed as a word at address 0xC3FE_4000, as a half-word at address 0xC3FE_4002, or as a byte at address 0xC3FE_4004.

Functional Event Status Register (RGM_FES)

Figure 69. Functional Event Status Register (RGM_FES)

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	0	0	0	0	0	F_PLL1	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
W	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

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’.

Table 58. Functional Event Status Register (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_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_FLASH	**Flag for code or data flash fatal error** 0 No code or data flash fatal error event has occurred since either the last clear or the last destructive reset assertion 1 A code or data flash fatal error event has occurred
F_LVD45	**Flag for 4.5V low-voltage detected** 0 No 4.5V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion 1 A 4.5V low-voltage detected event has occurred
F_CMU0_FHL	**Flag for CMU0 clock frequency higher/lower than reference** 0 No CMU0 clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A CMU0 clock frequency higher/lower than reference event has occurred
F_CMU0_OLR	**Flag for oscillator frequency lower than reference** 0 No oscillator frequency lower than reference event has occurred since either the last clear or the last destructive reset assertion 1 A oscillator frequency lower than reference 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_CHKSTOP	**Flag for checkstop reset** 0 No checkstop reset event has occurred since either the last clear or the last destructive reset assertion 1 A checkstop reset event has occurred
F_SOFT	**Flag for software reset** 0 No software reset event has occurred since either the last clear or the last destructive reset assertion 1 A software reset event has occurred

Table 58. Functional Event Status Register (RGM_FES) Field Descriptions (continued)

Field	Description
F_CORE	**Flag for core reset** 0 No core reset event has occurred since either the last clear or the last destructive reset assertion 1 A core reset event has occurred
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

Destructive Event Status Register (RGM_DES)

Figure 70. Destructive Event Status Register (RGM_DES)

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	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	0	F_SWT	0	F_LVD12
W	w1c									w1c	w1c	w1c		w1c		w1c
POR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write ‘1’.

Table 59. Destructive Event Status Register (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_LVD27_IO	**Flag for 2.7V low-voltage detected (I/O)** 0 No 2.7V low-voltage detected (I/O) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (I/O) event has occurred
F_LVD27_FLASH	**Flag for 2.7V low-voltage detected (flash)** 0 No 2.7V low-voltage detected (flash) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (flash) event has occurred

Table 59. Destructive Event Status Register (RGM_DES) Field Descriptions (continued)

Field	Description
F_LVD27_VREG	**Flag for 2.7V low-voltage detected (VREG)** 0 No 2.7V low-voltage detected (VREG) event has occurred since either the last clear or the last power-on reset assertion 1 A 2.7V low-voltage detected (VREG) 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 power-on reset assertion 1 A software watchdog timer event has occurred
F_LVD12	**Flag for 1.2V low-voltage detected** 0 No 1.2V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion 1 A 1.2V low-voltage detected event has occurred

Note: The F_POR flag is automatically cleared on a 1.2 V low-voltage detected or a 2.7 V low-voltage detected. This means that if the power-up sequence is not monotonic (i.e., the voltage rises and then drops enough to trigger a low-voltage detection), the F_POR flag may not be set but instead the <register>F_LVD12 or <register>F_LVD27_VREG flag is set on exiting the reset sequence. Therefore, if the F_POR, <register>F_LVD12 or <register>F_LVD27_VREG flags are set on reset exit, software should interpret the reset cause as power-on.

Functional Event Reset Disable Register (RGM_FERD)

Figure 71. Functional Event Reset Disable Register (RGM_FERD)

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	0	0	0	0	0	D_PLL1	D_FLASH	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register provides dedicated bits to disable functional reset sources.When a functional reset source is disabled, the associated functional event will trigger either a SAFE mode request or an interrupt request (see *Section , "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. Each byte can be written only once after power-on reset.

Table 60. Functional Event Reset Disable Register (RGM_FERD) Field Descriptions

Field	Description
D_EXR	**Disable External Reset** 0 An external reset event triggers a reset sequence
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_FLASH	**Disable code or data flash fatal error** 0 A code or data flash fatal error event triggers a reset sequence
D_LVD45	**Disable 4.5V low-voltage detected** 0 A 4.5V low-voltage detected event triggers a reset sequence 1 A 4.5V low-voltage detected event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_LVD45
D_CMU0_FHL	**Disable CMU0 clock frequency higher/lower than reference** 0 A CMU0 clock frequency higher/lower than reference event triggers a reset sequence 1 A CMU0 clock frequency higher/lower than reference 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 frequency lower than reference** 0 A oscillator frequency lower than reference event triggers a reset sequence 1 A oscillator frequency lower than reference 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_CHKSTOP	**Disable checkstop resetl** 0 A checkstop reset event triggers a reset sequence
D_SOFT	**Disable software reset** 0 A software reset event triggers a reset sequence
D_CORE	**Disable core reset** 0 A core reset event triggers a reset sequence 1 A core reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_CORE
D_JTAG	**Disable JTAG initiated reset** 0 A JTAG initiated reset event triggers a reset sequence 1 A JTAG initiated reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR.AR_JTAG

Destructive Event Reset Disable Register (RGM_DERD)

Figure 72. Destructive Event Reset Disable Register (RGM_DERD)

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	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	D_SWT	0	D_LVD12
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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.

Table 61. Destructive Event Reset Disable Register (RGM_DERD) Field Descriptions

Field	Description
D_LVD27_IO	**Disable 2.7V low-voltage detected (I/O)** 0 A 2.7V low-voltage detected (I/O) event triggers a reset sequence
D_LVD27_FLASH	**Disable 2.7V low-voltage detected (flash)** 0 A 2.7V low-voltage detected (flash) event triggers a reset sequence
D_LVD27_VREG	**Disable 2.7V low-voltage detected (VREG)** 0 A 2.7V low-voltage detected (VREG) event triggers a reset sequence
D_SWT	**Disable software watchdog timer** 0 A software watchdog timer event triggers a reset sequence
D_LVD12	**Disable 1.2V low-voltage detected** 0 A 1.2V low-voltage detected event triggers a reset sequence

Functional Event Alternate Request Register (RGM_FEAR)

Figure 73. Functional Event Alternate Request Register (RGM_FEAR)

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	0	0	AR_PLL1	0	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	0	0	AR_CORE	AR_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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.

Table 62. Functional Event Alternate Request Register (RGM_FEAR) Field Descriptions

Field	Description
AR_PLL1	**Alternate Request for PLL1 fail** 0 Generate a SAFE mode request on a PLL1 fail event if the reset is disabled 1 Generate an interrupt request on a PLL1 fail event if the reset is disabled
AR_LVD45	**Alternate Request for 4.5V low-voltage detected** 0 Generate a SAFE mode request on a 4.5V low-voltage detected event if the reset is disabled 1 Generate an interrupt request on a 4.5V low-voltage detected event if the reset is disabled
AR_CMU0_FHL	**Alternate Request for CMU0 clock frequency higher/lower than reference** 0 Generate a SAFE mode request on a CMU0 clock frequency higher/lower than reference event if the reset is disabled 1 Generate an interrupt request on a CMU0 clock frequency higher/lower than reference event if the reset is disabled
AR_CMU0_OLR	**Alternate Request for oscillator frequency lower than reference** 0 Generate a SAFE mode request on a oscillator frequency lower than reference event if the reset is disabled 1 Generate an interrupt request on a oscillator frequency lower than reference event if the reset is disabled
AR_PLL0	**Alternate Request for PLL0 fail** 0 Generate a SAFE mode request on a PLL0 fail event if the reset is disabled 1 Generate an interrupt request on a PLL0 fail event if the reset is disabled
AR_CORE	**Alternate Request for core reset** 0 Generate a SAFE mode request on a core reset event if the reset is disabled 1 Generate an interrupt request on a core reset event if the reset is disabled
AR_JTAG	**Alternate Request for JTAG initiated reset** 0 Generate a SAFE mode request on a JTAG initiated reset event if the reset is disabled 1 Generate an interrupt request on a JTAG initiated reset event if the reset is disabled

Functional Event Short Sequence Register (RGM_FESS)

Figure 74. Functional Event Short Sequence Register (RGM_FESS)

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	0	0	0	0	0	SS_PLL1	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register defines which reset sequence will be done 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 fast reset sequence, for example to skip flash reset.

It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 63. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions

Field	Description
SS_EXR	**Short Sequence for External Reset** 0 The reset sequence triggered by an **external** reset event will start from **PHASE1**
SS_PLL1	**Short Sequence for PLL1 fail** 0 The reset sequence triggered by a PLL1 fail event will start from PHASE1 1 The reset sequence triggered by a PLL1 fail event will start from PHASE3, skipping PHASE1 and PHASE2
SS_FLASH	**Short Sequence for code or data flash fatal error** 0 The reset sequence triggered by a code or data flash fatal error event will start from PHASE1
SS_LVD45	**Short Sequence for 4.5V low-voltage detected** 0 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE1 1 The reset sequence triggered by a 4.5V low-voltage detected event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU0_FHL	**Short Sequence for CMU0 clock frequency higher/lower than reference** 0 The reset sequence triggered by a CMU0 clock frequency higher/lower than reference event will start from PHASE1 1 The reset sequence triggered by a CMU0 clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CMU0_OLR	**Short Sequence for oscillator frequency lower than reference** 0 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE1 1 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2

Table 63. Functional Event Short Sequence Register (RGM_FESS) Field Descriptions

Field	Description
SS_PLL0	**Short Sequence for PLL0 fail** 0 The reset sequence triggered by a PLL0 fail event will start from PHASE1 1 The reset sequence triggered by a PLL0 fail event will start from PHASE3, skipping PHASE1 and PHASE2
SS_CHKSTOP	**Short Sequence for checkstop reset** 0 The reset sequence triggered by a checkstop reset event will start from PHASE1
SS_SOFT	**Short Sequence for software reset** 0 The reset sequence triggered by a software reset event will start from PHASE1
SS_CORE	**Short Sequence for core reset** 0 The reset sequence triggered by a core reset event will start from PHASE1 1 The reset sequence triggered by a core reset event will start 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 will start from PHASE1 1 The reset sequence triggered by a JTAG initiated reset event will start from PHASE3, skipping PHASE1 and PHASE2

Note: This register is reset on any enabled 'destructive' or 'functional' reset event.

Functional Bidirectional Reset Enable Register (RGM_FBRE)

Figure 75. Functional Bidirectional Reset Enable Register (RGM_FBRE)

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	0	0	0	0	0	BE_PLL1	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
W																
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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.

Table 64. Functional Bidirectional Reset Enable Register (RGM_FBRE) Field Descriptions

Field	Description
BE_EXR	**Bidirectional Reset Enable for External Reset** 0RESET_B is asserted on an **external** reset event if the reset is enabled 1RESET_B is not asserted on an **external** reset event
BE_PLL1	**Bidirectional Reset Enable for PLL1 fail** 0RESET_B is asserted on a PLL1 fail event if the reset is enabled 1RESET_B is not asserted on a PLL1 fail event
BE_FLASH	**Bidirectional Reset Enable for code or data flash fatal error** 0RESET_B is asserted on a code or data flash fatal error event if the reset is enabled 1RESET_B is not asserted on a code or data flash fatal error event
BE_LVD45	**Bidirectional Reset Enable for 4.5V low-voltage detected** 0RESET_B is asserted on a 4.5V low-voltage detected event if the reset is enabled 1RESET_B is not asserted on a 4.5V low-voltage detected event
BE_CMU0_FHL	**Bidirectional Reset Enable for CMU0 clock frequency higher/lower than reference** 0RESET_B is asserted on a CMU0 clock frequency higher/lower than reference event if the reset is enabled 1RESET_B is not asserted on a CMU0 clock frequency higher/lower than reference event
BE_CMU0_OLR	**Bidirectional Reset Enable for oscillator frequency lower than reference** 0RESET_B is asserted on a oscillator frequency lower than reference event if the reset is enabled 1RESET_B is not asserted on a oscillator frequency lower than reference event
BE_PLL0	**Bidirectional Reset Enable for PLL0 fail** 0RESET_B is asserted on a PLL0 fail event if the reset is enabled 1RESET_B is not asserted on a PLL0 fail event
BE_CHKSTOP	**Bidirectional Reset Enable for checkstop reset** 0RESET_B is asserted on a checkstop reset event if the reset is enabled 1RESET_B is not asserted on a checkstop reset event
BE_SOFT	**Bidirectional Reset Enable for software reset** 0RESET_B is asserted on a software reset event if the reset is enabled 1RESET_B is not asserted on a software reset event
BE_CORE	**Bidirectional Reset Enable for core reset** 0RESET_B is asserted on a core reset event if the reset is enabled 1RESET_B is not asserted on a core reset event
BE_JTAG	**Bidirectional Reset Enable for JTAG initiated reset** 0RESET_B is asserted on a JTAG initiated reset event if the reset is enabled 1RESET_B is not asserted on a JTAG initiated reset event

8.4 Functional Description

8.4.1 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 65*.

Table 65. 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 RESET_B 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 is comprised of five phases managed by a state machine, which 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 76*.

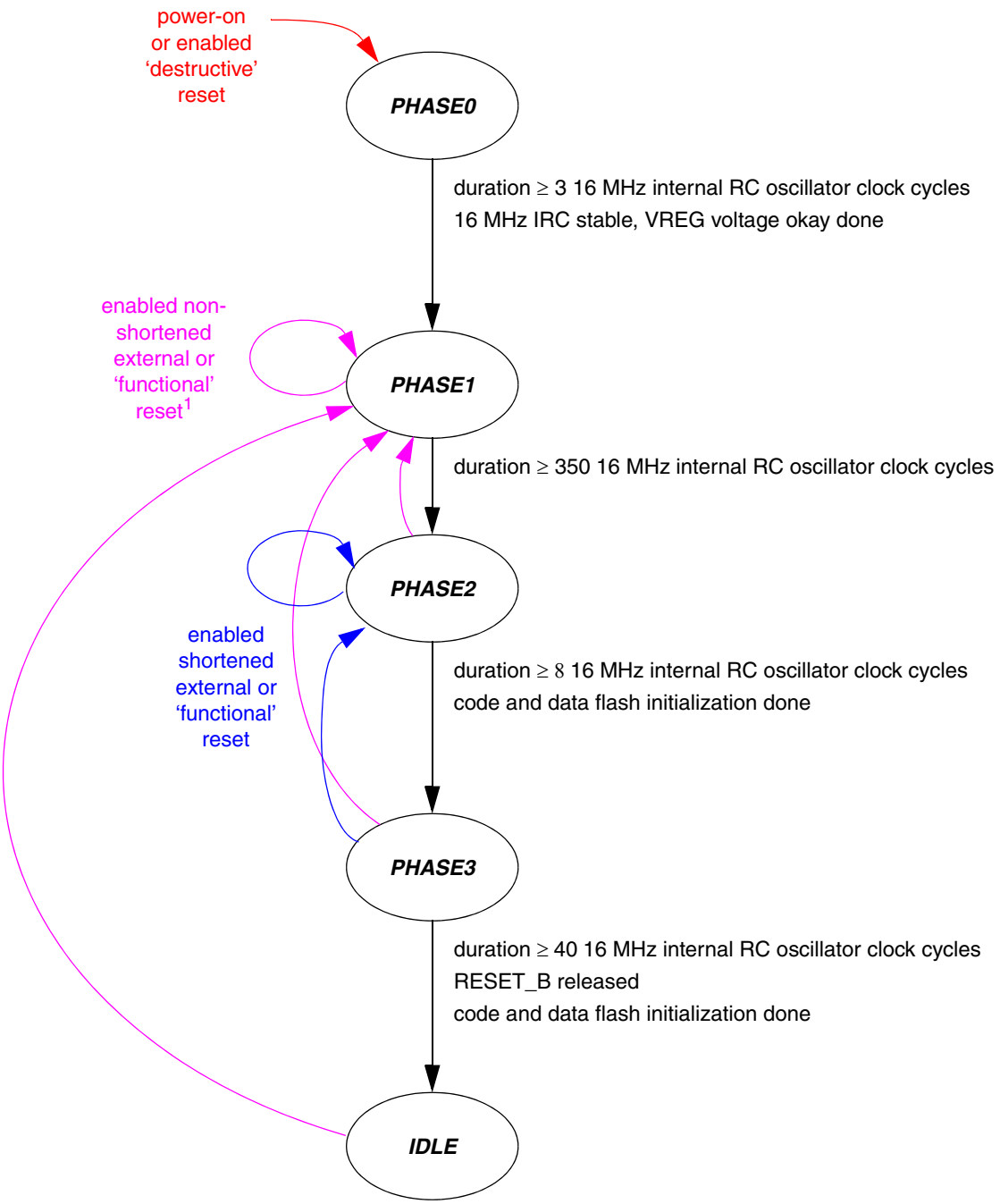


Figure 76. MC_RGM State Machine

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
  - 16 MHz IRC stable, VREG voltage okay
- a minimum of 3 16 MHz internal RC oscillator clock cycles have elapsed since power-up completion and the last enabled ‘destructive’ reset event

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 350 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled external or non-shortened ‘functional’ reset event

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

PHASE3 Phase

This phase is a 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
  - code and data flash initialization
- a minimum of 40 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled, shortened ‘functional’ reset event

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.

8.4.2 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 1.2V 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 will be 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 will trigger a reset sequence starting from the beginning of PHASE0.

8.4.3 External Reset

The MC_RGM manages the external reset coming from RESET_B. The detection of a falling edge on RESET_B will start 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 will normally trigger 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 will trigger 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.

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.

8.4.4 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 will normally trigger 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 will trigger 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 the MC_ME chapter for details on the STANDBY0 and DRUN modes.

8.4.5 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 either 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 66*.

Table 66. 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 (i.e., 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 which is configured to generate a SAFE mode/interrupt request occurs during PHASE1, it is ignored, and the MC_RGM will not send any safe mode/interrupt request to the MC_ME.

8.4.6 Boot Mode Capturing

The MC_RGM provides sampling of the boot mode PAD[4:2] for use by the system. This sampling is done five 16 MHz internal RC oscillator clock cycles before the rising edge of RESET_B. The result of the sampling is then provided to the system. For each bit, a value of ‘1’ is produced only if each of the oldest three of the five samples have the value ‘1’, otherwise a value of ‘0’ is produced.

Note: In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value to the device at least five 16 MHz internal RC oscillator clock periods before the external reset deassertion crosses the V_{IH} threshold.

Note: RESET_B can be low as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See Table 65 for details.)

9 Interrupt Controller (INTC)

9.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 128 interrupt requests. It is targeted to work with Power Architecture technology and automotive 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 peripheral's interrupt request 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. Because 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 tasks sharing the resource will not preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These software configurable interrupt requests can also be used to separate 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 configurable interrupt request to finish the servicing in a lower priority ISR. Therefore these software configurable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

9.2 Features

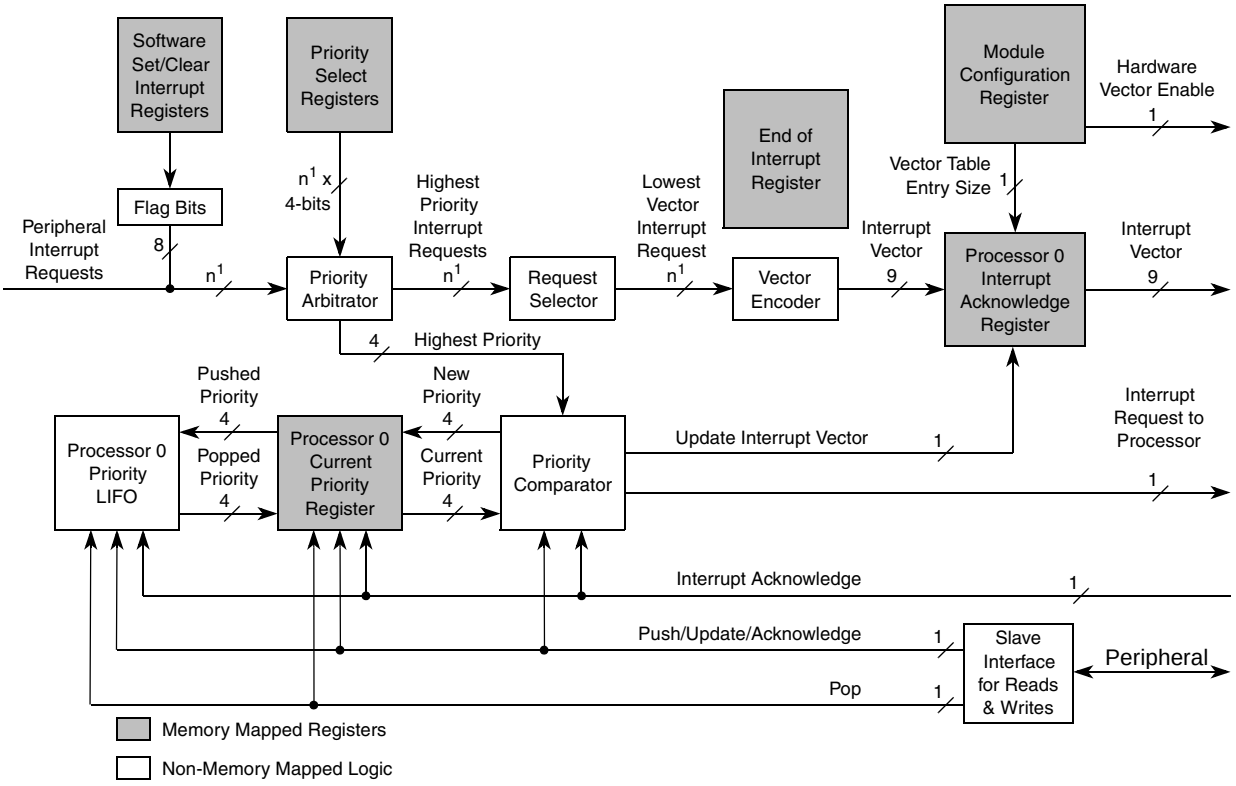
- Supports 120 peripheral interrupt and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Each interrupt source programmable to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts 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—3 clock cycles from receipt of interrupt request from peripheral to interrupt request to processor

Table 67. Interrupt sources available

Interrupt sources (128)	Number available
Software	8
ECSM	3
eDMA2x	17
SWT	1
STM	4
SIUL	4
MC_ME	4
MC_RGM	1
XOSC	1
PIT	4
ADC	3
FlexCAN	8
eTimer	8
FlexPWM	10
CTU	15
Safety Port	8
DSPI	15
LINFlex	6

9.3 Block diagram

Figure 77 shows a block diagram of the interrupt controller (INTC).



1. The total number of interrupt sources is 128, which includes 16 reserved sources and 8 software sources.

Figure 77. INTC block diagram

9.4 Modes of operation

9.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

Note: To correctly configure the interrupts in both software and hardware vector mode, the user must also configure the IVPR. The core register IVPR contains the base address for the interrupt handlers. Please refer to the core reference manual for more information.

Software vector mode

In software vector mode, the interrupt exception handler software must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN bit in INTC_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN bit is negated. The vector is read from

INC_IACKR. Reading the INTC_IACKR 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 INTC_CPR onto the associated LIFO and updates PRI in the associated INTC_CPR 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.

Hardware vector mode

In hardware vector mode, the hardware signals the interrupt vector from the INTC in conjunction with a processor that can use that vector. 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 configurable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN bit in the INTC_MCR 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 configurable interrupt request. The vector value matches the value of the INTVEC field in the INTC_IACKR field in the INTC_IACKR, depending on which processor was assigned to handle a given interrupt source.

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 the 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 register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR does not occur when the associated interrupt acknowledge signal asserts and INTC_SSCIR0_3–INTC_SSCIR4_7 is written at a time such that the PRI value in the associated INTC_CPR 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 is updated with the new priority, and the associated LIFO is neither pushed or popped.

Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

Stop mode

The INTC supports stop mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor.

9.5 Memory map and registers description

9.5.1 Module memory map

Table 68 shows the INTC memory map.

Table 68. INTC memory map

Offset from INTC_BASE 0xFFF4_8000	Register	Location
0x0000	*INTC Module Configuration Register (INTC_MCR)*	*on page 9-213*
0x0004	Reserved	
0x0008	*INTC Current Priority Register (INTC_CPR)*	*on page 9-213*
0x000C	Reserved	
0x0010	*INTC Interrupt Acknowledge Register(INTC_IACKR)*	*on page 9-215*
0x0014	Reserved	
0x0018	*INTC End-of-Interrupt Register (INTC_EOIR)*	*on page 9-216*
0x001C	Reserved	
0x0020–0x0027	*INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)*	*on page 9-216*
0x0028– 0x003C	Reserved	
0x0040–0x011C	*INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)*(1)	*on page 9-218*
0x0120–0x3FFF	Reserved	

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled as Reserved in *Table 75*.

9.5.2 Registers description

With exception of the INTC_SSCI*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_SSCI*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 INTC_IACKR are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_SSCIR0_3–INTC_SSCIR4_7 or INTC_EOIR does not affect the operation of the write.

INTC registers are accessible only when the core is in supervisor mode (see *Section 15.4.3, “ECSM_reg_protection*).

INTC Module Configuration Register (INTC_MCR)

The module configuration register configures options of the INTC.

Figure 78. INTC Module Configuration Register (INTC_MCR)

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	0	0	0	0	HVEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 69. INTC_MCR field descriptions

Field	Description
26 VTES	Vector table entry size Controls the number of 0s to the right of INTVEC in *Section , "INTC Interrupt Acknowledge Register(INTC_IACKR).* If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of right most 0s will determine the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes 1 8 bytes
31 HVEN	Hardware vector enable Controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 9.4, "Modes of operation,* for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

INTC Current Priority Register (INTC_CPR)

Figure 79. INTC Current Priority Register (INTC_CPR)

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

Table 70. INTC_CPR field descriptions

Field	Description
28–31 PRI[0:3]	Priority PRI is the priority of the currently executing ISR according to the following: 1111 Priority 15—highest priority 1110 Priority 14 1101 Priority 13 1100 Priority 12 1011 Priority 11 1010 Priority 10 1001 Priority 9 1000 Priority 8 0111 Priority 7 0110 Priority 6 0101 Priority 5 0100 Priority 4 0011 Priority 3 0010 Priority 2 0001 Priority 1 0000 Priority 0—lowest priority

The INTC_CPR masks any peripheral or software settable interrupt request set at the same or lower priority as the current value of the INTC_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC_IACKR) 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 end-of-interrupt register (INTC_EOIR) is written, the LIFO is popped into the INTC_CPR's PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 9.7.5, "Priority ceiling protocol.*

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 the resource. Refer to Section , "Ensuring coherency, for example code to ensure coherency.

INTC Interrupt Acknowledge Register(INTC_IACKR)

Figure 80. INTC Interrupt Acknowledge Register (INTC_IACKR)

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	VTBA (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 5 bits)					INTVEC[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. When the VTES bit in INTC_MCR is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a '0'. VTBA is narrowed to 20 bits in width.

Table 71. INTC_IACKR field descriptions

Field	Description
0–20 or 0–19 VTBA	Vector Table Base Address Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the leftmost 20 bits when the VTES bit in INTC_MCR is asserted.
21–29 or 20–28 INTVEC	Interrupt Vector It is the vector of the peripheral or software configurable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. **Note:** If INTC_MCR[VTES] = 1, then the INTVEC field is shifted left one position to bits 20–28. VTBA is then shortened by one bit to bits 0–19.

The interrupt acknowledge register 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.

In software vector mode, the INTC_IACKR has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR does not have side effects in hardware vector mode.

INTC End-of-Interrupt Register (INTC_EOIR)

Figure 81. INTC End-of-Interrupt Register (INTC_EOIR)

Address Base + 0x0018 Access: 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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC_CPR. An exception to this behavior is described in *Section , "Hardware vector mode.* The values and size of data written to the INTC_EOIR are ignored. The values and sizes written to this register neither update the INTC_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR.

Reading the INTC_EOIR has no effect on the LIFO.

INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

Figure 82. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

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							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	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							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 83. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

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							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	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							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 72. INTC_SSCIR[0:7] field descriptions

Field	Description
6, 14, 22, 30 SET[0:7]	Set Flag Bits Writing a ‘1’ sets the corresponding CLR*x* bit. Writing a ‘0’ has no effect. Each SET*x* always will be read as a ‘0’.
7, 15, 23, 31 CLR[0:7]	Clear Flag Bits CLR*x* 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 Interrupt request not pending within INTC 1 Interrupt request pending within INTC

The software set/clear interrupt registers support the setting or clearing of software configurable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Excepting 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 like a peripheral interrupt request. Writing a ‘1’ to SET*x* will leave SET*x* unchanged at 0 but sets 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 simultaneously to a pair of SET*x* and CLR*x* bits, CLR*x* will be asserted, regardless of whether CLR*x* was asserted before the write.

INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)

Figure 84. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

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 85. INTC Priority Select Register 220–221 (INTC_PSR[220:221])

Address Base + 0x011C 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	PRI220				0	0	0	0	PRI221			
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

Table 73. INTC_PSR0_3–INTC_PSR220–221 field descriptions

Field	Description
4–7, 12–15, 20–23, 28–31 PRI[0:3]–PRI220:221	Priority Select PRI*x* selects the priority for interrupt requests. Refer to *Section 9.6, "Functional description.*

Table 74. INTC Priority Select Register address offsets

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR0_3	0x0040	INTC_PSR112_115	0x00B0
INTC_PSR4_7	0x0044	INTC_PSR116_119	0x00B4
INTC_PSR8_11	0x0048	INTC_PSR120_123	0x00B8
INTC_PSR12_15	0x004C	INTC_PSR124_127	0x00BC
INTC_PSR16_19	0x0050	INTC_PSR128_131	0x00C0
INTC_PSR20_23	0x0054	INTC_PSR132_135	0x00C4
INTC_PSR24_27	0x0058	INTC_PSR136_139	0x00C8

Table 74. INTC Priority Select Register address offsets (continued)

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR28_31	0x005C	INTC_PSR140_143	0x00CC
INTC_PSR32_35	0x0060	INTC_PSR144_147	0x00D0
INTC_PSR36_39	0x0064	INTC_PSR148_151	0x00D4
INTC_PSR40_43	0x0068	INTC_PSR152_155	0x00D8
INTC_PSR44_47	0x006C	INTC_PSR156_159	0x00DC
INTC_PSR48_51	0x0070	INTC_PSR160_163	0x00E0
INTC_PSR52_55	0x0074	INTC_PSR164_167	0x00E4
INTC_PSR56_59	0x0078	INTC_PSR168_171	0x00E8
F60_63	0x007C	INTC_PSR172_175	0x00EC
INTC_PSR64_67	0x0080	INTC_PSR176_179	0x00F0
INTC_PSR68_71	0x0084	INTC_PSR180_183	0x00F4
INTC_PSR72_75	0x0088	INTC_PSR184_187	0x00F8
INTC_PSR76_79	0x008C	INTC_PSR188_191	0x00FC
INTC_PSR80_83	0x0090	INTC_PSR192_195	0x0100
INTC_PSR84_87	0x0094	INTC_PSR196_199	0x0104
INTC_PSR88_91	0x0098	INTC_PSR200_203	0x0108
INTC_PSR92_95	0x009C	INTC_PSR204_207	0x010C
INTC_PSR96_99	0x00A0	INTC_PSR208_211	0x0110
INTC_PSR100_103	0x00A4	INTC_PSR212_215	0x0114
INTC_PSR104_107	0x00A8	INTC_PSR216_219	0x0118
INTC_PSR108_111	0x00AC	INTC_PSR220_221	0x011C

9.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

Note: The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRIn value in INTC_PSR0–INTC_PSR221 is higher than the PRI value in INTC_CPR, 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 will be updated with the corresponding PRIn value in INTC_PSRn. 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.

Table 75. Interrupt vector table

IRQ #	Offset	Interrupt	Module
On-Platform Peripherals			
Software Interrupts			
0	0x0800	Software configurable flag 0	Software
1	0x0804	Software configurable flag 1	Software
2	0x0808	Software configurable flag 2	Software
3	0x080C	Software configurable flag 3	Software
4	0x0810	Software configurable flag 4	Software
5	0x0814	Software configurable flag 5	Software
6	0x0818	Software configurable flag 6	Software
7	0x081C	Software configurable flag 7	Software
8	0x0820	Reserved	
ECSM			
9	0x0824	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
DMA2x			
10	0x0828	Combined Error	DMA2x
11	0x082C	Channel 0	DMA2x

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
12	0x0830	Channel 1	DMA2x
13	0x0834	Channel 2	DMA2x
14	0x0838	Channel 3	DMA2x
15	0x083C	Channel 4	DMA2x
16	0x0840	Channel 5	DMA2x
17	0x0844	Channel 6	DMA2x
18	0x0848	Channel 7	DMA2x
19	0x084C	Channel 8	DMA2x
20	0x0850	Channel 9	DMA2x
21	0x0854	Channel 10	DMA2x
22	0x0858	Channel 11	DMA2x
23	0x085C	Channel 12	DMA2x
24	0x0860	Channel 13	DMA2x
25	0x0864	Channel 14	DMA2x
26	0x0868	Channel 15	DMA2x
27	0x086C	Reserved	
SWT			
28	0x0870	Timeout	Software Watchdog (SWT)
29	0x0874	Reserved	
STM			
30	0x0878	Match on channel 0	STM
31	0x087C	Match on channel 1	STM
32	0x0880	Match on channel 2	STM
33	0x0884	Match on channel 3	STM
34	0x0888	Reserved	
ECSM			
35	0x088C	ECC_DBD_PlatformFlash | ECC_DBD_PlatformRAM	ECSM
36	0x0890	ECC_SBC_PlatformFlash | ECC_SBC_PlatformRAM	ECSM
37	0x0894	Reserved	
38	0x0898	Reserved	
39	0x089C	Reserved	
40	0x08A0	Reserved	

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
SIUL			
41	0x08A4	SIU External IRQ_0	SIUL
42	0x08A8	SIU External IRQ_1	SIUL
43	0x08AC	SIU External IRQ_2	SIUL
44	0x08B0	SIU External IRQ_3	SIUL
45	0x08B4	Reserved	
46	0x08B8	Reserved	
47	0x08BC	Reserved	
48	0x08C0	Reserved	
49	0x08C4	Reserved	
50	0x08C8	Reserved	
MC_ME			
51	0x08CC	Safe Mode Interrupt	Mode Entry module (MC_ME)
52	0x08D0	Mode Transition Interrupt	Mode Entry module (MC_ME)
53	0x08D4	Invalid Mode Interrupt	Mode Entry module (MC_ME)
54	0x08D8	Invalid Mode Configuration	Mode Entry module (MC_ME)
55	0x08DC	Reserved	
MC_RGM			
56	0x08E0	Functional and destructive reset alternate event interrupt	Reset Generation Module (MC_RGM)
XOSC			
57	0x08E4	XOSC counter expired	XOSC
PIT			
58	0x08E8	Reserved	
59	0x08EC	PITimer Channel 0	PIT
60	0x08F0	PITimer Channel 1	PIT
61	0x08F4	PITimer Channel 2	PIT
ADC0			
62	0x08F8	ADC_EOC	ADC_0
63	0x08FC	ADC_ER	ADC_0
64	0x0900	ADC_WD	ADC_0
FlexCAN0			
65	0x0904	FLEXCAN_ESR[ERR_INT]	FlexCAN_0

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
66	0x0908	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_0
67	0x090C	FLEXCAN_ESR_WAK	FlexCAN_0
68	0x0910	FLEXCAN_BUF_00_03	FlexCAN_0
69	0x0914	FLEXCAN_BUF_04_07	FlexCAN_0
70	0x0918	FLEXCAN_BUF_08_11	FlexCAN_0
71	0x091C	FLEXCAN_BUF_12_15	FlexCAN_0
72	0x0920	FLEXCAN_BUF_16_31	FlexCAN_0
73	0x0924	Reserved	
DSPI0			
74	0x0928	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_0
75	0x092C	DSPI_SR[EOQF]	DSPI_0
76	0x0930	DSPI_SR[TFFF]	DSPI_0
77	0x0934	DSPI_SR[TCF]	DSPI_0
78	0x0938	DSPI_SR[RFDF]	DSPI_0
LINFlex0			
79	0x093C	LINFlex_RXI	LINFlex_0
80	0x0940	LINFlex_TXI	LINFlex_0
81	0x0944	LINFlex_ERR	LINFlex_0
82	0x0948	Reserved	
83	0x094C	Reserved	
84	0x0950	Reserved	
FlexCAN1			
85	0x0954	FLEXCAN_ESR[ERR_INT]	FlexCAN_1
86	0x0958	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_1
87	0x095C	FLEXCAN_ESR_WAK	FlexCAN_1
88	0x0960	FLEXCAN_BUF_00_03	FlexCAN_1
89	0x0964	FLEXCAN_BUF_04_07	FlexCAN_1
90	0x0968	FLEXCAN_BUF_08_11	FlexCAN_1
91	0x096C	FLEXCAN_BUF_12_15	FlexCAN_1
92	0x0970	FLEXCAN_BUF_16_31	FlexCAN_1

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
93	0x0974	Reserved	
DSPI1			
94	0x0978	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_1
95	0x097C	DSPI_SR[EOQF]	DSPI_1
96	0x0980	DSPI_SR[TFFF]	DSPI_1
97	0x0984	DSPI_SR[TCF]	DSPI_1
98	0x0988	DSPI_SR[RFDF]	DSPI_1
LINFlex1			
99	0x098C	LINFlex_RXI	LINFlex_1
100	0x0990	LINFlex_TXI	LINFlex_1
101	0x0994	LINFlex_ERR	LINFlex_1
102	0x0998	Reserved	
103	0x099C	Reserved	
104	0x09A0	Reserved	
105	0x09A4	Reserved	
106	0x09A8	Reserved	
107	0x09AC	Reserved	
108	0x09B0	Reserved	
109	0x09B4	Reserved	
110	0x09B8	Reserved	
111	0x09BC	Reserved	
112	0x09C0	Reserved	
113	0x09C4	Reserved	
DSPI2			
114	0x09C8	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_2
115	0x09CC	DSPI_SR[EOQF]	DSPI_2
116	0x09D0	DSPI_SR[TFFF]	DSPI_2
117	0x09D4	DSPI_SR[TCF]	DSPI_2
118	0x09D8	DSPI_SR[RFDF]	DSPI_2
119	0x09DC	Reserved	
120	0x09E0	Reserved	
121	0x09E4	Reserved	

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
122	0x09E8	Reserved	
123	0x09EC	Reserved	
124	0x09F0	Reserved	
125	0x09F4	Reserved	
126	0x09F8	Reserved	
PIT			
127	0x09FC	PITimer Channel 3	PIT
128	0x0A00	Reserved	
129	0x0A04	Reserved	
130	0x0A08	Reserved	
131	0x0A0C	Reserved	
132	0x0A10	Reserved	
133	0x0A14	Reserved	
134	0x0A18	Reserved	
135	0x0A1C	Reserved	
136	0x0A20	Reserved	
137	0x0A24	Reserved	
138	0x0A28	Reserved	
139	0x0A2C	Reserved	
140	0x0A30	Reserved	
141	0x0A34	Reserved	
142	0x0A38	Reserved	
143	0x0A3C	Reserved	
144	0x0A40	Reserved	
145	0x0A44	Reserved	
146	0x0A48	Reserved	
147	0x0A4C	Reserved	
148	0x0A50	Reserved	
149	0x0A54	Reserved	
150	0x0A58	Reserved	
151	0x0A5C	Reserved	
152	0x0A60	Reserved	
153	0x0A64	Reserved	
154	0x0A68	Reserved	

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
155	0x0A6C	Reserved	
156	0x0A70	Reserved	
eTimer			
157	0x0A74	TC0IR	eTimer_0
158	0x0A78	TC1IR	eTimer_0
159	0x0A7C	TC2IR	eTimer_0
160	0x0A80	TC3IR	eTimer_0
161	0x0A84	TC4IR	eTimer_0
162	0x0A88	TC5IR	eTimer_0
163	0x0A8C	Reserved	
164	0x0A90	Reserved	
165	0x0A94	WTIF	eTimer_0
166	0x0A98	Reserved	
167	0x0A9C	RCF	eTimer_0
168	0x0AA0	Reserved	
169	0x0AA4	Reserved	
170	0x0AA8	Reserved	
171	0x0AAC	Reserved	
172	0x0AB0	Reserved	
173	0x0AB4	Reserved	
174	0x0AB8	Reserved	
175	0x0ABC	Reserved	
176	0x0AC0	Reserved	
177	0x0AC4	Reserved	
178	0x0AC8	Reserved	
FlexPWM			
179	0x0ACC	RF0	FlexPWM_0
180	0x0AD0	COF0	FlexPWM_0
181	0x0AD4	Reserved	
182	0x0AD8	RF1	FlexPWM_0
183	0x0ADC	COF1	FlexPWM_0
184	0x0AE0	Reserved	
185	0x0AE4	RF2	FlexPWM_0
186	0x0AE8	COF2	FlexPWM_0

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
187	0x0AEC	Reserved	
188	0x0AF0	RF3	FlexPWM_0
189	0x0AF4	COF3	FlexPWM_0
190	0x0AF8	Reserved	
191	0x0AFC	FFLAG	FlexPWM_0
192	0x0B00	REF	FlexPWM_0
CTU			
193	0x0B04	MRS_I	CTU_0
194	0x0B08	T0_I	CTU_0
195	0x0B0C	T1_I	CTU_0
196	0x0B10	T2_I	CTU_0
197	0x0B14	T3_I	CTU_0
198	0x0B18	T4_I	CTU_0
199	0x0B1C	T5_I	CTU_0
200	0x0B20	T6_I	CTU_0
201	0x0B24	T7_I	CTU_0
202	0x0B28	FIFO1_I	CTU_0
203	0x0B2C	FIFO2_I	CTU_0
204	0x0B30	FIFO3_I	CTU_0
205	0x0B34	FIFO4_I	CTU_0
206	0x0B38	ADC_I	CTU_0
207	0x0B3C	ERR_I	CTU_0
SafetyPort			
208	0x0B40	FLEXCAN_ESR[ERR_INT]	SafetyPort (FlexCAN)
209	0x0B44	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	SafetyPort (FlexCAN)
210	0x0B48	FLEXCAN_ESR_WAK	SafetyPort (FlexCAN)
211	0x0B4C	FLEXCAN_BUF_0_3	SafetyPort (FlexCAN)
212	0x0B50	FLEXCAN_BUF_4_7	SafetyPort (FlexCAN)
213	0x0B54	FLEXCAN_BUF_8_11	SafetyPort (FlexCAN)
214	0x0B58	FLEXCAN_BUF_12_15	SafetyPort (FlexCAN)
215	0x0B5C	FLEXCAN_BUF_16_31	SafetyPort (FlexCAN)
216	0x0B60	Reserved	
217	0x0B64	Reserved	

Table 75. Interrupt vector table (continued)

IRQ #	Offset	Interrupt	Module
218	0x0B68	Reserved	
219	0x0B6C	Reserved	
220	0x0B70	Reserved	
221	0x0B74	Reserved	

9.6.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the 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.

External interrupts are handled by the SIU (see *Section 11.6.4, "External interrupts*).

Software configurable interrupt requests

An interrupt request is triggered by software by writing a '1' to a SET*x* bit in *INTC_SSCIR0_3–INTC_SSCIR4_7*. This write sets the corresponding flag bit, CLR*x*, 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.

Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see *Table 67*).

9.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221). The result is compared to PRI in the associated *INTC_CPR*. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 77* compare the priority of the asserted interrupt requests to the current priority. If the priority of

any asserted peripheral or software configurable 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 INTC interrupt acknowledge register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

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 configurable. 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.

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, only the one with the lowest vector passes 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 configurable interrupt requests.

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.

Priority comparator subblock

The priority comparator submodule compares the highest priority output from the priority arbitrator submodule with PRI in INTC_CPR. If the priority comparator submodule detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC_CPR or the PRI value in INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority that will be written to PRI in INTC_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n* are zero will not cause a preemption because their PRI*n* will not be higher than PRI in INTC_CPR.

Last-in first-out (LIFO)

The LIFO stores the preempted PRI values from the INTC_CPR. 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 INTC_CPR does not need to be loaded from the INTC_CPR 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 INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR 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 how 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.

9.6.3 Handshaking with processor

Software vector mode handshaking

This section describes handshaking in software vector mode.

Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode and the handshake near the end of the interrupt exception handler, is shown in *Figure 86*. The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in the associated *INTC_CPR*, it asserts the interrupt request to the processor. The INTVEC field in the associated *INTC_IACKR* 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 handshaking process is described in *Section , "Software vector mode. "*

End of interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR) must be written.When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

Note: To ensure proper operation across all eSys MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC_EOIR.

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 INTC_CPR 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.

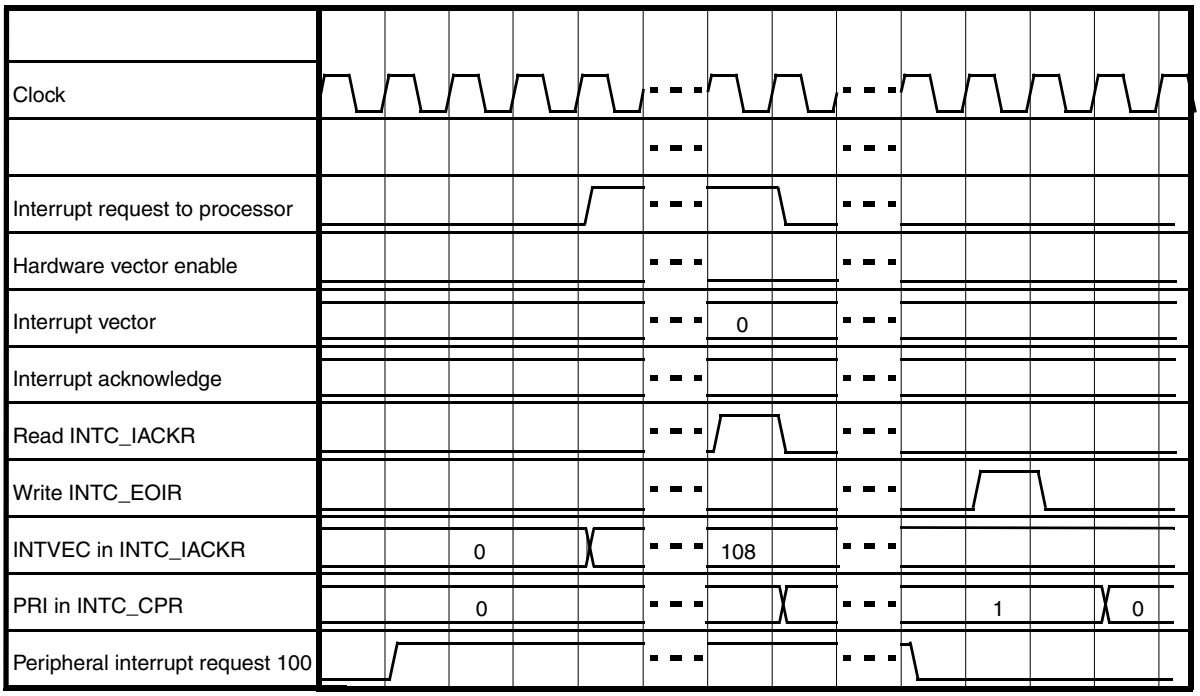


Figure 86. Software vector mode handshaking timing diagram

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 87*. 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 INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR 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 processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in" *Section , "Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. Refer to *Section , "End of interrupt exception handler*.

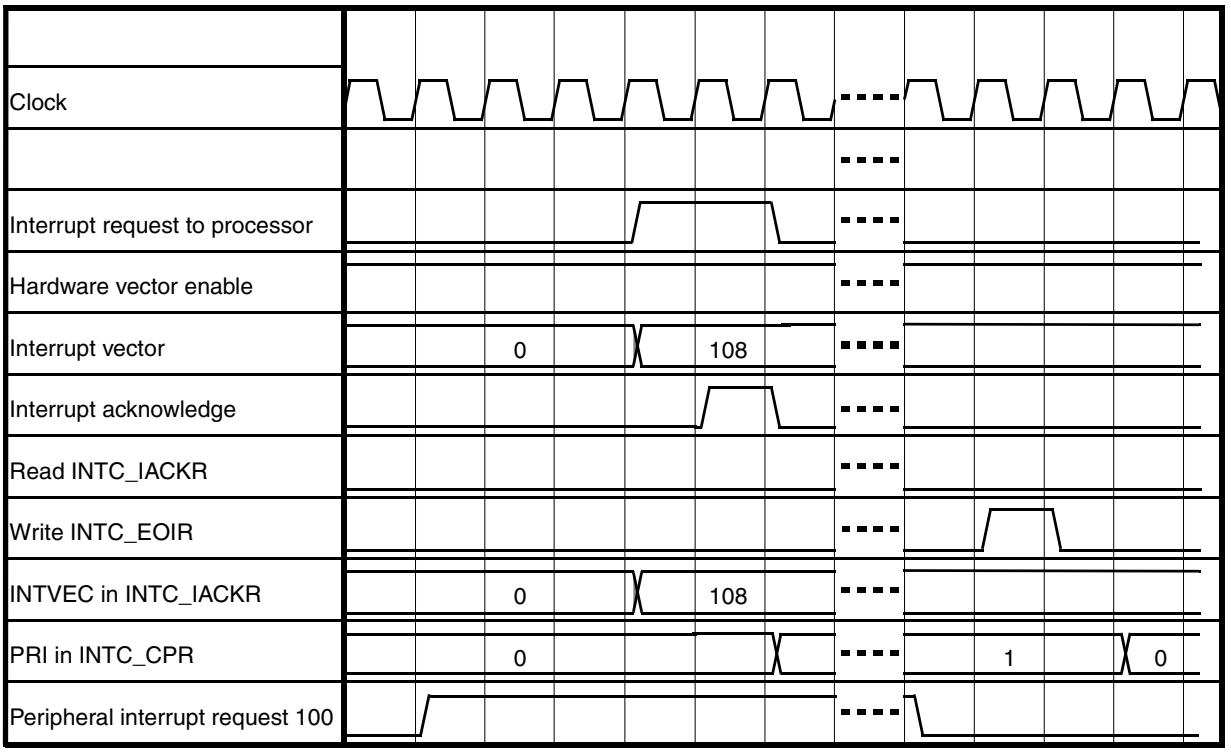


Figure 87. Hardware vector mode handshaking timing diagram

9.7 Initialization/application information

9.7.1 Initialization flow

After exiting reset, all of the PRI*n* fields in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221) will be zero, and PRI in INTC current priority register (INTC_CPR) will be 15. These reset values will prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:
interrupt_request_initialization:

```
interrupt_request_initialization:
configure VTES and HVEN in INTC_MCR
configure VTBA in INTC_IACKR
raise the PRIn fields in INTC_PSRn
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR to zero
enable processor recognition of interrupts
```

9.7.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

Software vector mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
lis   r3,INTC_IACKR@ha # form adjusted upper half of INTC_IACKR address
lwz   r3,INTC_IACKR@l(r3) # load INTC_IACKR, which clears request to
processor
lwz   r3,0x0(r3)      # load address of ISR from vector table
wrteei 1              # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr r3               # move INTC_IACKR contents into link register
blrl                  # branch to ISR; link register updated with epilog
                      # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar                  # ensure store to clear flag bit has completed
lis   r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li    r4,0x0          # form 0 to write to INTC_EOIR
wrteei 0              # disable processor recognition of interrupts
stw   r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

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 that drives interrupt request to INTC

blr # return to epilog
```

Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each

interrupt_exception_handler*x* only has space for four instructions, and therefore a branch to interrupt_exception_handler_continued*x* is needed.

```
interrupt_exception_handlerx:
b interrupt_exception_handler_continuedx# 4 instructions available, branch
to continue
interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei 1        # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl    ISRx      # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar            # ensure store to clear flag bit has completed
lis   r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li    r4,0x0    # form 0 to write to INTC_EOIR
wrteei 0        # disable processor recognition of interrupts
stw   r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi

ISRx:
code to service the interrupt event
code to clear flag bit that drives interrupt request to INTC
blr             # branch to epilog
```

9.7.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR) having a value of 0. The RTOS will execute 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 priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR 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 while the shared resource is being accessed.

An ISR whose PRI*n* in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221) has a value of 0 will not cause an interrupt request to the 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.

9.7.4 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 configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, 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 configurable interrupt requests asserted.

The example in *Table 76* shows the order of execution of both ISRs with different priorities and the same priority

Table 76. 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	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3

Table 76. Order of ISR execution example (continued)

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	
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

9.7.5 Priority ceiling protocol

Elevating priority

The PRI field in *INTC_CPR* is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol 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 share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR can be lowered. If they do not raise their priority, 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 eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. 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.

Ensuring coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the INTC_CPR. The instruction following this store is a store to a value in a shared coherent data block. Either immediately 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 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 corruption of a coherent data block, modifications to PRI in INTC_CPR can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

9.7.6 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 may be 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 included, regardless of the number of ISRs.

Reducing the number of priorities 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 do not need to use the PCP to access the shared resource.

9.7.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and they may also be used by processors to interrupt other processors in a multiple processor system.

Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221), which becomes the PRI value in *INTC_CPR* with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent 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 what the later portion of the ISR needs. This preemptive scheduling inefficiency 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 *INTC_SSCIR0_3–INTC_SSCIR4_7*. Writing a '1' to SET*x* causes a software configurable interrupt request. This software configurable interrupt request will usually have a lower PRI*x*

value in the INTC_PSR*x*_*x* and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

Scheduling an ISR on another processor

Because 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 the processor executing the software configurable ISR has not completed the work before asking it to again execute the 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. To do this, the first processor writes a '1' to a SET*x* bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLR*x* bit and then writes 1 to a SET*x* bit on the first processor, informing it that it can now access the block of data.

9.7.8 Lowering priority within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see *Section , "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 INTC_CPR within an ISR to below the ISR's corresponding PRI value in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221) allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

9.7.9 Negating an interrupt request outside of its ISR

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 their corresponding interrupt requests. 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.

Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

Proper setting of interrupt request priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software configurable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits 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 *INTC_SSCIR0_3–INTC_SSCIR4_7* as the clearing of the flag bit that caused the present ISR to be executed (see *Section , "End of interrupt exception handler*).

A flag bit whose enable bit or mask bit negates 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*.

9.7.10 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 wants to read the contents, such as in debug mode, they are not memory mapped. The contents can be read by popping the LIFO and reading the PRI field in either *INTC_CPR*. The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, 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
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

10 System Status and Configuration Module (SSCM)

10.1 Introduction

10.1.1 Overview

The System Status and Configuration Module (SSCM), pictured in *Figure 88*, provides central device functionality.

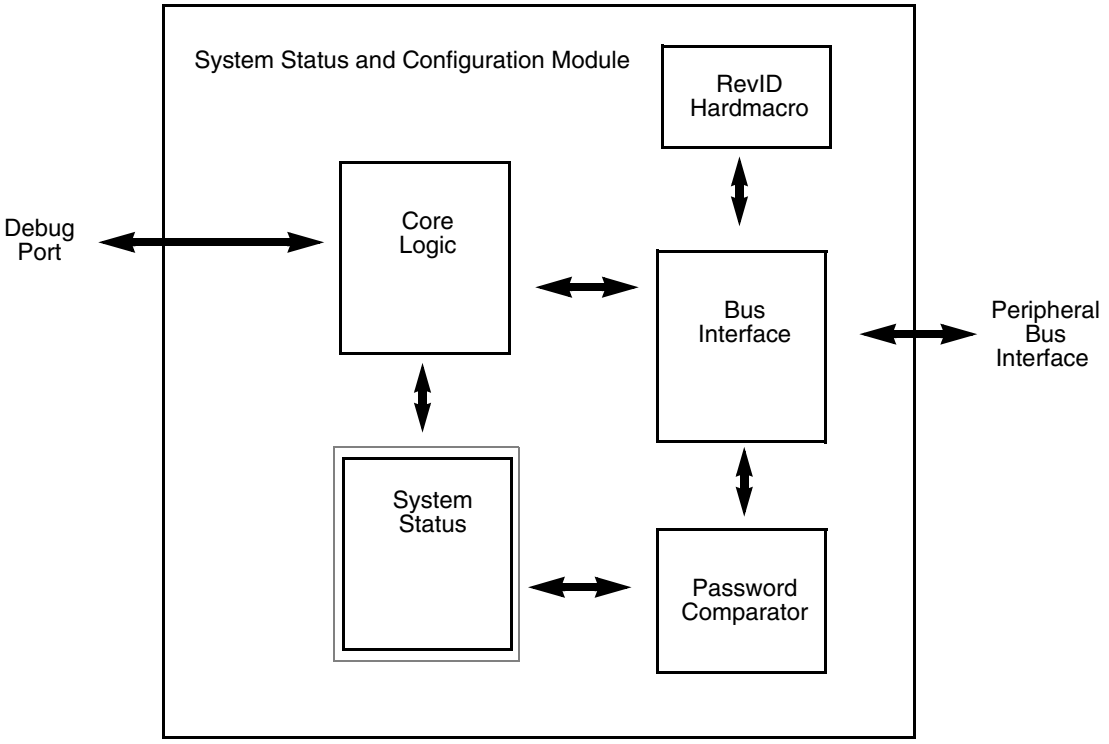


Figure 88. SSCM block diagram

10.1.2 Features

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search Code Flash for bootable sector
  - DMA status
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

10.1.3 Modes of operation

The SSCM operates identically in all system modes.

10.2 Memory map and register description

This section provides a detailed description of all memory-mapped registers in the SSCM.

10.2.1 Memory map

Table 77 shows the memory map for the SSCM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSCM.

Table 77. SSCM memory map

Offset from SSCM_BASE (0xC3FD_8000)	Register	Location
0x0000	STATUS—System Status register	*on page 10-242*
0x0002	MEMCONFIG—System Memory Configuration register	*on page 10-243*
0x0004	Reserved (Reads/Writes have no effect)	
0x0006	ERROR—Error Configuration register	*on page 10-244*
0x0008	DEBUGPORT—Debug Status Port register	*on page 10-245*
0x000A	Reserved (Reads/Writes have no effect)	
0x000C	PWCMPH—Password Comparison High Word register	*on page 10-246*
0x0010	PWCMPL—Password Comparison Low Word register	*on page 10-246*
0x0014–0x3FFF	Reserved	

All registers are accessible via 8, 16 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, the MEMCONFIG register is accessible by a 16-bit read/write to address Base + 0x0002, but performing a 16-bit access to Base + 0x0003 is illegal.

10.2.2 Register description

Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of the registers is MSB = 0, however the numbering of the internal fields is LSB = 0, for example, register SSCM_STATUS[8] = BMODE[2].

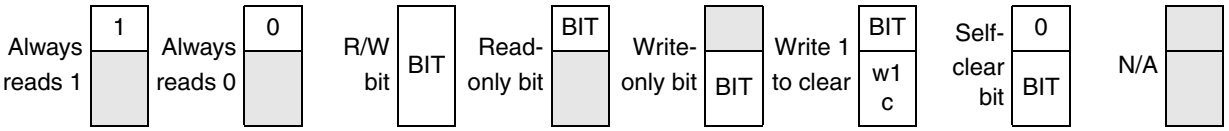


Figure 89. Key to register fields

System Status register (STATUS)

The system status register is a read-only register that reflects the current state of the system.

Figure 90. Status (STATUS) register

Address: Base + 0x0000 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	NXEN	PUB	SEC	0	BMODE[2:0]			0	ABD	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Table 78. STATUS allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8, or 0xC).

Table 79. STATUS field descriptions

Field	Description
NXEN	Nexus enabled
PUB	Public Serial Access Status This bit indicates whether serial boot mode with public password is allowed. 1: Serial boot mode with public password allowed 0: Serial boot mode with private Flash password allowed, provided the key has not been swallowed
SEC	Security Status This bit reflects the current security state of the Flash. 1: Flash secured 0: Flash not secured
BMODE [2:0]	Device Boot Mode 000: TestFlash 001: CAN Serial Boot Loader 010: SCI Serial Boot Loader 011: Single Chip 100–111: Reserved This field is updated only during reset.
ABD	Autobaud Indicates that autobaud detection is active when in SCI or CAN serial boot loader mode. No meaning in other modes.

System Memory Configuration register (MEMCONFIG)

The system memory configuration register is a read-only register that reflects the memory configuration of the system.

Figure 91. System memory configuration (MEMCONFIG) register

Address: Base + 0x0002 Access: 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	IVLD	0	0	0	0	DVLD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 80. MEMCONFIG field descriptions

Field	Description
IVLD	CFlash Valid This bit identifies whether or not the on-chip CFlash is accessible in the system memory map. The Flash may not be accessible due to security limitations. 1: CFlash accessible 0: CFlash not accessible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.
DVLD	DFlash Valid This bit identifies whether or not the on-chip DFlash is visible in the system memory map. The Flash may not be accessible due to security limitations. 1: DFlash visible 0: DFlash not visible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.

Table 81. MEMCONFIG allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed (also reads STATUS register)
Write	Not allowed	Not allowed	Not allowed

Error Configuration (ERROR) register

The Error Configuration register is a read-write register that controls the error handling of the system.

Figure 92. Error Configuration (ERROR) register

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

Table 82. 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. 1: Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to non-existing peripherals do not 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. 1: Illegal accesses to peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception.

Note: Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (i.e., at the PRIDGE level). In this case, the PAE and RAE register bits will have no effect on the abort.

Table 83. ERROR allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed
Write	Allowed	Allowed	Not allowed

Debug Status Port (DEBUGPORT) register

The Debug Status Port register provides debug data on a set of pins.

Figure 93. Debug Status Port (DEBUGPORT) register

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[2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 84. DEBUGPORT field descriptions

Field	Description
13-15 DEBUG_MODE[2:0]	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 85* describes the functionality of the Debug Status Port in each mode.

Table 85. 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 86. DEBUGPORT allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Not allowed
Write	Allowed	Allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

Password comparison registers

These registers allow to unsecure the device, if the correct password is known.

Figure 94. Password Comparison Register High Word (PWCMPH) register

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	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 95. Password Comparison Register Low Word (PWCMPL) 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	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

Table 87. PWCMPH/L field descriptions

Field	Description
PWD_HI[31:0]	Upper 32 bits of the password
PWD_LO[31:0]	Lower 32 bits of the password

Table 88. PWCMPH/L allowed register accesses

Access type	Access width		
	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 (i.e., 0x0, 0x4, 0x8 or 0xC).

10.3 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 debug of the system.

10.4 Initialization/application information

10.4.1 Reset

The reset state of each individual bit is shown in *Section 10.2.2, "Register description*.

11 System Integration Unit Lite (SIUL)

11.1 Introduction

This chapter describes the System Integration Unit Lite (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 managing the external interrupts to the device.

11.2 Overview

The System Integration Unit Lite (SIUL) controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals and external interrupts with trigger event configuration. *Figure 96* is 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, you can detect the state of the associated pad 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 a method of checking if the written value appeared on the pad.

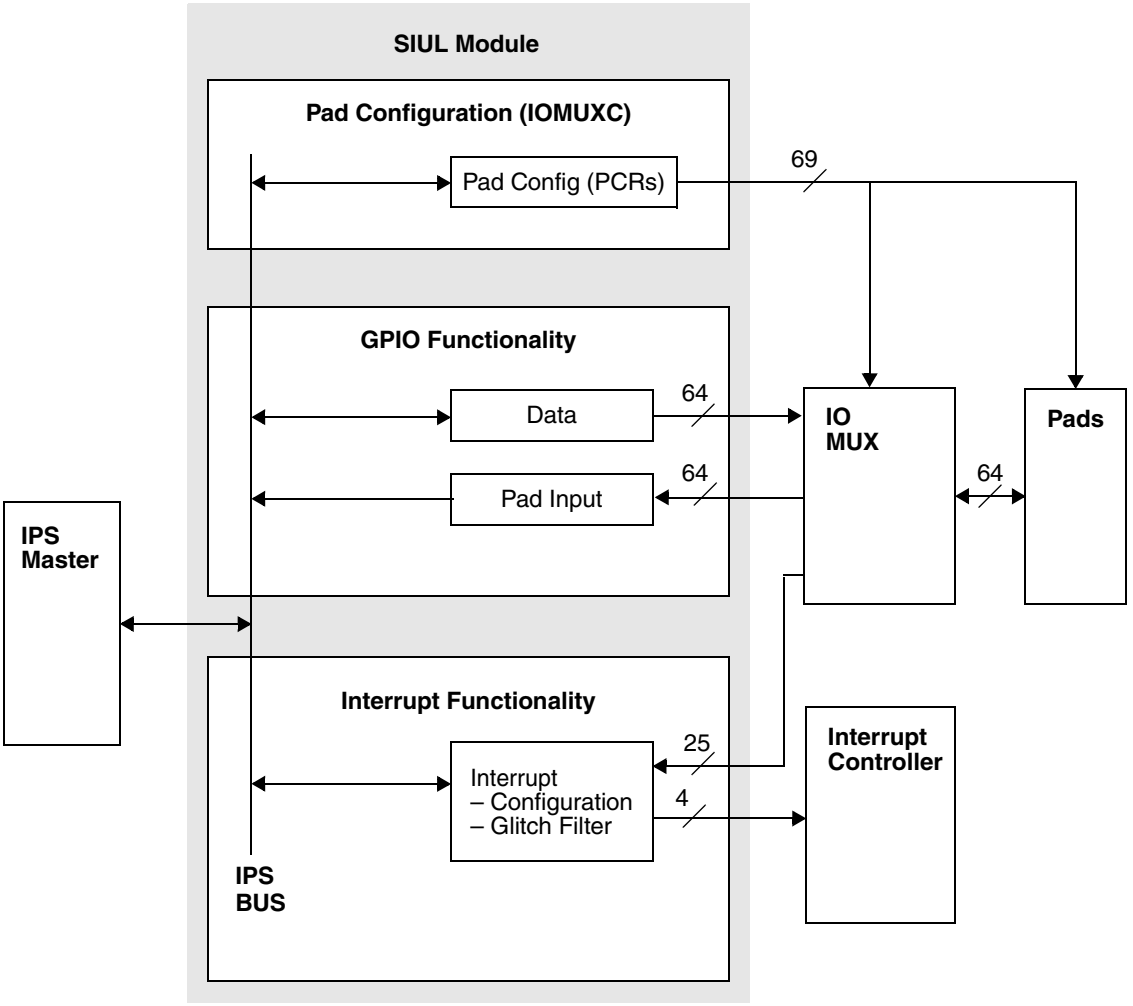


Figure 96. System Integration Unit Lite block diagram

11.3 Features

The System Integration Unit Lite provides these features:

- GPIO
  - GPIO function on up to 64 I/O pins
  - Dedicated input and output registers for each GPIO pin
- External interrupts
  - 4 system interrupt vectors for up to 25 interrupt sources
  - 25 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

11.3.1 Register protection

Most of the configuration registers of the System Integration Unit Lite are protected from accidental writes, see *Appendix A: Registers Under Protection*.

11.4 External signal description

The pad configuration allows flexible, centralized control of the pin electrical characteristics of the MCU with the GPIO control providing 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, will normally be the peripherals functions. The internal multiplexing allows user selection of the input to chip-level signal multiplexers. Each GPIO port communicates via 16 I/O channels. In order to use the pad as a GPIO, the corresponding Pad Configuration Registers (PCR[0:71]) for all pads used in the port must be configured as GPIO rather than as the alternate pad function.

Table 89 lists the external pins used by the SIUL.

Table 89. SIUL signal properties

GPIO category	Name	I/O direction	Function
System configuration	GPIO[0:19], GPIO[22], GPIO[35:62]	Input/Output	General-purpose input/output
	GPIO[23:34], GPIO[63], GPIO[65:66]	Input	Analog precise channel pins
External interrupt	EIRQ[0:24]	Input	Pins with External Interrupt Request functionality. Please refer to the signal description chapter of this reference manual for details.

11.4.1 Detailed signal descriptions

General-purpose I/O pins (GPIO[0:66])

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 , “GPIO Pad Data Output registers 0_3–68_71 (GPDO[0_3:68_71])* and *Section , “GPIO Pad Data Input registers 0_3–68_71 (GPDI[0_3:68_71])*.

External interrupt request input pins (EIRQ[0:24])

The EIRQ[0:24] are connected to the SIUL inputs. Rising or falling edge events are enabled by setting the corresponding bits in the “n” SIUL_IREER or the SIUL_IFEER register. See *Section , “Interrupt Rising-Edge Event Enable Register (IREER)* and *Section , “Interrupt Falling-Edge Event Enable Register (IFEER)*.

11.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

11.5.1 SIUL memory map

Table 90 lists the SIUL registers.

Table 90. SIUL memory map

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x0000–0x0003	Reserved	
0x0004	MCU ID Register #1 (MIDR1)	*on page 11-252*
0x0008	MCU ID Register #2 (MIDR2)	*on page 11-254*
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	*on page 11-255*
0x0018	Interrupt Request Enable Register (IRER)	*on page 11-255*
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	*on page 11-256*
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	*on page 11-256*
0x0030	Interrupt Filter Enable Register (IFER)	*on page 11-257*
0x0034–0x003F	Reserved	
0x0040–0x00CE	Pad Configuration Registers (PCR[0:71])	*on page 11-257*
0x00D0–0x04FF	Reserved	
0x0500–0x0520	Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])	*on page 11-259*
0x0524–0x05FF	Reserved	
0x0600–0x0644	GPIO Pad Data Output registers 0_3–68_71 (GPDO[0_3:68_71])	*on page 11-262*
0x0648–0x07FF	Reserved	
0x0800–0x0844	GPIO Pad Data Input registers 0_3–68_71 (GPDI[0_3:68_71])	*on page 11-262*
0x0848–0x0BFF	Reserved	
0x0C00–0x0C0C	Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])	*on page 11-263*
0x0C10–0x0C3F	Reserved	
0x0C40–0x0C4C	Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])	*on page 11-263*
0x0C50–0x0C7F	Reserved	
0x0C80–0x0C98	Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])	*on page 11-264*
0x0C9C–0x0FFF	Reserved	

Table 90. SIUL memory map (continued)

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x1000–0x1060	Interrupt Filter Maximum Counter registers 0–24 (IFMC[0:24])	*on page 11-265*
0x1064–0x107C	Reserved	
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	*on page 11-266*
0x1084–0x3FFF	Reserved	

Note: A transfer error will be issued when trying to access completely reserved register space.

11.5.2 Register description

This section describes in address order all the SIUL registers. Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of register is MSB = 0, however the numbering of internal field is LSB = 0, for example PARTNUM[5] = MIDR1[10].

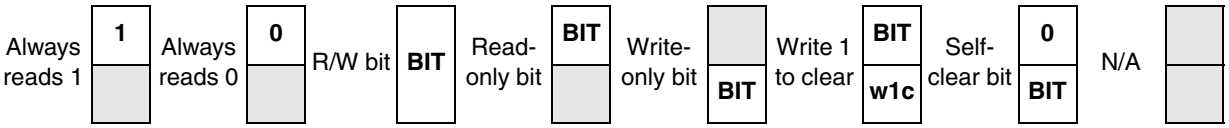


Figure 97. Key to register fields

MCU ID Register #1 (MIDR1)

This register contains the part number and the package ID of the device.

Figure 98. MCU ID Register #1 (MIDR1)

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	0	0	0	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG[4:0]					0	0	MAJOR_MASK[3:0](1)				MINOR_MASK[3:0]1			
W																
Reset	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

1. See *Table 91*.

Table 91. MIDR1 field descriptions

Field	Description
PARTNUM[15:0]	MCU Part Number Device part number of the MCU. 0101_0110_0000_0001: 192 KB 0101_0110_0000_0010: 256 KB 0101_0110_0000_0011: 320/384 KB For the full part number this field needs to be combined with MIDR2.PARTNUM[23:16]
CSP	Always reads back 0
PKG[4:0]	Package Settings Can by read by software to determine the package type that is used for the particular device: 00001: 64-pin LQFP 01001: 100-pin LQFP
MAJOR_MASK[3:0]	Major Mask Revision Counter starting at 0x0. Incremented each time a resynthesis is done.
MINOR_MASK[3:0]	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

MCU ID Register #2 (MIDR2)

This register contains additional configuration information about the device.

Figure 99. MCU ID Register #2 (MIDR2)

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[3:0]				0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	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	PARTNUM[23:16]								0	0	0	EE	0	0	0	0
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Table 92. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0: Reserved 1: ST
FLASH_SIZE[3:0]	Coarse granularity for Flash memory size Needs to be combined with FLASH_SIZE_2 to calculate the actual memory size. 0011: 192 KB 0100: 256 KB Other values are reserved.
PARTNUM[23:16]	ASCII character in MCU Part Number 0x50: P family (Steering)
EE	Data Flash present 0: No Data Flash present 1: Data Flash present

Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Figure 100. Interrupt Status Flag Register (ISR)

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[24:16]								
W								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															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 93. ISR field descriptions

Field	Description
EIF*n*	External Interrupt Status Flag *n* This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER*n*), EIF*n* causes an interrupt request. 0: No interrupt event has occurred on the pad. 1: An interrupt event as defined by IREER*n* and IFEER*n* has occurred.

Interrupt Request Enable Register (IRER)

This register enables the interrupt messaging to the interrupt controller.

Figure 101. Interrupt Request Enable Register (IRER)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R								IRE[24:16]								
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	IRE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 94. IRER field descriptions

Field	Description
IRE*n*	External Interrupt Request Enable *n* 0: Interrupt requests from the corresponding EIF*n* bit are disabled. 1: A set EIF*n* bit causes an interrupt request.

Interrupt Rising-Edge Event Enable Register (IREER)

This register allows rising-edge triggered events to be enabled on the corresponding interrupt pads.

Figure 102. Interrupt Rising-Edge Event Enable Register (IREER)

Address: Base + 0x0028 Access: User read/write

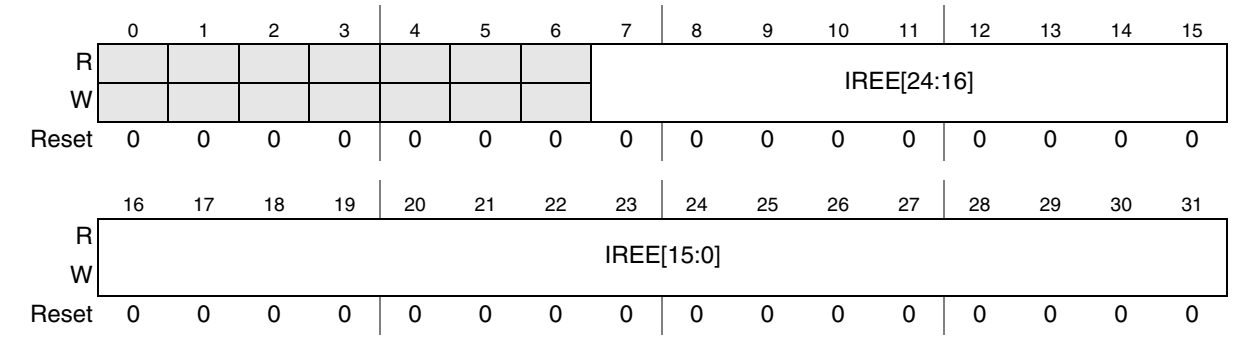


Table 95. IREER field descriptions

Field	Description
IREE*n*	Enable rising-edge events to cause the EIF*n* bit to be set. 0: Rising-edge event disabled 1: Rising-edge event enabled

Interrupt Falling-Edge Event Enable Register (IFEER)

This register allows falling-edge triggered events to be enabled on the corresponding interrupt pads.

Figure 103. Interrupt Falling-Edge Event Enable Register (IFEER)

Address: Base + 0x002C Access: User read/write

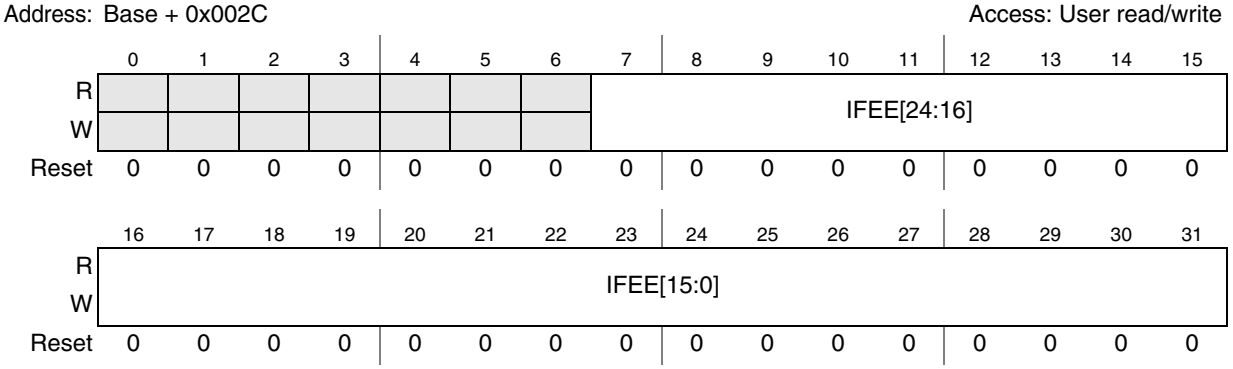


Table 96. IFEER field descriptions

Field	Description
IFEE*n*	Enable falling-edge events to cause the EIF*n* bit to be set. 0: Falling-edge event disabled 1: Falling-edge event enabled

Note: If both the IREER.IREE and IFEER.IFEE bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set.

Interrupt Filter Enable Register (IFER)

This register enables a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.

Figure 104. Interrupt Filter Enable Register (IFER)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R								IFE[24:16]								
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	IFE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 97. IFER field descriptions

Field	Description
IFE*n*	Enable digital glitch filter on the interrupt pad input. 0: Filter disabled 1:Filter enabled

Pad Configuration Registers (PCR[0:71])

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Figure 105. Pad Configuration Registers 0–71 (PCR[0:71])

Base + 0x0040 (PCR0)
Address: ... Access: User read/write
Base + 0x00CE (PCR71) 72 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC	0	PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. See *Table 99*.

Note: 16/32-bit access is supported for the PCR[0:71] registers.

Table 98. PCR[0:71] field descriptions

Field	Description
SMC	Safe Mode Control This bit supports the overriding of the automatic deactivation of the output buffer of the associated pad upon entering Safe mode of the device. 0: In Safe mode, output buffer of the pad disabled 1: In Safe mode, output buffer remains functional
APC	Analog Pad Control This bit enables the usage of 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 0 to 2 bits, depending on the number of output functions associated with this pad. 00: Alternative mode 0: GPIO 01: Alternative mode 1 (see *Chapter 3: Signal Description*) 10: Alternative mode 2 (see *Chapter 3: Signal Description*) 11: Alternative mode 3 (see *Chapter 3: Signal Description*) The number of bits in the PA bitfield depends of the number of actual alternate functions provided for each pad. Please see the *SPC560P40/34 Datasheet (SPC560P40/34)*.
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 disabled when PA = 00 1: Output buffer of the pad enabled when PA = 00
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0: Input buffer of the pad disabled 1: Input buffer of the pad 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 negated for the pad 1: Open drain enable signal asserted for the pad
SRC	Slew Rate Control 0: Slowest configuration 1: Fastest configuration
WPE	Weak Pull Up/Down Enable This bit controls whether the weak pull up/down devices are enabled/disabled for the pad connected to this signal. 0: Weak pull device enable signal negated for the pad 1: Weak pull device enable signal asserted for the pad
WPS	Weak Pull Up/Down Select This bit controls whether weak pull up or weak pull down devices are used for the pads connected to this signal when weak pull up/down devices are enabled. 0: Pull down enabled 1: Pull up enabled

Table 99. PCR[*n*] reset value exceptions

Field	Description
PCR[2] PCR[3] PCR[4]	These registers correspond to the ABS[0], ABS[1], and FAB boot pins, respectively. Their default state is input, pull enabled. Their reset value is 0x0102.
PCR[20]	This register corresponds to the TDO pin. Its default state is ALT1, slew rate = 1. Its reset value is 0x0604.
PCR[21]	This register corresponds to the TDI pin. Its default state is input, pull enabled, pull selected, slew enabled. So its reset value is 0x0107.
PCR[*n*]	For other PCR[*n*] registers, the reset value is 0x0000.

In addition to the bit map above, the following *Table 100 [PCR bit implementation by pad type]* describes the PCR depending on the pad type (refer to *Section 3.3.3: Pin multiplexing* for pad types description). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 100. PCR bit implementation by pad type

Pad type	PCR bit No.															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S, M, F (Pad with GPIO and digital alternate functionality)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS
I (Pad with GPIO and analog functionality)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS

Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])

The purpose of the PSMI[0_3:32_35] registers is to allow connecting a single input pad to one of several peripheral inputs. Thus, it is possible to define different pads to be possible inputs for a certain peripheral function.

Figure 106. Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])

Address: Base + 0x0500 (PSMI0_3)
Base + 0x0504 (PSMI4_7)
Base + 0x0508 (PSMI8_11)
Base + 0x050C (PSMI12_15)
Base + 0x0510 (PSMI16_19)
Base + 0x0514 (PSMI20_23)
Base + 0x0518 (PSMI024_27)
Base + 0x051C (PSMI28_31)
Base + 0x0520 (PSMI32_35)

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[3:0]				0	0	0	0	PADSEL1[3:0]			
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	PADSEL2[3:0]				0	0	0	0	PADSEL3[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 101. PSMI[0_3:32_35] field descriptions

Field	Description
PADSEL0–3 ... PADSEL32–35	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See *Table 102* Pad selection.

Table 102. Pad selection

Register	PADSEL	Module	Port	PADSEL[3:0] value(1)	Port name	LQFP pin	
						64-pin	100-pin
PSMI0_3(2)	PADSEL0	ctu0	EXT_IN	0000	C[13]	—	71
				0001	C[15]	—	85
	PADSEL1	dspi2	SCK	0000	A[0]	—	51
				0001	A[11]	53	82
	PADSEL2	dspi2	SIN	0000	A[2]	—	57
				0001	A[13]	61	95
	PADSEL3	dspi2	CS0	0000	A[3]	41	64
				0001	A[10]	52	81
PSMI4_7(3)	PADSEL0	—	—	—	—	—	—
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	eTimer0	ETC[4]	0000	A[4]	48	75
				0001	C[11]	33	55
				0010	B[14]	—	44

Table 102. Pad selection (continued)

Register	PADSEL	Module	Port	PADSEL[3:0] value(1)	Port name	LQFP pin	
						64-pin	100-pin
PSMI8_11[2]	PADSEL0	eTimer0	ETC[5]	0000	C[12]	34	56
				0001	B[8]	22	31
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	—	—	—	—	—	—
PSMI12_15[2]	PADSEL0	—	—	—	—	—	—
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	flexpwm0	EXT_SYNC	0000	C[13]	—	71
				0001	C[15]	—	85
PSMI16_19[3]	PADSEL0	flexpwm0	FAULT0	0000	A[9]	60	94
				0001	A[13]	61	95
	PADSEL1	flexpwm0	FAULT1	0000	C[10]	—	78
				0001	D[6]	—	23
	PADSEL2	—	—	—	—	—	—
	PADSEL3	—	—	—	—	—	—
PSMI20_23	PADSEL0	—	—	—	—	—	—
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	—	—	—	—	—	—
PSMI24_27	PADSEL0	—	—	—	—	—	—
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	—	—	—	—	—	—
PSMI28_31[2]	PADSEL0	—	—	—	—	—	—
	PADSEL1	—	—	—	—	—	—
	PADSEL2	—	—	—	—	—	—
	PADSEL3	LINflex0	RXD—Receive Data Input Line	0000	B[3]	—	80
				0001	B[7]	20	29
PSMI32_35[3]	PADSEL0	LINflex1	RXD—Receive Data Input Line	0000	B[13]	30	42
				0001	D[12]	45	70

1. Values not listed are reserved.
2. Writing to PADSELx[3:1] has no effect—a write to these three bits will return ‘0’.
3. Writing to PADSELx[3:2] has no effect—a write to these two bits will return ‘0’.

GPIO Pad Data Output registers 0_3–68_71 (GPDO[0_3:68_71])

These registers can be used to set or clear a single GPIO pad with a byte access.

Figure 107. Port GPIO Pad Data Output registers 0_3–68_71 (GPDO[0_3:68_71])

Base + 0x0600 (GPDO0_3)
Address: ... Access: User read/write
Base + 0x0644 (GPDO68_71) 18 registers

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

Table 103. GPDO[0_3:68_71] field descriptions

Field	Description
PDO[*n*]	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.

GPIO Pad Data Input registers 0_3–68_71 (GPDI[0_3:68_71])

These registers can be used to read the GPIO pad data with a byte access.

Figure 108. GPIO Pad Data Input registers 0_3–68_71 (GPDI[0_3:68_71])

Base + 0x0800 (GPDI0_3)
Address: ... Access: User read-only
Base + 0x0844 (GPDI68_71) 18 registers

	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

Table 104. GPDI[0_3:68_71] 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.

Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])

These registers set or clear the respective pads of the device.

Figure 109. Parallel GPIO Pad Data Out register 0–3(PGPDO[0:3])

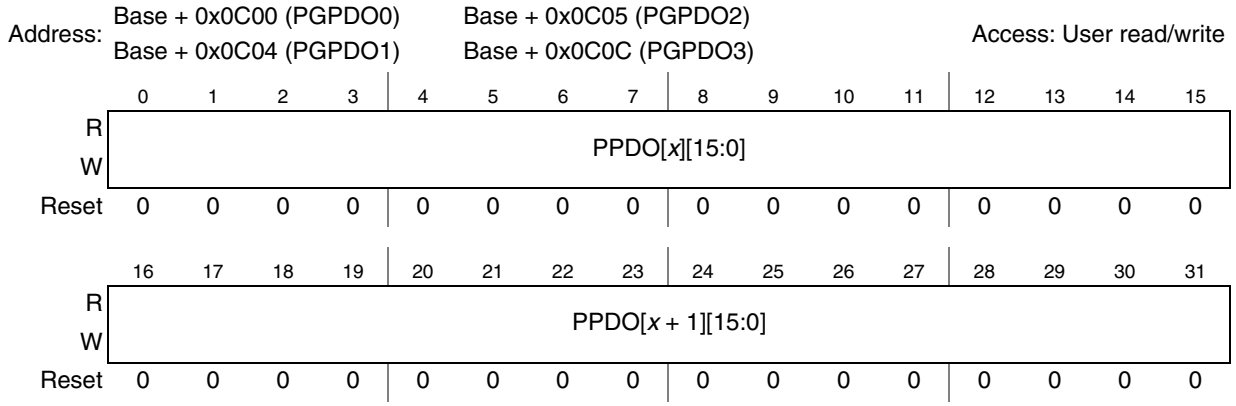


Table 105. PGPDO0_3 field descriptions

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 0_3–68_71 (GPDO[0_3:68_71]). 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*]

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]

Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

These registers hold the synchronized input value from the pads.

Figure 110. Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

Address: Base + 0x0C40 (PGPDI0), Base + 0x0C44 (PGPDI1), Base + 0x0C45 (PGPDI2), Base + 0x0C4C (PGPDI3) — Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PPDI[*x*][15:0]															
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	PPDI[*x* + 1][15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 106. PGPDI[0:3] 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 0_3–68_71 (GPDI[0_3:68_71]). 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*]

Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

This register 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 will not modify any bits in the register and cause a transfer error response by the module. Read accesses will return 0.

Figure 111. Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

Address: Base + 0x0C80 (MPGPDO0), Base + 0x0C84 (MPGPDO1), Base + 0x0C88 (MPGPDO2), Base + 0x0C8C (MPGPDO3), Base + 0x0C80 (MPGPDO4), Base + 0x0C84 (MPGPDO5), Base + 0x0C98 (MPGPDO6) — 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

Table 107. MPGPDO[0:6] field descriptions

Field	Description
MASK[*x*] [15:0]	Mask Field Each bit corresponds to one data bit in the MPPDO[x] field 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 0_3–68_71 (GPDO[0_3:68_71]). 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*]

Interrupt Filter Maximum Counter registers 0–24 (IFMC[0:24])

These registers configure the filter counter associated with each digital glitch filter.

Figure 112. Interrupt Filter Maximum Counter registers 0–24 (IFMC[0:24])

Base + 0x1000 (IFMC0)

Address: ... Access: User read/write

Base + 0x1060 (IFMC24) 25 registers

	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*[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 108. IFMC[0:24] field descriptions

Field	Description
MAXCNT*x* [3:0]	Maximum Interrupt Filter Counter setting. Filter Period = (T_{CK} × MAXCNT*x*) + (*n* × T_{CK}) Where *n* can be –2 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_{IRC} = 16 MHz)

Interrupt Filter Clock Prescaler Register (IFCPR)

This register configures a clock prescaler that selects the clock for all digital filter counters in the SIUL.

Figure 113. Interrupt Filter Clock Prescaler Register (IFCPR)

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

Table 109. IFCPR field descriptions

Field	Description
IFCP [3:0]	Interrupt Filter Clock Prescaler setting Prescaled Filter Clock Period = T_{IRC} × (IFCP + 1) T_{IRC} is the internal oscillator period. IFCP can be 0 to 15.

11.6 Functional description

11.6.1 General

This section provides a functional description of the System Integration Unit Lite.

11.6.2 Pad control

The SIUL controls the configuration and electrical characteristic 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 each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output). The pad configuration registers (PCR*n*, see *Section , "Pad Configuration Registers (PCR[0:71])*) allow software control of the static electrical characteristics of external pins with a single write. These configure the following pad features:

- Open drain output enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode behavior configuration

11.6.3 General purpose input and output pads (GPIO)

The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and 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 up to 64 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 114*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

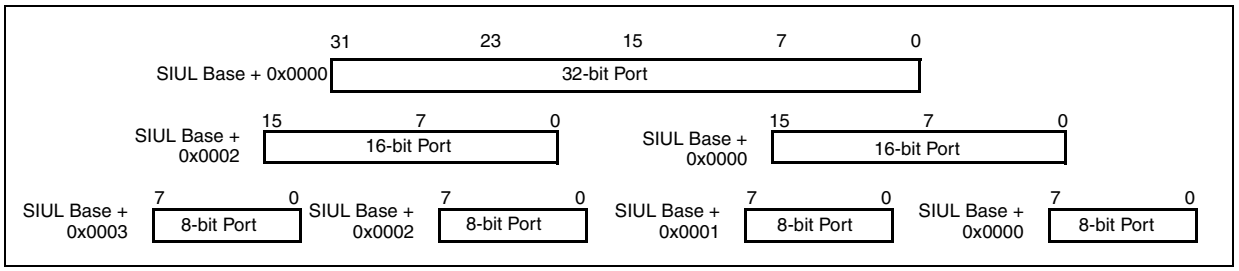


Figure 114. Data port example arrangement showing configuration for different port width accesses

This implementation requires that the registers are arranged in such a way as 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 , "GPIO Pad Data Input registers 0_3–68_71 (GPDI[0_3:68_71])*) and data output (GPDO*n_n*, see *Section , "GPIO Pad Data Output registers 0_3–68_71 (GPDO[0_3:68_71])*) registers for all pads, allowing the

possibility of reading back an input or output value of a pad directly. This supports the ability to validate what is present on the pad rather than merely 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 to be performed.

The data input registers support read access only.

When the pad is configured to use one of its alternate functions, the data input value reflect 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 will not be reflected by the pad value until reconfigured to GPIO.

The allocation of what input function is connected to the pin is defined by the PSMI registers (see *Section , "Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])*).

11.6.4 External interrupts

The SIUL supports 25 external interrupts, EIRQ[0:24]. The signal description chapter of this reference manual provides a map of the external interrupts.

The SIUL supports four interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

Refer to *Figure 115* for an overview of the external interrupt implementation.

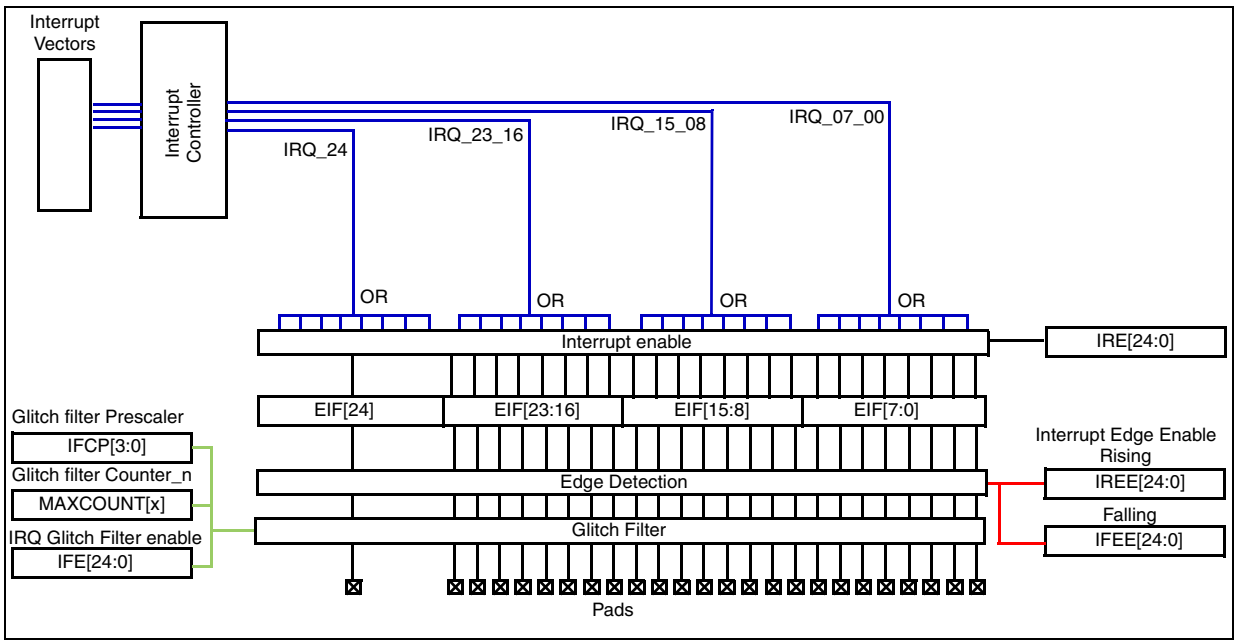


Figure 115. External interrupt pad diagram

External interrupt management

Each interrupt can be enabled or disabled independently. This can be performed using the IRER (see *Section , "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 both edges being active. A setting of having 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 ISR (see *Section , "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.

11.7 Pin muxing

For pin muxing, please refer to *Chapter 3: Signal Description* of this reference manual.

12 e200z0 and e200z0h Core

12.1 Overview

The SPC560P40/34 microcontroller implements the e200z0h core.

The e200 processor family is a set of CPU cores built on the Power Architecture technology. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0 and e200z0h processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two 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 some cases.

The e200z0 core is a single-issue, 32-bit Power Architecture technology VLE-only design with 32-bit general purpose registers (GPRs). Implementing only the VLE (variable-length encoding) APU provides improved code density. All arithmetic instructions that execute in the core operate on data in the GPRs.

12.2 Features

The following is a list of some of the key features of the e200z0 and e200z0h cores:

- 32-bit Power Architecture technology VLE-only programmer's model
- Single issue, 32-bit 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 acceleration using Branch Target Buffer (e200z0h only)
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data bus interface units (BIUs) (e200z0h only)
- Supports instruction and data access via a unified 32-bit Instruction/Data BIU (e200z0 only)
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Dynamic power management of execution units

- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

12.2.1 Microarchitecture summary

The e200z0 processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit 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), an 8 × 32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC 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 from the BTB is performed to accelerate certain taken branches in the e200z0. Prefetched instructions are placed into an instruction buffer with 4 entries (2 entries in e200z0), each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches that are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock on e200z0h. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) 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. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture 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.

Block diagram

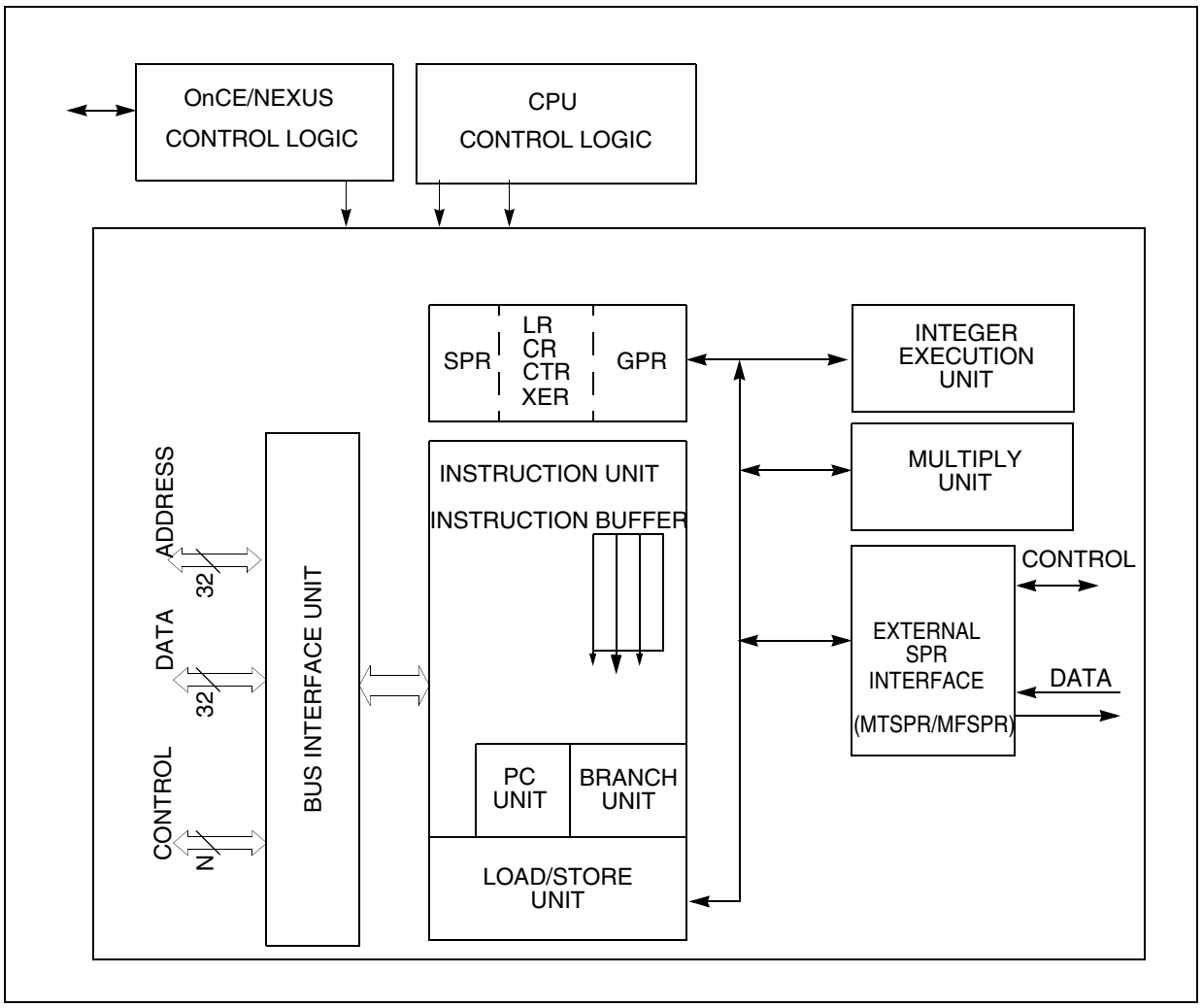


Figure 116. e200z0 block diagram

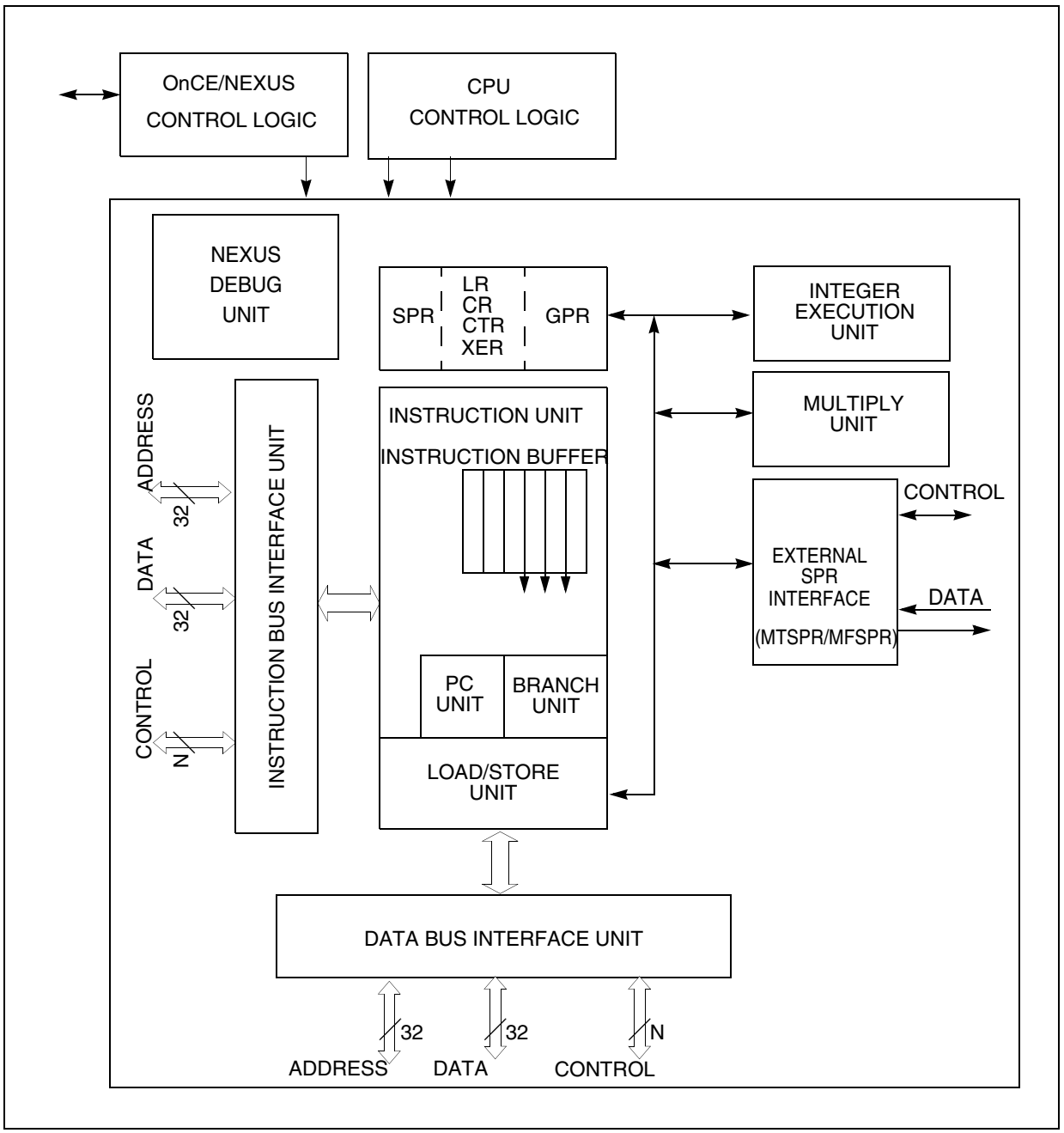


Figure 117. e200z0h block diagram

Instruction unit features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or as many as two 16-bit VLE instructions per clock
- Instruction buffer with 4 entries in e200z0h, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Instruction buffer with 2 entries in e200z0, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

Integer unit features

The e200 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 zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8 × 32 hardware multiplier array supports 1 to 4 cycle 32 × 32 → 32 multiply (early out)

Load/Store unit features

The e200 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
- 32-bit interface to memory (dedicated memory interface on e200z0h)

e200z0h system bus features

The features of the e200z0h System Bus interface are as follows:

- Independent Instruction and Data Buses
- AMBA AHB Lite Rev 2.0 Specification with support for ARM v6 AMBA Extensions
  - Exclusive Access Monitor
  - Byte Lane Strobes
  - Cache Allocate Support
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for Instruction Interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for Data Interface
- Overlapped, in-order accesses

Nexus features

The Nexus 1 module is compliant with Class 1 of the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging
- Watchpoint Trigger enable of Program Trace Messaging
- Registers for Program Trace, Ownership Trace and Watchpoint Trigger control
- All features controllable and configurable via the JTAG port

12.3 Core registers and programmer's model

This section describes the registers implemented in the e200z0 and e200z0h cores. It includes an overview of registers defined by the Power Architecture technology, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Specification.

The Power Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 118 and *Figure 120* show the e200 register set, including the registers that are accessible while in supervisor mode and the 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).

Note: e200z0 and e200z0h is a 32-bit implementation of the Power Architecture specification.

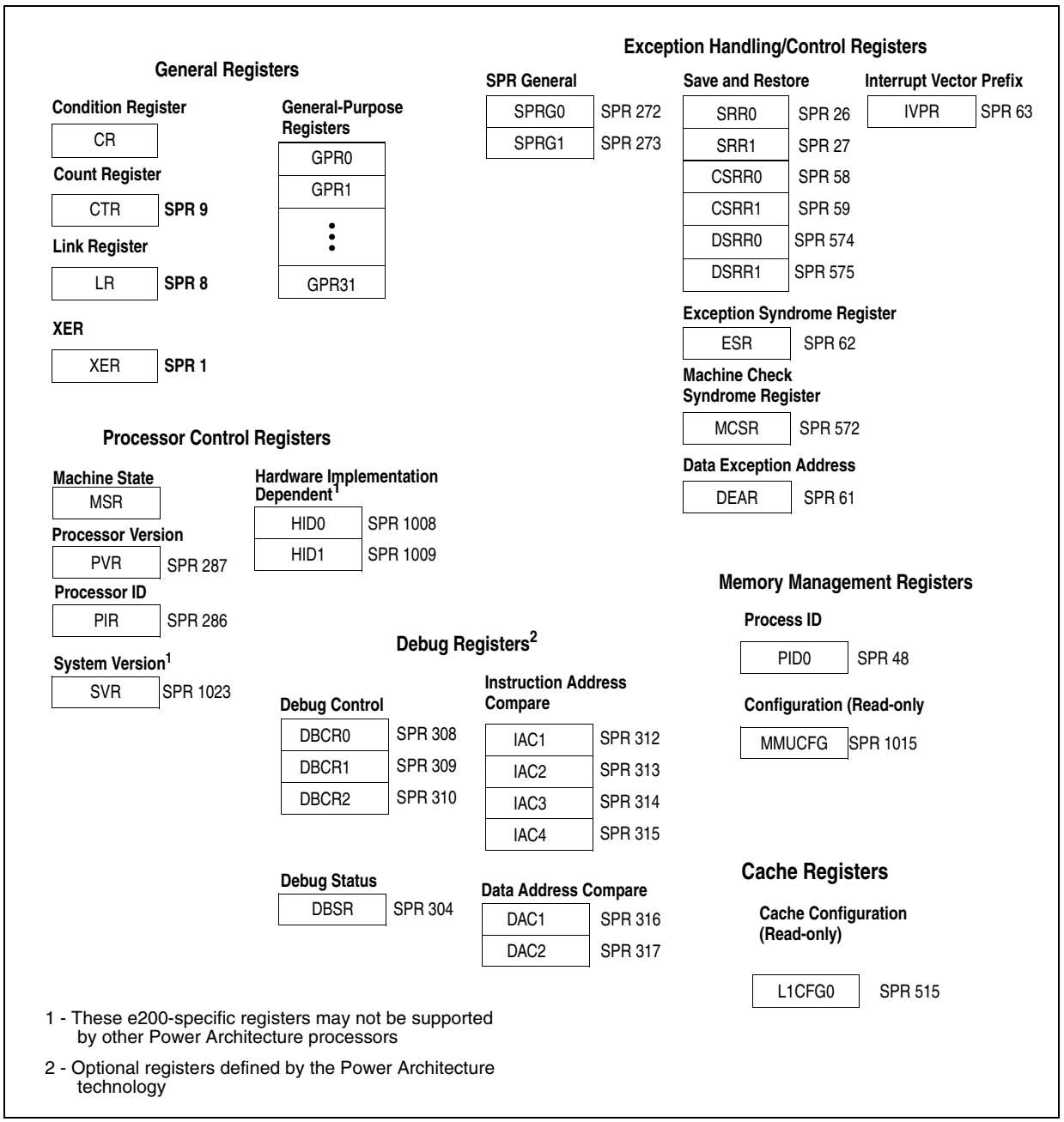


Figure 118. e200z0 Supervisor mode programmer's model

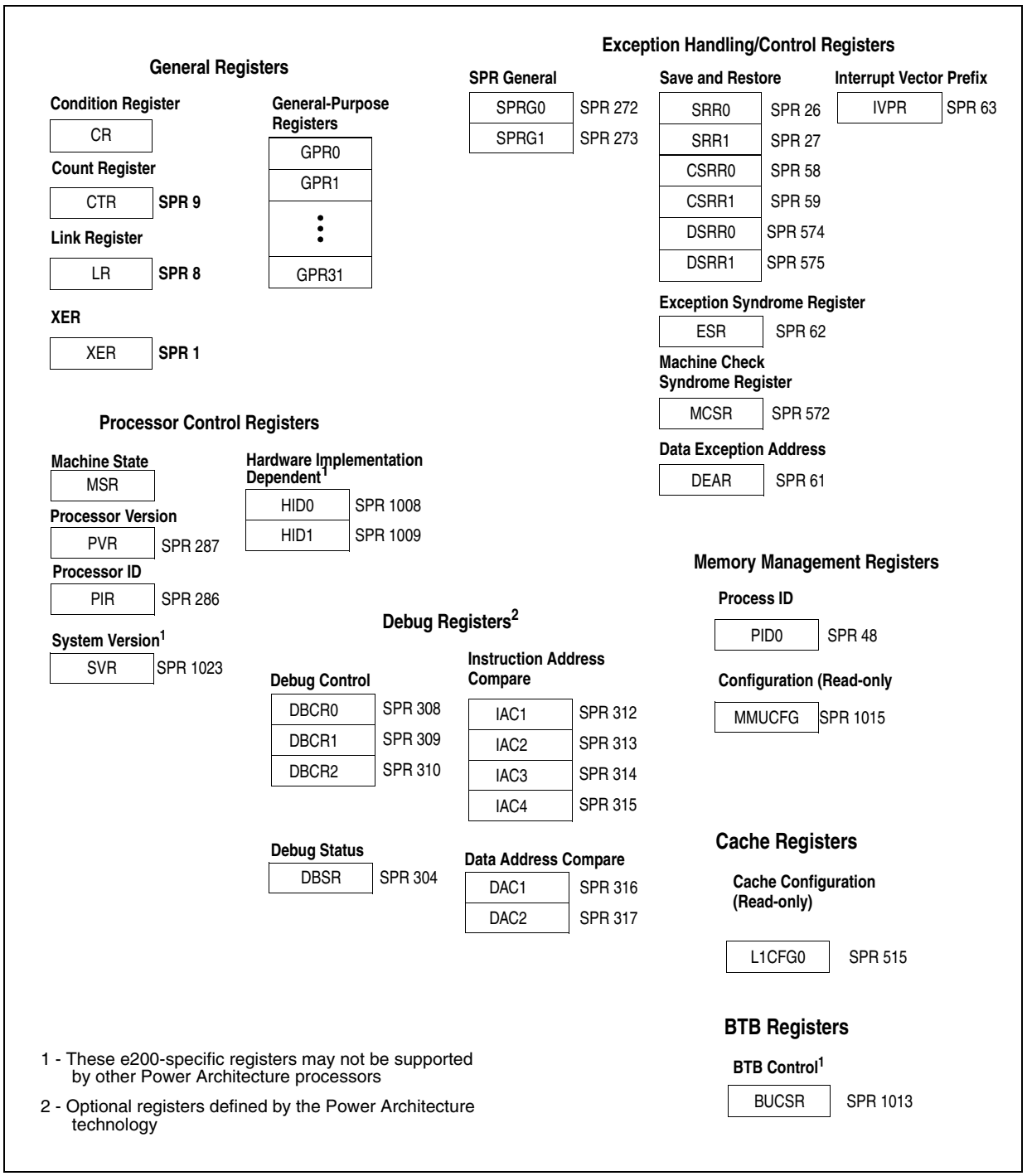


Figure 119. e200z0h Supervisor mode programmer's model

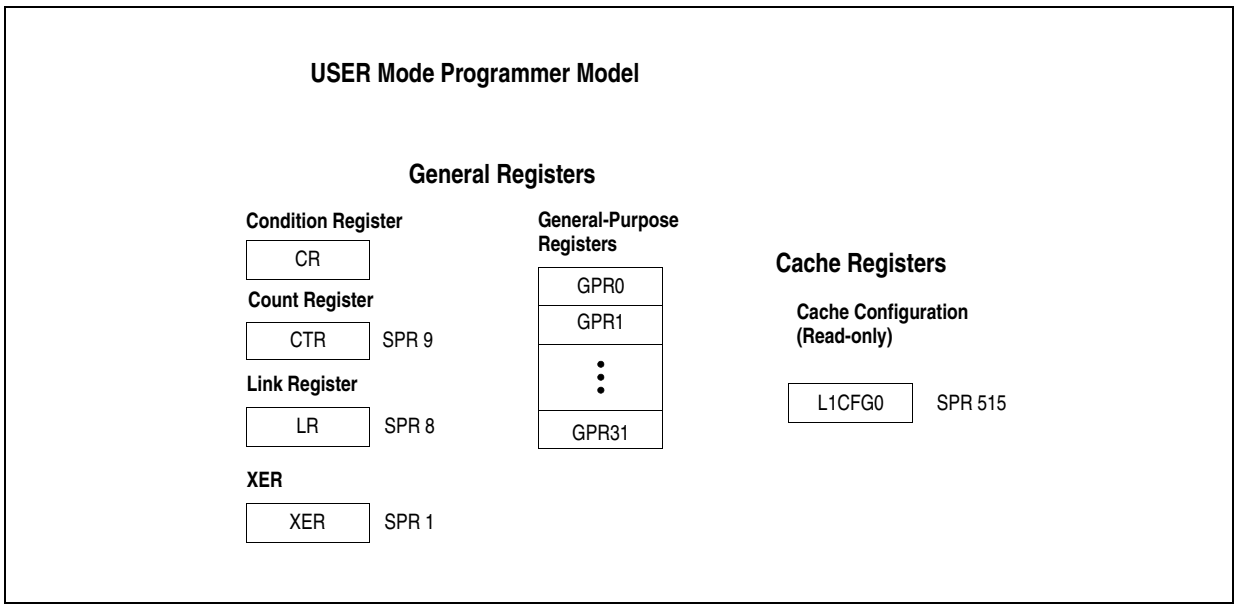


Figure 120. e200 User mode program model

12.3.1 Unimplemented SPRs and read-only SPRs

e200 fully decodes the SPR field of the **mfspr** and **mtspr** instructions. If the SPR specified is undefined and not privileged, an illegal instruction exception is generated. If the SPR specified is undefined and privileged and the CPU is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is undefined and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

For the **mtspr** instruction, if the SPR specified is read-only and not privileged, an illegal instruction exception is generated. If the SPR specified is read-only and privileged and the core is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is read-only and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

12.4 Instruction summary

The e200z0 core supports Power Architecture technology VLE instructions..

13 Peripheral Bridge (PBRIDGE)

13.1 Introduction

The Peripheral Bridge (PBRIDGE) is the interface between the system bus and on-chip peripherals.

13.1.1 Block diagram

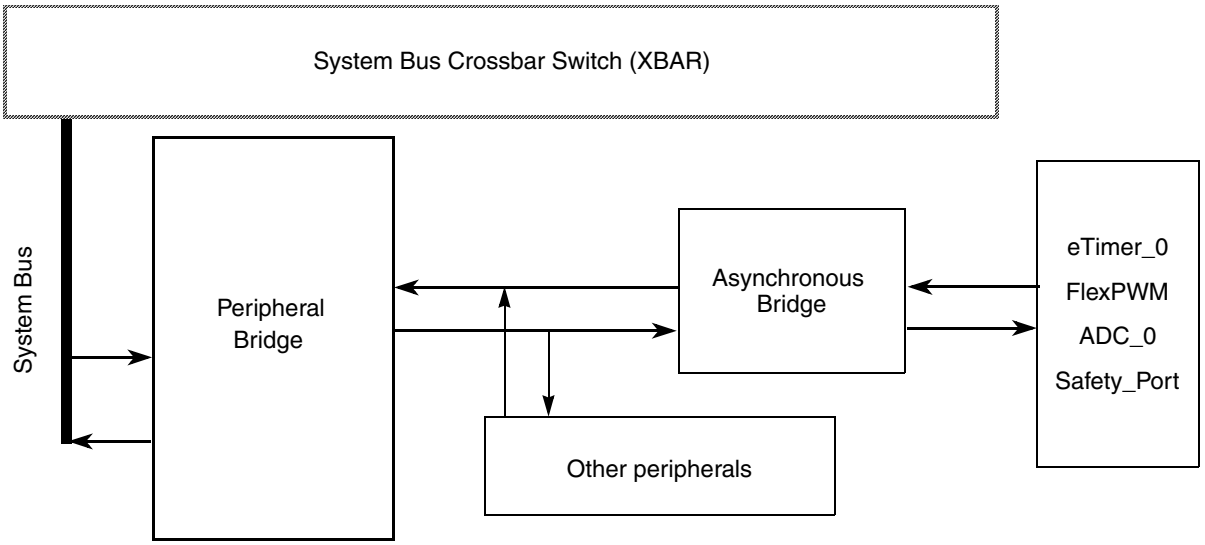


Figure 121. PBRIDGE interface

13.1.2 Overview

The PBRIDGE acts as interface between the system bus and lower bandwidth 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.

As shown in *Figure 121*, the asynchronous bridge is a dedicated module that resynchronizes signals synchronous to the system clock (SYS_CLK) to the ones synchronous to the motor control clock (MC_PLL_CLK).

The PBRIDGE has the following key features:

- Supports the slave interface signals. This interface is only meant for slave peripherals.
- Supports 32-bit slave peripherals (byte, halfword, and word reads and writes are supported to each)
- Provides configurable per-master access protections

13.1.3 Modes of operation

The PBRIDGE has only one operating mode.

13.2 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.

13.2.1 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.

Peripheral Write Buffering

Buffered writes are not supported by the device PBRIDGE.

Read cycles

Two-clock read accesses are possible with the Peripheral Bridge when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary.

Write cycles

Three clock write accesses are possible with the Peripheral Bridge when the requested access size is 32-bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported.

13.2.2 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 transfers are done in supervisor mode.

14 Crossbar Switch (XBAR)

14.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between three master ports and three slave ports. XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

14.2 Block diagram

Figure 122 shows a block diagram of the crossbar switch.

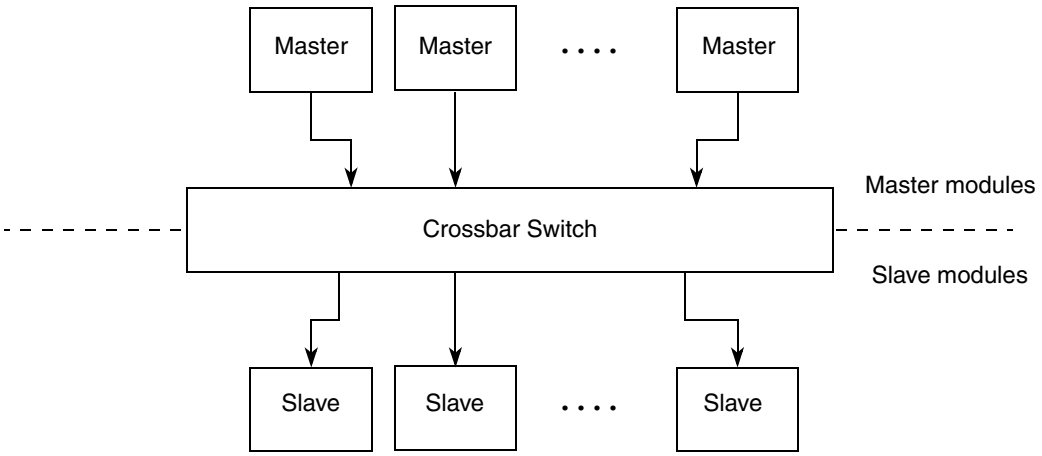


Figure 122. XBAR block diagram

Table 110 gives the crossbar switch port for each master and slave, the assigned and fixed ID number for each master and shows the master ID numbers as they relate to the master port numbers.

Table 110. Device XBAR switch ports

Module	Port		Physical master ID
	Type	Logical number	
e200z0 core—CPU instructions	Master	0	0
e200z0 core—Data	Master	0	1
eDMA	Master	2	2
Flash Controller	Slave	0	—
Internal SRAM Controller	Slave	2	—
Peripheral bridge	Slave	7	—

14.3 Overview

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.

Requesting masters are granted access based on a fixed priority.

14.4 Features

- 3 Master ports
  - e200z0 core complex Instruction port
  - e200z0 core complex Load/Store Data port
  - eDMA
- 3 Slave ports
  - Flash memory (code and data)
  - Internal SRAM
  - Peripheral bridge
- 32-bit internal address, 32-bit internal data paths
- Fully concurrent transfers between independent master and slave ports
- Fixed priority scheme and fixed parking strategy

14.5 Modes of operation

14.5.1 Normal mode

In normal mode, the XBAR provides the register interface and logic that controls crossbar switch configuration.

14.5.2 Debug mode

The XBAR operation in debug mode is identical to operation in normal mode.

14.6 Functional description

This section describes the functionality of the XBAR in more detail.

14.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. 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 masters, or inserts bubbles on the slave side.

14.6.2 General operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be just another slave to the master device, the master device has no indication that it 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 a 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 the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a 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, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed.

When a slave bus is idled by the XBAR, it is parked on the master that did the last transfer.

14.6.3 Master ports

A master access is taken if 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 access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master.

If the slave port is currently parked on another master, 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. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly 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.

14.6.4 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. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master that does not own the slave port is granted access after a one clock delay.

14.6.5 Priority assignment

Each master port is assigned a fixed 3-bit priority level (hard-wired priority). *Table 111* shows the priority levels assigned to each master (the lowest has highest priority).

Table 111. Hardwired bus master priorities

Module	Port		Priority level
	Type	Number	
e200z0 core–CPU instructions	Master	0	7
e200z0 core—Data	Master	1	6
eDMA	Master	2	5

14.6.6 Arbitration

XBAR supports only a fixed-priority comparison algorithm.

Fixed priority operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. 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 (if any). 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 higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But 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 is finished accessing the current slave port.

Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks always to the most recently requesting master (park-on-last). When parked on the last master, the slave port is passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.

15 Error Correction Status Module (ECSM)

15.1 Introduction

The Error Correction Status Module (ECSM) provides control functions for the device Standard Product Platform (SPP) including program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, and wakeup control for exiting sleep modes, and optional features such as an address map for the device's crossbar switch, information on memory errors reported by error-correcting codes and/or generic access error information for certain processor cores. It also provides with register access protection for the following slave modules: INTC, ECSM, STM, and SWT.

15.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of control functions for the platform device.

15.3 Features

The ECSM includes these features:

- Program-visible information on the platform device configuration and revision
- Reset status register (MRSR)
- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes if error-correcting codes are implemented
- Access address information for faulted memory accesses for certain processor core micro-architectures
- XBAR priority functions, including forcing round robin and high priority enabling
- Capability to restrict register access to supervisor mode to selected on-platform slave devices: INTC, ECSM, STM, and SWT

15.4 Memory map and registers description

This section details the programming model for the ECSM. 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 an "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.

ECSM registers are accessible only when the core is in supervisor mode (see *Section 15.4.3, "ECSM_reg_protection*).

15.4.1 Memory map

Table 112 lists the registers in the ECSM.

Table 112. ECSM registers

Offset from ECSM_BASE 0xFFF4_0000	Register	Location	Size (bits)
0x0000	PCT—Processor Core Type register	*on page 15-288*	16
0x0002	REV—Revision register	*on page 15-288*	16
0x0004	PLAMC — Platform XBAR Master Configuration	*on page 15-289*	16
0x0006	PLASC — Platform XBAR Sleave Configuration	*on page 15-289*	16
0x0008	IMC—IPS Module Configuration register	*on page 15-290*	16
0x000C–0x000E	Reserved		
0x000F	MRSR—Miscellaneous Reset Status register	*on page 15-290*	8
0x0010–0x001E	Reserved		
0x001F	MIR—Miscellaneous Interrupt register	*on page 15-291*	8
0x0020–0x0023	Reserved		
0x0024	MUDCR—Miscellaneous User-Defined Control Register	*on page 15-292*	32
0x0028–0x0042	Reserved		
0x0043	ECR—ECC Configuration register	*on page 15-293*	8
0x0044–0x0046	Reserved		
0x0047	ESR—ECC Status register	*on page 15-294*	8
0x0048–0x0049	Reserved		
0x004A	EEGR—ECC Error Generation register	*on page 15-296*	16
0x004C–0x004F	Reserved		
0x0050	FEAR—Flash ECC Address register	*on page 15-298*	32
0x0054–0x0055	Reserved		
0x0056	FEMR—Flash ECC Master Number Register	*on page 15-299*	8
0x0057	FEAT—Flash ECC Attributes register	*on page 15-299*	8
0x0058–0x005B	Reserved		
0x005C	FEDR—Flash ECC Data register	*on page 15-300*	32
0x0060	REAR—RAM ECC Address register	*on page 15-301*	32
0x0064	Reserved		
0x0065	RESR—RAM ECC Syndrome register	*on page 15-302*	8
0x0066	REMR—RAM ECC Master register	*on page 15-304*	8
0x0067	REAT—RAM ECC Attributes register	*on page 15-304*	8
0x0068–0x006B	Reserved		

Table 112. ECSM registers (continued)

Offset from ECSM_BASE 0xFFF4_0000	Register	Location	Size (bits)
0x006C	REDR—RAM ECC Data register	on page 15-305	32
0x0070–0x3FFF	Reserved		

15.4.2 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*, e.g., an *n*-bit register only supports *n*-bit writes, etc. 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.

Processor core type (PCT) register

The PCT is a 16-bit read-only register that specifies the architecture of the processor core in the device. 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.

Figure 123. Processor core type (PCT) register

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[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0

Table 113. PCT field descriptions

Name	Description
0-15 PCT[15:0]	Processor Core Type 0xE012 identifies the z0H Power Architecture.

Revision (REV) register

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Figure 124. Revision (REV) register

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[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 114. REV field descriptions

Name	Description
0-15 REV[15:0]	Revision The REV[15:0] field is specified by an input signal to define a software-visible revision number.

Platform XBAR Master Configuration (PLAMC)

The PLAMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the device'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.

Figure 125. Platform XBAR Master Configuration (PLAMC) register

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	AMC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	AMC[7:0]							

Table 115. PLAMC field descriptions

Field	Description
AMC[7:0]	XBAR Master Configuration 0 Bus master connection to XBAR input port *n* is not present. 1 Bus master connection to XBAR input port *n* is present.

Platform XBAR Slave Configuration (PLASC)

The PLASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the device'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.

Figure 126. Platform XBAR Slave Configuration (PLASC) register

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	ASC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	ASC[7:0]							

Table 116. ASC field descriptions

Field	Description
DP64	64-bit Datapath 0 Datapath width is 32 bits. 1 Datapath width is 64 bits.
ASC[7:0]	XBAR Slave Configuration 0 Bus slave connection to XBAR output port *n* is not present. 1 Bus slave connection to XBAR output port *n* is present.

IPS Module Configuration (IMC) register

The IMC is a 32-bit read-only register identifying the presence/absence of the 32 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.

Figure 127. IPS Module Configuration (IMC) register

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[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	MC[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 117. IMC field descriptions

Field	Description
0-31 MC[31:0]	IPS Module Configuration 0 IPS module connection to decoded slot *n* not present 1 IPS module connection to decoded slot *n* present

Miscellaneous Reset Status Register (MRSR)

The 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 device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored.

Figure 128. Miscellaneous Reset Status Register (MRSR)

Address: Base + 0x000F Access: User read-only

	0	1	2	3	4	5	6	7
R	POR	DIR	0	0	0	0	0	0
W								
Reset	x	x	0	0	0	0	0	0

Table 118. MRSR field descriptions

Field	Description
0 POR	Power-On Reset 0 Last recorded event was not caused by a power-on reset (based on a device input signal). 1 Last recorded event was caused by a power-on reset (based on a device input signal).
1 DIR	Device Input Reset 0 Last recorded event was not caused by a device input reset. 1 Last recorded event was a reset caused by a device input reset.

Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR register. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the ECSMIR must be explicitly cleared.

Figure 129. Miscellaneous Interrupt Register (MIR)

Address: Base + 0x001F Access: User read/write

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	0	0	0	0
W	1	1	1	1	x	x	x	x
Reset	0	0	0	0	0	0	0	0

Table 119. MIR field descriptions

Field	Description
0 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 negated by writing a 1 to this bit. Writing a 0 has no effect.
1 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 negated by writing a 1 to this bit. Writing a 0 has no effect.
2 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 negated by writing a 1 to this bit. Writing a 0 has no effect.

Table 119. MIR field descriptions (continued)

Field	Description
3 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 negated by writing a 1 to this bit. Writing a 0 has no effect.

Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It provides configuration control for assorted modules on the device. The contents of this register is output from the ECSM to other modules where these user-defined control functions are implemented.

Figure 130. Miscellaneous User-Defined Control register (MUDCR)

Address Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MUDCR[31]	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 120. MUDCR field descriptions

Name	Description
0 MUDCR[31]	XBAR force_round_robin bit This bit is used to drive the force_round_robin bit of the XBAR. This forces the slaves into round robin mode of arbitration rather than fixed mode (unless a master is using priority elevation, which forces the design back into fixed mode regardless of this bit). By setting the hardware definition to ENABLE_ROUND_ROBIN_RESET, this bit resets to 1. 0 XBAR is in fixed priority mode. 1 XBAR is in round robin mode.

ECC registers

There are a number of program-visible registers for the sole purpose of reporting and logging of 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. If the design does not include ECC on the memories, these addresses are reserved locations within the ECSM's programming model.

ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches, which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

The reporting of single-bit memory corrections can only be enabled via an SoC-configurable module input signal. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. While not directly accessible to a user, this capability is viewed as important for error logging and failure analysis.

Figure 131. ECC Configuration register (ECR)

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

Table 121. ECR field descriptions

Field	Description
2 ER1BR	Enable RAM 1-bit Reporting This bit can only be set if the input enable signal is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. The occurrence of a single-bit RAM correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0 Reporting of single-bit RAM corrections disabled 1 Reporting of single-bit RAM corrections enabled
3 EF1BR	Enable Flash 1-bit Reporting This bit can only be set if the input enable signal is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0Reporting of single-bit flash corrections disabled 1Reporting of single-bit flash corrections enabled
6 ERNCR	Enable RAM Non-Correctable Reporting The occurrence of a non-correctable multi-bit RAM error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0Reporting of non-correctable RAM errors disabled 1Reporting of non-correctable RAM errors enabled
7 EFNCR	Enable Flash Non-Correctable Reporting The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0 Reporting of non-correctable flash errors disabled 1 Reporting of non-correctable flash errors enabled

ECC Status Register (ESR)

The ECC Status Register 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_ECC2BIT_IRQ
    =  ECSM_ECCRNCR_IRQ   // ram, noncorrectable error
    |  ECSM_ECCFNCR_IRQ   // flash, noncorrectable error
ECSM_ECC_IRQ
    =  ECSM_ECC1BIT_IRQ   // 1-bit correction
    |  ECSM_ECC2BIT_IRQ   // noncorrectable error
```

where the combination of a properly 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 an 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 guaranteeing that only a single flag is asserted.

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.

Figure 132. ECC Status register (ESR)

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								
Reset	0	0	0	0	0	0	0	0

Table 122. ESR field descriptions

Field	Description
2 R1BC	RAM 1-bit Correction This bit can only be set if ECR[ER1BR] is asserted. The occurrence of a properly enabled single-bit RAM correction generates a 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. 0 No reportable single-bit RAM correction detected 1 Reportable single-bit RAM correction detected
3 F1BC	Flash 1-bit Correction This bit can only be set if ECR[EF1BR] is asserted. The occurrence of a properly enabled single-bit flash correction generates a 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. 0 No reportable single-bit flash correction detected 1 Reportable single-bit flash correction detected

Table 122. ESR field descriptions (continued)

Field	Description
6 RNCE	RAM Non-Correctable Error The occurrence of a properly enabled non-correctable RAM error generates a 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. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable RAM error detected 1 Reportable non-correctable RAM error detected
7 FNCE	Flash Non-Correctable Error The occurrence of a properly enabled non-correctable flash error generates a 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. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable flash error detected 1 Reportable non-correctable flash error detected

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.

ECC Error Generation Register (EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- 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, that is, the ability to program the non-volatile memory with single- 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.

The enabling of these error generation modes requires the same input enable signal (as that used to enable single-bit correction reporting) be asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.

Figure 133. ECC Error Generation register (EEGR)

Address Base + 0x004A — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC 1BI	FR11 BI	0	0	FRC NCI	FR1 NCI	0	ERRBIT[6:0]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 123. EEGR field descriptions

Field	Description
2 FRC1BI	Force RAM Continuous 1-Bit Data Inversions 0 No RAM continuous 1-bit data inversions generated 1 1-bit data inversions in the RAM 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[6:0], 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 properly re-enable the error generation logic. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.
3 FR11BI	Force RAM One 1-bit Data Inversion 0 No RAM single 1-bit data inversion generated 1 One 1-bit data inversion in the RAM 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[6:0], 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. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.
6 FRCNCI	Force RAM Continuous Non-Correctable Data Inversions 0 No RAM continuous 2-bit data inversions generated 1 2-bit data inversions in the RAM 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[6:0] 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.

Table 123. EEGR field descriptions (continued)

Field	Description
7 FR1NCI	Force RAM One Non-Correctable Data Inversions 0 No RAM single 2-bit data inversions generated 1 One 2-bit data inversion in the RAM 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[6:0] 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.
9-15 ERRBIT [6:0]	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 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 32-bit RAM implementation. The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32 bits data + 7 bits for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted. if ERRBIT = 1, then RAM[1] of the odd bank is inverted. ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted. if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted. if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted. ... if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted. For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.

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 will be generated.

The only allowable values for the 4 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 undefined behavior.

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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 134. Flash ECC Address register (FEAR)

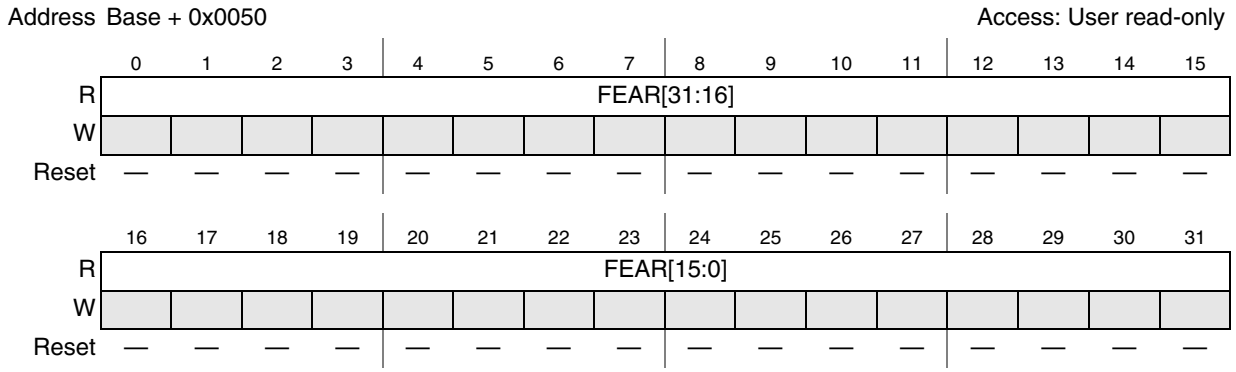


Table 124. FEAR field descriptions

Field	Description
0-31 FEAR[31:0]	Flash ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled flash ECC event.

Flash ECC Master Number Register (FEMR)

The FEMR is a 4-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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 135. Flash ECC Master Number Register (FEMR)

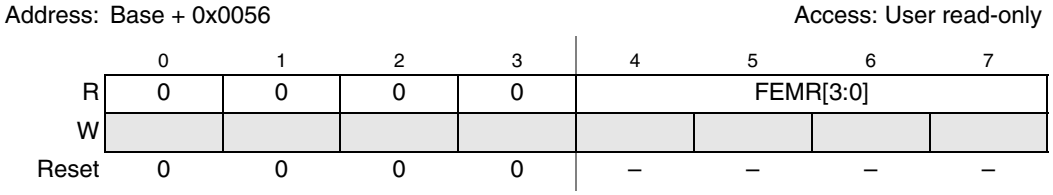


Table 125. FEMR field descriptions

Name	Description
4-7 FEMR[3:0]	Flash ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last properly enabled flash ECC event.

Flash ECC Attributes (FEAT) register

The FEAT 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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 136. Flash ECC Attributes (FEAT) Register

Address: Base + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROTECTION[3:0]			
W								
Reset	0	0	0	0	–	–	–	–

Table 126. FEAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

Flash ECC Data Register (FEDR)

The FEDR is a 32-bit register for capturing the data associated with the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, 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 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.

Figure 137. Flash ECC Data register (FEDR)

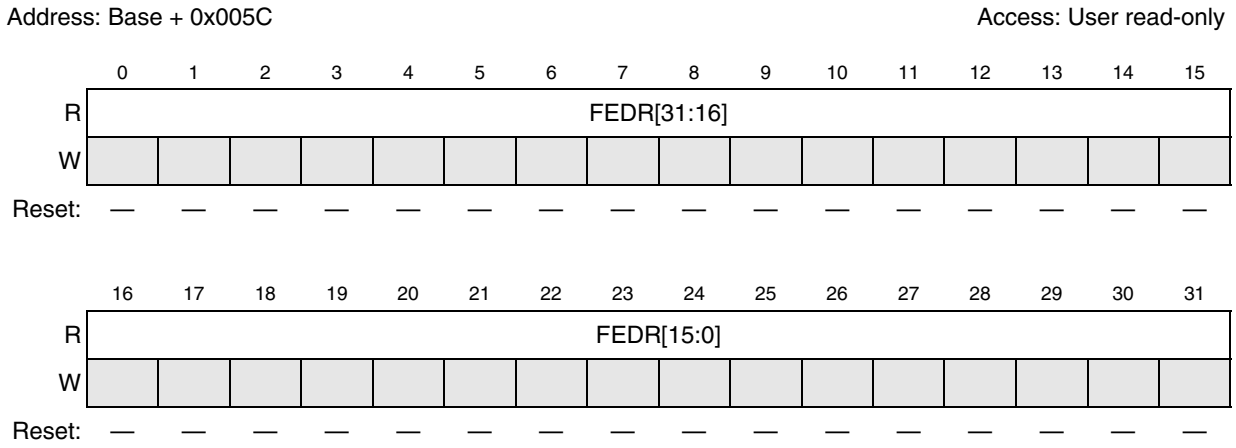


Table 127. FEDR field descriptions

Name	Description
0-31 FEDR[31:0]	Flash ECC Data Register This 32-bit register 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.

RAM ECC Address Register (REAR)

The 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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 138. RAM ECC Address register (REAR)

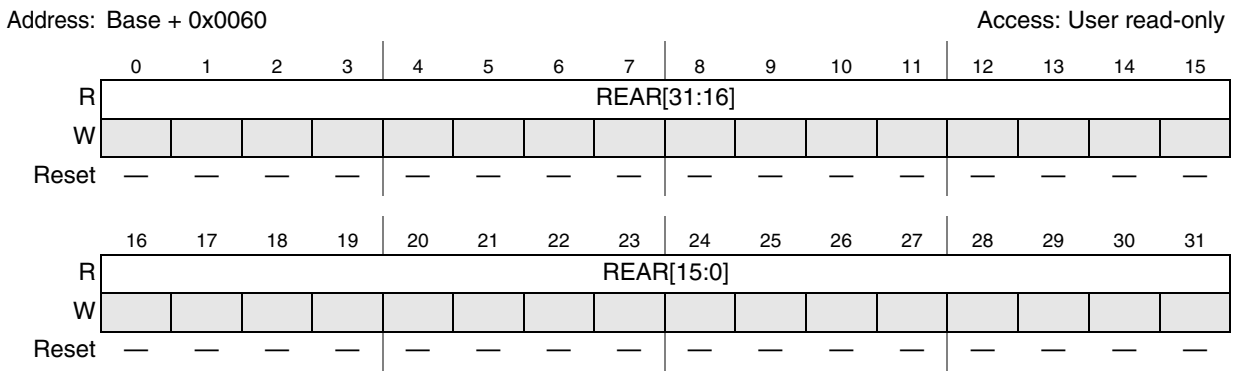


Table 128. REAR field descriptions

Name	Description
0-31 REAR[31:0]	RAM ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled RAM ECC event.

RAM ECC Syndrome Register (RESR)

The 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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 139. RAM ECC Syndrome Register (RESR)

Address: Base + 0x0065 Access: User read-only

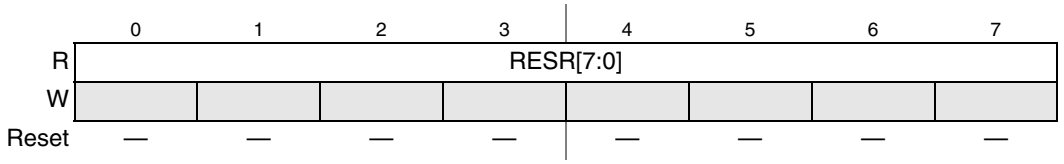


Table 129. RESR field descriptions

Name	Description
0-7 RESR[7:0]	RAM ECC Syndrome Register This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown in *Table 130* associates the upper 7 bits of the syndrome with the data bit in error.

Note: Table 130 associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB=0. Note that the syndrome value of 0x0001 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 130. RAM syndrome mapping for single-bit correctable errors

RESR[7:0]	Data Bit in Error
0x00	ECC ODD[0]
0x01	No Error
0x02	ECC ODD[1]
0x04	ECC ODD[2]

Table 130. RAM syndrome mapping for single-bit correctable errors (continued)

RESR[7:0]	Data Bit in Error
0x06	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x0A	DATA ODD BANK[30]
0x0C	DATA ODD BANK[29]
0x0E	DATA ODD BANK[28]
0x10	ECC ODD[4]
0x12	DATA ODD BANK[27]
0x14	DATA ODD BANK[26]
0x16	DATA ODD BANK[25]
0x18	DATA ODD BANK[24]
0x1A	DATA ODD BANK[23]
0x1C	DATA ODD BANK[22]
0x50	DATA ODD BANK[21]
0x20	ECC ODD[5]
0x22	DATA ODD BANK[20]
0x24	DATA ODD BANK[19]
0x26	DATA ODD BANK[18]
0x28	DATA ODD BANK[17]
0x2A	DATA ODD BANK[16]
0x2C	DATA ODD BANK[15]
0x58	DATA ODD BANK[14]
0x30	DATA ODD BANK[13]
0x32	DATA ODD BANK[12]
0x34	DATA ODD BANK[11]
0x64	DATA ODD BANK[10]
0x38	DATA ODD BANK[9]
0x62	DATA ODD BANK[8]
0x70	DATA ODD BANK[7]
0x60	DATA ODD BANK[6]
0x40	ECC ODD[6]
0x42	DATA ODD BANK[5]
0x44	DATA ODD BANK[4]
0x46	DATA ODD BANK[3]
0x48	DATA ODD BANK[2]
0x4A	DATA ODD BANK[1]

Table 130. RAM syndrome mapping for single-bit correctable errors (continued)

RESR[7:0]	Data Bit in Error
0x4C	DATA ODD BANK[0]
0x03,0x05.....0x4D	Multiple bit error
> 0x4D	Multiple bit error

RAM ECC Master Number Register (REMR)

The REMR is an 8-bit register in which the 4-bit field REMR[0:3] is used 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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 140. RAM ECC Master Number register (REMR)

Address: Base + 0x0066 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR[3:0]			
W								
Reset	0	0	0	0	—	—	—	—

Table 131. REMR field descriptions

Name	Description
4-7 REMR[3:0]	RAM ECC Master Number Register This 4-bit field contains the XBAR bus master number of the faulting access of the last properly enabled RAM ECC event.

RAM ECC Attributes (REAT) register

The REAT 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, 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 to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Figure 141. RAM ECC Attributes (REAT) register

Address: Base + 0x0067 Access: User read/write

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROTECTION[3:0]			
W								
Reset	—	—	—	—	—	—	—	—

Table 132. REAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

RAM ECC Data Register (REDR)

The REDR is a -bit register for capturing the data associated with the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, 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 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.

Figure 142. Platform RAM ECC Data register (PREDR)

Address: Base + 0x006C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REDR[31:16]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REDR[15:0]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 133. REDR field descriptions

Name	Description
0-31 REDR[31:0]	RAM ECC Data Register This 32-bit register 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.

15.4.3 ECSM_reg_protection

The ECSM_reg_protection logic provides hardware enforcement of supervisor mode access protection for four on-platform IPS modules: INTC, ECSM, STM, and SWT. This logic resides between the on-platform bus sourced by the PBRIDGE bus controller and the individual slave modules. It monitors the bus access type (supervisor or user) and if a user access is attempted, the transfer is terminated with an error and inhibited from reaching the slave module. Identical logic is replicated for each of the five, targeted slave modules. A block diagram of the ECSM_reg_protection module is shown in *Figure 143*.

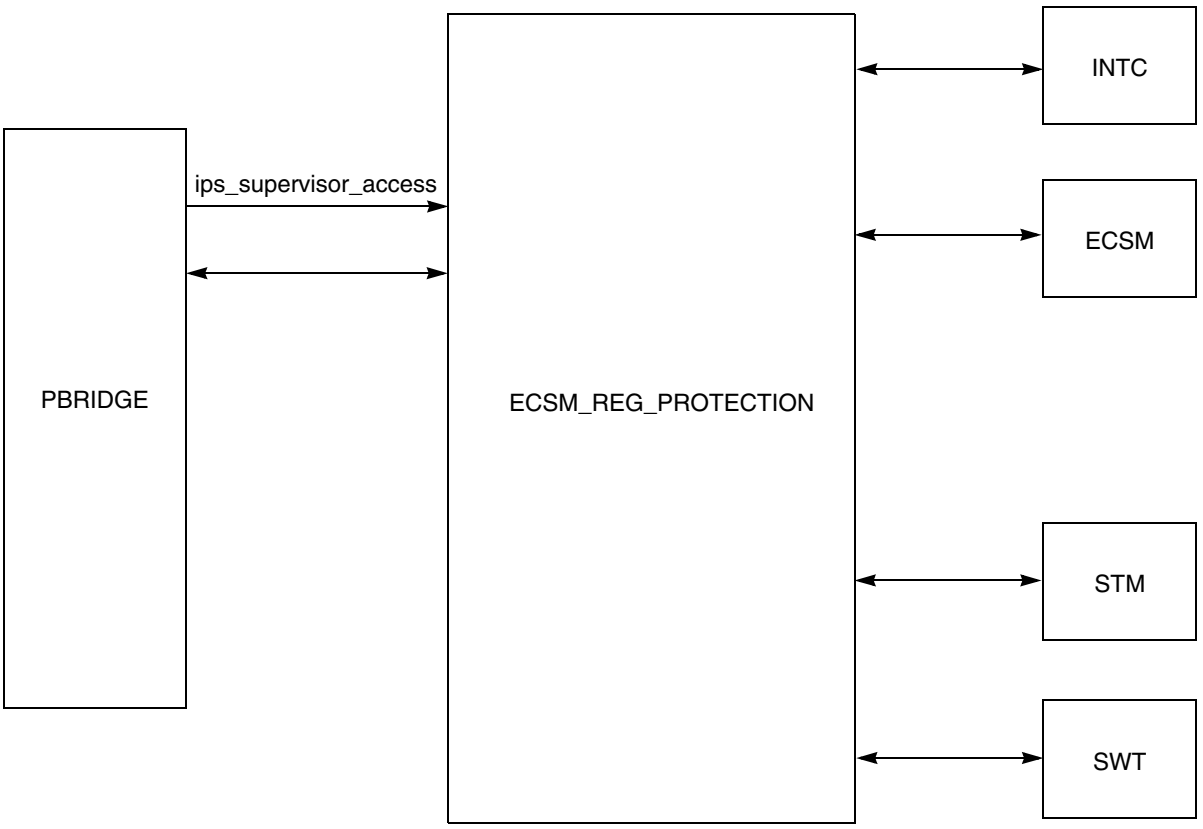


Figure 143. Spp_Ips_Reg_Protection block diagram

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, etc. 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.

16 Internal Static RAM (SRAM)

16.1 Introduction

The general-purpose SRAM has a size of 20 KB.

The SRAM provides 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

16.2 SRAM operating mode

The SRAM has only one operating mode. No standby mode is available.

Table 134. SRAM operating modes

Mode	Configuration
Normal (functional)	Allows reads and writes of SRAM

16.3 Module memory map

The SRAM occupies up to 20 KB of address space.

Table 135 shows the SRAM memory map.

Table 135. SRAM memory map

Address	Use
0x4000_0000 (Base)	20 KB RAM

16.4 Register descriptions

The SRAM has no registers. Registers associated with the ECC are located in the ECSM. See *Section , "ECC registers*.

16.5 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 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

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)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1 or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1 or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

16.5.1 Access timing

The system bus is a two-stage pipelined bus that makes the timing of any access dependent on the access during the previous clock. *Table 136* 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 currently executing
- 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 136. Number of wait states required for SRAM operations

Operation type	Current operation	Previous operation	Number of wait states required
Read	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

Table 136. Number of wait states required for SRAM operations (continued)

Operation type	Current operation	Previous operation	Number of wait states required
Write	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	0
		32-bit write	
		Read	

16.5.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt RAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. If no access is occurring when reset occurs, RAM corruption does not happen.

Instead synchronous reset (SW reset) should be used in controlled function (without RAM accesses) in case initialization procedure is needed without RAM initialization.

16.6 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 R/W operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior any read accesses. This is also true for implicit read accesses caused by any write accesses smaller than 32 bits as discussed in *Section 16.5, "SRAM ECC mechanism*.

16.7 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. All writes must specify an even number of registers performed on 32-bit word-aligned boundaries. If the write is not the entire 32-bits (8 or 16 bits), a read/modify/write operation is generated that checks the ECC value upon the read. Refer to *Section 16.5, "SRAM ECC mechanism*.

Note: You must initialize SRAM, even if the application does not use ECC reporting.

17 Flash Memory

17.1 Introduction

The Flash memory comprises a platform Flash controller interface and two Flash memory arrays: one array of 256 KB for code (code Flash) and one array of 64 KB for data (data Flash). The Flash architecture of the SPC560P40/34 device is illustrated in *Figure 144*.

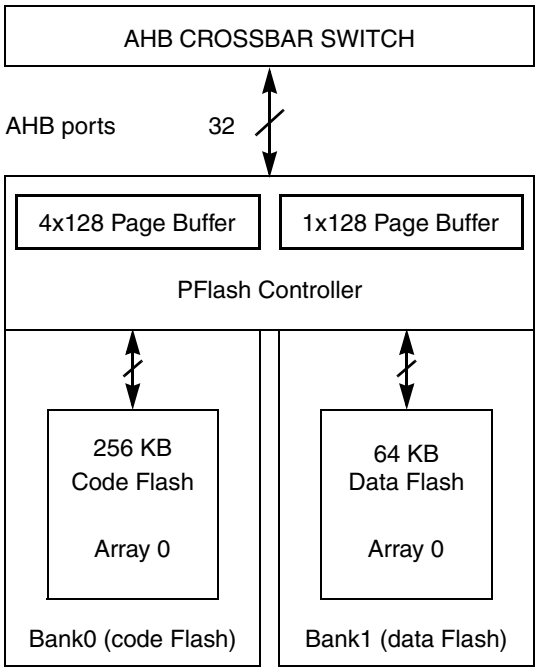


Figure 144. SPC560P40/34 Flash memory architecture

17.2 Platform Flash controller

17.2.1 Introduction

This section provides an introduction of the platform Flash controller, which acts as the interface between the system bus and as many as two banks of Flash memory arrays (program and data). It intelligently converts the protocols between the system bus and the dedicated Flash array interfaces. Several important terms are used to describe the platform Flash controller module and its connections. These terms are defined here.

- **Port**—This term describes the AMBA-AHB connection(s) into the platform Flash controller. From an architectural and programming model viewpoint, the definition supports as many as two AHB ports, even though this specific controller only supports a single AHB connection.
- **Bank**—This term describes the attached Flash memories. From the platform Flash controller's perspective, there may be one or two attached banks of Flash memory. The code Flash bank is required and always attached to bank0. Additionally, there is a data Flash attached to bank1. The platform Flash controller interface supports two separate

connections, one to each memory bank. On the SPC560P40/34 device, bank0 and bank1 are internal to the device.

- **Array**—Each memory bank has one Flash array instantiation.
- **Page**—This value defines the number of bits read from the Flash array in a single access. For this controller and memory, the page size is 128 bits (16 bytes).

The nomenclature “page buffers” and “line buffers” are used interchangeably.

Overview

The platform Flash controller supports a 32-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one instantiation of the Flash memory array. One Flash bank is connected to the code Flash memory and the other bank is connected to the data Flash memory. The memory controller capabilities vary between the two banks with each bank’s functionality optimized with the typical use cases associated with the attached Flash memory. As an example, the platform Flash controller logic associated with the code Flash bank contains a four-entry “page” buffer, each entry containing 128 bits of data (1 Flash page) plus an associated controller that prefetches sequential lines of data from the Flash array into the buffer, while the controller logic associated with the data Flash bank only supports a 128-bit register that serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code Flash bank support 0-wait AHB data phase responses. AHB read requests that miss the buffers generate the needed Flash array access and are forwarded to the AHB upon completion, typically incurring two wait states at an operating frequency of 60 to 64 MHz.

This memory controller is optimized for applications where a cacheless processor core, for example the Power e200z0h, is connected through the platform to on-chip memories, for example Flash and RAM, where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor’s pipeline and 0 wait state responses for most memory accesses are critical for providing the required level of system performance.

Features

The following list summarizes the key features of the platform Flash controller:

- Single AHB port interface supports a 32-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank.
- Interface with code Flash (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller support single-cycle read responses (0 AHB data phase wait states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Interface with data Flash (bank1) includes a 128-bit register to temporarily hold a single Flash page. This logic supports single-cycle read responses (0 AHB data phase wait

states) for accesses that hit in the holding register. There is no support for prefetching associated with bank1.

- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional Flash operation termination, and optional termination notification interrupt
- Separate and independent configurable access timing (on a per bank basis) to support use across a wide range of platforms and frequencies
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit Flash ECC events
- Typical operating configuration loaded into programming model by system reset

17.2.2 Modes of operation

The platform Flash controller module does not support any special modes of operation. Its operation is driven from the AMBA-AHB memory references it receives from the platform's bus masters. Its configuration is defined by the setting of the programming model registers, physically located as part of the Flash array modules.

17.2.3 External signal descriptions

The platform Flash controller does not directly interface with any external signals. Its primary internal interfaces include a connection to an AMBA-AHB crossbar (or memory protection unit) slave port and connections with as many as two banks (code and data) of Flash memory, each containing one instantiation of the Flash array. Additionally, the operating configuration for the platform Flash controller is defined by the contents of certain code Flash array0 registers that are inputs to the module.

17.2.4 Memory map and registers description

Two memory maps are associated with the platform Flash controller: one for the Flash memory space and another for the program-visible control and configuration registers. The Flash memory space is accessed via the AMBA-AHB port. The program-visible registers are accessed via the slave peripheral bus. Details on both memory spaces are provided in *Section , "Memory map.*

There are no program-visible registers that physically reside inside the platform Flash controller. Rather, the platform Flash controller receives control and configuration information from the Flash array controller(s) to determine the operating configuration. These are part of the Flash array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

Note: Updating the configuration fields that control the platform flash controller behavior should only occur while the flash controller is idle. Changing configuration settings while a flash access is in progress can lead to non-deterministic behavior.

Memory map

First, consider the Flash memory space accessed via transactions from the platform Flash controller's AHB port. To support the two separate Flash memory banks, the platform Flash controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual Flash memory regions, there are shadow and test sectors included in the system memory map. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address region. The

system memory map defines one code Flash array and one data Flash array. See *Table 137*.

Caution:

Software executing from flash memory must not write to registers that control flash behavior (such as wait state settings or prefetch enable/disable). Doing so can cause data corruption. On this chip, these registers include PFCR0 and PFAPR.

Note: 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-, 16-bit) boundaries.

Table 137. Flash-related regions in the system memory map

Start address	End address	Size (KB)	Region
0x0000_0000	0x0007_FFFF	512	Code Flash array 0
0x0008_0000	0x001F_FFFF	1536	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash array 0: shadow sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash array 0: test sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash array 0
0x0081_0000	0x00C0_1FFF	4040	Reserved
0x00C0_2000	0x00C0_3FFF	8	Data Flash array 0: test sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Emulation Mapping
0xFFE8_8000	0xFFE8_BFFF	16	Code Flash array 0 configuration(1)
0xFFE8_C000	0xFFE8_FFFF	16	Data Flash array 0 configuration(1)
0xFFEB_0000	0xFFEB_BFFF	48	Reserved

1. This region is also aliased to address 0xC3F8_nnnn.

For additional information on the address-based read access timing for emulation of other memory types, see *Section 17.2.17, "Wait state emulation.*

Next, consider the memory map associated with the control and configuration registers.

There are multiple registers that control operation of the platform Flash controller. Note the first two Flash array registers (PFCR0, PFCR1) are reset to a device-defined value, while the remaining register (PFAPR) is loaded at reset from specific locations in the array's shadow region.

Regardless of the number of populated banks or the number of Flash arrays included in a given bank, the configuration of the platform Flash controller is wholly specified by the platform Flash controller control registers associated with code Flash array0. The code array0 register settings define the operating behavior of **both** Flash banks. It is recommended to set the platform Flash controller control registers for both arrays to the array0 values.

Note: To perform program and erase operations, the control registers in the actual referenced Flash array must both be programmed, but the configuration of the platform Flash controller module is defined by the platform Flash controller control registers of code array0.

The 32-bit memory map for the platform Flash controller control registers is shown in *Table 138*.

Table 138. Platform Flash controller 32-bit memory map

Offset from PFlash_BASE (0xFFE8_8000)	Register	Location
0x001C	Platform Flash Configuration Register 0 (PFCR0)	*on page 17-353*
0x0020	Platform Flash Configuration Register 1 (PFCR1)	*on page 17-357*
0x0024	Platform Flash Access Protection Register (PFAPR)	*on page 17-359*

17.2.5 Functional description

The platform Flash controller interfaces between the AHB-Lite 2.v6 system bus and the Flash memory arrays.

The platform Flash controller generates read and write enables, the Flash array address, write size, and write data as inputs to the Flash array. The platform Flash controller captures read data from the Flash array interface and drives it onto the AHB. As much as four pages of data (128-bit width) from bank0 are buffered by the platform Flash controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (0 AHB wait states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch.

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

Throughout this discussion, bk*n*_ is used as a prefix to refer to two signals, each for each bank: bk0_ and bk1_. Also, the nomenclature B*x*_P*y*_RegName is used to reference a program-visible register field associated with bank "*x*" and port "*y*".

17.2.6 Basic interface protocol

The platform Flash controller interfaces to the Flash array by driving addresses (bk*n*_fl_addr[23:0]) and read or write enable signals (bk*n*_fl_rd_en, bk*n*_fl_wr_en).

The read or write enable signal (bk*n*_fl_rd_en, bk*n*_fl_wr_en) is asserted in conjunction with the reference address for a single rising clock when a new access request is made.

Addresses are driven to the Flash array in a flow-through fashion to minimize array access time. When no outstanding access is in progress, the platform Flash controller drives addresses and asserts bk*n*_fl_rd_en or bk*n*_fl_wr_en and then may change to the next outstanding address in the next cycle.

Accesses are terminated under control of the appropriate read/write wait state control setting. Thus, the access time of the operation is determined by the settings of the wait state control fields. Access timing can be varied to account for the operating conditions of the device (frequency, voltage, temperature) by appropriately setting the fields in the programming model for either bank.

The platform Flash controller also has the capability of extending the normal AHB access time by inserting additional wait states for reads and writes. This capability is provided to allow emulation of other memories that have different access time characteristics. The added wait state specifications are provided by bit 28 to bit 24 of Flash address (haddr[28:24], see *Table 140* and *Table 141*). These wait states are applied in addition to the normal wait states incurred for Flash accesses. Refer to *Section 17.2.17, "Wait state emulation* for more details.

Prefetching of next sequential page is blocked when haddr[28:24] is non-zero. Buffer hits are also blocked as well, regardless of whether the access corresponds to valid data in one of the page read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided. In addition, to prevent erroneous operation in certain rare cases, the buffers are invalidated on any non-sequential AHB access with a non-zero value on haddr[28:24].

17.2.7 Access protections

The platform Flash controller provides programmable configurable access protections for both read and write cycles from masters via the Platform Flash Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in *Section , "Platform Flash Access Protection Register (PFAPR).* Detection of a protection violation results in an error response from the platform Flash controller on the AHB transfer.

17.2.8 Read cycles — buffer miss

Read cycles from the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0] and asserting bk*n*_fl_rd_en for the required setup (and hold) time before (and after) the rising edge of hclk. The platform Flash controller then waits for the programmed number of read wait states before sampling the read data on bk*n*_fl_rdata[127:0]. This data is normally stored in the least-recently updated page read buffer for bank0 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the page-wide temporary holding register as the requested data is forwarded to the AHB bus. Timing diagrams of basic read accesses from the Flash array are shown in *Figure 145* through *Figure 148*.

If the Flash access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the Flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

17.2.9 Read cycles — buffer hit

Single cycle read responses to the AHB are possible with the platform Flash controller when the requested read access was previously loaded into one of the bank0 page buffers. In these "buffer hit" cases, read data is returned to the AHB data phase with a 0 wait state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses that “hit” in this register are also serviced with a 0 wait state response.

17.2.10 Write cycles

In a write cycle, address, write data, and control signals are launched off the same edge of hclk at the completion of the first AHB data phase cycle. Write cycles to the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0], driving write data on bk*n*_fl_wdata[63:0], and asserting bk*n*_fl_wr_en. Again, the controller drives the address and control information for the required setup time before the rising edge of hclk, and provides the required amount of hold time. The platform Flash controller then waits for the appropriate number of write wait states before terminating the write operation. On the cycle following the programmed wait state value, the platform Flash controller asserts hready_out to indicate to the AHB port that the cycle has terminated.

17.2.11 Error termination

The platform Flash controller follows the standard procedure when an AHB bus cycle is terminated with an ERROR response. First, the platform Flash controller asserts hresp[0] and negates hready_out to signal an error has occurred. On the following clock cycle, the platform Flash controller asserts hready_out and holds both hresp[0] and hready_out asserted until hready_in is asserted.

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform Flash controller does not initiate a Flash array access.

The second case that can cause an error response to the AHB is when an access is performed to the Flash array and is terminated with a Flash error response. See *Section 17.2.13, “Flash error response operation.* This may occur for either a read or a write operation.

The third case that can cause an error response to the AHB is when a write access is attempted to the Flash array and is disallowed by the state of the bk*n*_fl_ary_access control input. This case is similar to case 1.

A fourth case involves an attempted read access while the Flash array is busy doing a write (program) or erase operation if the appropriate read-while-write control field is programmed for this response. The 3-bit read-while-write control allows for immediate termination of an attempted read, or various stall-while-write/erase operations are occurring.

The platform Flash controller can also terminate the current AHB access if hready_in is asserted before the end of the current bus access. While this circumstance should not occur, this does not result in an error condition being reported, as this behavior is initiated by the AHB. In this circumstance, the platform Flash controller control state machine completes any Flash array access in progress (without signaling the AHB) before handling a new access request.

17.2.12 Access pipelining

The platform Flash controller does not support access pipelining since this capability is not supported by the Flash array. As a result, the APC (Address Pipelining Control) field should typically be the same value as the RWSC (Read Wait State Control) field for best performance, that is, BK*n*_APC = BK*n*_RWSC. It cannot be less than the RWSC.

17.2.13 Flash error response operation

The Flash array may signal an error response by asserting bk*n*_fl_xfr_err to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform Flash controller does not update or validate a bank0 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For more information on the specifics related to signaling of errors, including Flash ECC, refer to subsequent sections in this chapter. For additional information on the system registers that capture the faulting address, attributes, data and ECC information, see *15, "Error Correction Status Module (ECSM).*

17.2.14 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform Flash controller contains four 128-bit page read buffers that hold data read from the Flash array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

The organization of each page buffer is described as follows in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags.

```
struct {        // bk0_page_buffer
     reg  addr[23:4];// page address
     reg  valid;     // valid bit
     reg  rdata[127:0];// page read data
     reg  xfr_error; // transfer error indicator from Flash array
     reg  multi_ecc_error;// multi-bit ECC error indicator from Flash array
     reg  single_ecc_error;// single-bit correctable ECC indicator from
Flash array
}    bk0_page_buffer[4];
```

For the general case, a page buffer is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that "hit" the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in *Section 17.2.13, "Flash error response operation* a page buffer is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see *Section , "Buffer invalidation.*

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted based on whether a read access is for instruction or data. A read access to the platform Flash controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a Flash array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in prioritized order:

1. Invalid—the buffer contains no valid data.
2. Used—the buffer contains valid data that has been provided to satisfy an AHB burst type read.
3. Valid—the buffer contains valid data that has been provided to satisfy an AHB single type read.
4. Prefetched—the buffer contains valid data that has been prefetched to satisfy a potential future AHB access.
5. Busy AHB—the buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill—the buffer has been allocated to receive data from the Flash array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate page buffer has been selected, the Flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of Flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted Flash access.

Several algorithms are available for prefetch control that trade off performance versus power. They are defined by the B*x*_P*y*_PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (B*x*_P*y*_BFE) must be set, the prefetch limit (B*x*_P*y*_PFLM) must be non-zero and either instruction prefetching (B*x*_P*y*_IPFE) or data prefetching (B*x*_P*y*_DPFE) enabled. Refer to *Section 17.3.6, “Registers description* for a description of these control fields.

Instruction/data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the B*x*_P*y*_IPFE control field, while prefetching for data reads is enabled via the B*x*_P*y*_DPFE control field. Additionally, the B*x*_P*y*_PFLIM field must also be set to enable prefetching. Prefetches are never triggered by write cycles.

Per-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. AHB accesses indicate the requesting master via the hmaster[3:0] inputs. Refer to *Section , “Platform Flash Access Protection Register (PFAPR)* for details on these controls.

Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration (B*x*_P*y*_BCFG) field. This field defines the operating organization of the four page buffers. The buffers can be organized as a “pool” of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1, and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

Any falling edge transition of the array’s bk*n*_fl_done signal causes the page read buffers to be marked as invalid. This input is negated by the Flash array at the beginning of all program/erase operations as well as in certain other cases. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer in progress.

Software may invalidate the buffers by clearing the B*x*_P*y*_BFE bit, which also disables the buffers. Software may then re-assert the B*x*_P*y*_BFE bit to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer “hits” on Flash data that was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes an status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the platform Flash controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

Finally, the buffers are invalidated by hardware on any non-sequential access with a non-zero value on haddr[28:24] to support wait state emulation.

17.2.15 Bank1 temporary holding register

Recall the bank1 logic within the Flash includes a single 128-bit data register, used for capturing read data. Since this bank does not support prefetching, the read data for the referenced address is bypassed directly back to the AHB data bus. The page is also loaded into the temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (B1_Py_BFE).

The organization of the temporary holding register is described as follows, in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags and is the same as an individual bank0 page buffer.

```
struct {        // bk1_page_buffer
     reg  addr[23:4];// page address
     reg  valid;     // valid bit
     reg  rdata[127:0];// page read data
     reg  xfr_error; // transfer error indicator from Flash array
     reg  multi_ecc_error;// multi-bit ECC error indicator from Flash array
```

```
     reg  single_ecc_error;// single-bit correctable ECC indicator from
Flash array
}    bk1_page_buffer;
```

For the general case, a temporary holding register is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait states as the stored read data is routed from the temporary register back to the requesting bus master.

The contents of the holding register are invalidated by the falling edge transition of bk1_fl_done and on any non-sequential access with a non-zero value on haddr[28:24] (to support wait state emulation) in the same manner as the bank0 page buffers. Additionally, the B1_Py_BFE register bit can be cleared by software to invalidate the contents of the holding register.

As noted in *Section 17.2.13, “Flash error response operation* the temporary holding register is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register “hits” on Flash data that was tagged with a single-bit ECC event. Depending on the specific hardware configuration, the reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

The bank1 temporary holding register effectively operates like a single page buffer.

17.2.16 Read-While-Write functionality

The platform Flash controller supports various programmable responses for read accesses while the Flash is busy performing a write (program) or erase operation. For all situations, the platform Flash controller uses the state of the Flash array’s bk*n*_fl_done output to determine if it is busy performing some type of high-voltage operation, namely, if bk*n*_fl_done = 0, the array is busy.

Specifically, there are two 3-bit read-while-write (BK*n*_RWWC) control register fields that define the platform Flash controller’s response to these types of access sequences. There are five unique responses that are defined by the BK*n*_RWWC setting: one immediately reports an error on an attempted read, and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- BK*n*_RWWC = 0b0*xx*
  - For this mode, any attempted Flash read to a busy array is immediately terminated with an AHB error response and the read is blocked in the controller and not seen by the Flash array.
- BK*n*_RWWC = 0b111
  - This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform Flash controller module stalls any read reference until the Flash has completed its program/erase operation. If a read access arrives while the array is busy or if a falling-edge on bk*n*_fl_done occurs while a read is still in progress, the AHB data phase is stalled by negating hready_out and saving the address and attributes into holding registers. Once the array has completed its program/erase operation, the platform Flash controller uses the saved address and attribute information to create a pseudo address

phase cycle to “retry” the read reference and sends the registered information to the array as bk*n*_fl_rd_en is asserted. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus and hready_out negated to terminate the system bus transfer.

- BK*n*_RWWC = 0b110
  - This setting is similar to the basic stall-while-write capability provided when BK*n*_RWWC = 0b111 with the added ability to generate a notification interrupt if a read arrives while the array is busy with a program/erase operation. There are two notification interrupts, one for each bank.
- BK*n*_RWWC = 0b101
  - Again, this setting provides the basic stall-while-write capability with the added ability to terminate any program/erase operation if a read access is initiated. For this setting, the read request is captured and retried as described for the basic stall-while-write, plus the program/erase operation is terminated by the platform Flash controller’s assertion of the bkc_fl_abort signal. The bk*n*_fl_abort signal remains asserted until bk*n*_fl_done is driven high. For this setting, there are no notification interrupts generated.
- BK*n*_RWWC = 0b100
  - This setting provides the basic stall-while-write capability with the ability to terminate any program/erase operation if a read access is initiated plus the generation of a termination notification interrupt. For this setting, the read request is captured and retried as described for the basic stall-while-write, the program/erase operation is terminated by the platform Flash controller’s assertion of the bk*n*_fl_abort signal and a termination notification interrupt generated. There are two termination notification interrupts, one for each bank.

As detailed above, there are a total of four interrupt requests associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM’s Interrupt Register and logically summed together to form a single request to the interrupt controller.

Table 139. Platform Flash controller stall-while-write interrupts

MIR[*n*]	Interrupt description
ECSM.MIR[7]	Platform Flash bank0 termination notification, MIR[FB0AI]
ECSM.MIR[6]	Platform Flash bank0 stall notification, MIR[FB0SI]
ECSM.MIR[5]	Platform Flash bank1 termination notification, MIR[FB1AI]
ECSM.MIR[4]	Platform Flash bank1 stall notification, MIR[FB1S1]

For example timing diagrams of the stall-while-write and terminate-while-write operations, see *Figure 149* and *Figure 150* respectively.

17.2.17 Wait state emulation

Emulation of other memory array timings are supported by the platform Flash controller on read cycles to the Flash. This functionality may be useful to maintain the access timing for blocks of memory that were used to overlay Flash blocks for the purpose of system calibration or tuning during code development.

The platform Flash controller inserts additional wait states according to the values of haddr[28:24],where haddr represents the Flash address. When these inputs are non-zero, additional cycles are added to AHB read cycles. Write cycles are not affected. In addition, no page read buffer prefetches are initiated, and buffer hits are ignored.

Table 140 and *Table 141* show the relationship of haddr[28:24] to the number of additional primary wait states. These wait states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait state specification consists of two components: haddr[28:26] and haddr[25:24] and effectively extends the Flash read by (8 × haddr[25:24] + haddr[28:26]) cycles.

Table 140. Additional wait state encoding

Memory address haddr[28:26]	Additional wait states
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 141 shows the relationship of haddr[25:24] to the number of additional wait states. These are applied in addition to those specified by haddr[28:26] and thus extend the total wait state specification capability.

Table 141. Extended additional wait state encoding

Memory address haddr[25:24]	Additional wait states (added to those specified by haddr[28:26])
00	0
01	8
10	16
11	24

17.2.18 Timing diagrams

Since the platform Flash controller is typically used in platform configurations with a cacheless core, the operation of the processor accesses to the platform memories, for example Flash and SRAM, plays a major role in the overall system performance. Given the core/platform pipeline structure, the platform's memory controllers (PFlash, PRAM) are designed to provide a 0 wait state data phase response to maximize processor

performance. The following diagrams illustrate operation of various cycle types and responses referenced earlier in this chapter including stall-while-read (*Figure 149*) and terminate-while-read (*Figure 150*) diagrams.

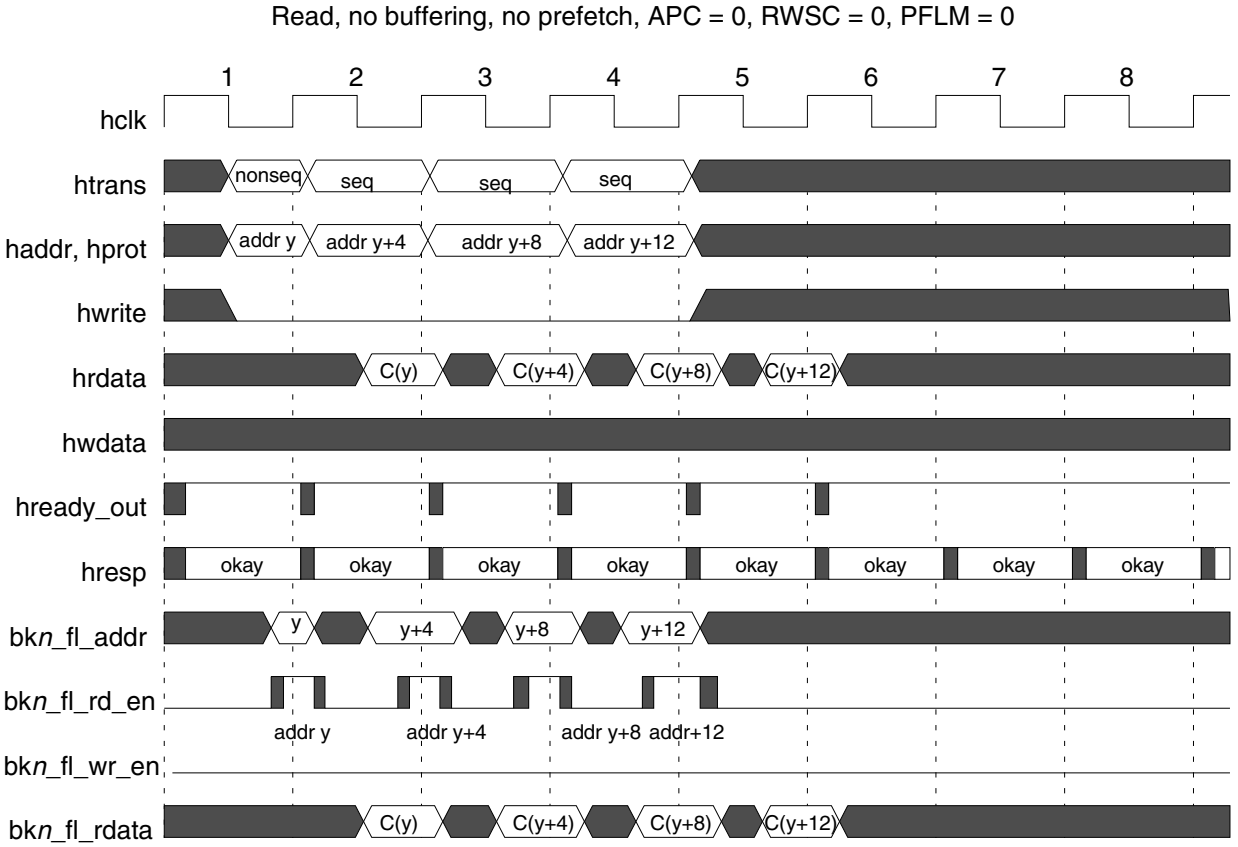


Figure 145. 1-cycle access, no buffering, no prefetch

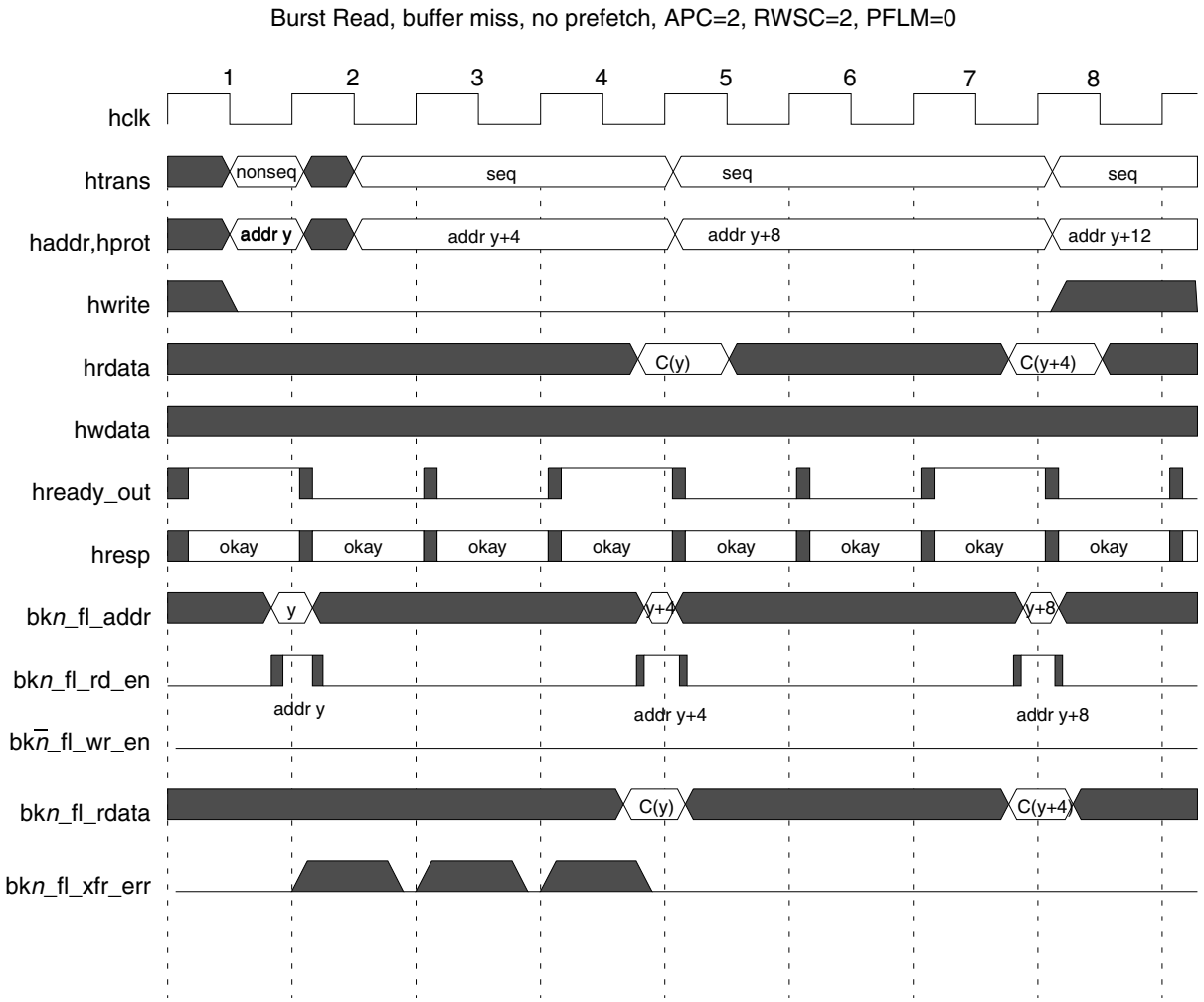


Figure 146. 3-cycle access, no prefetch, buffering disabled

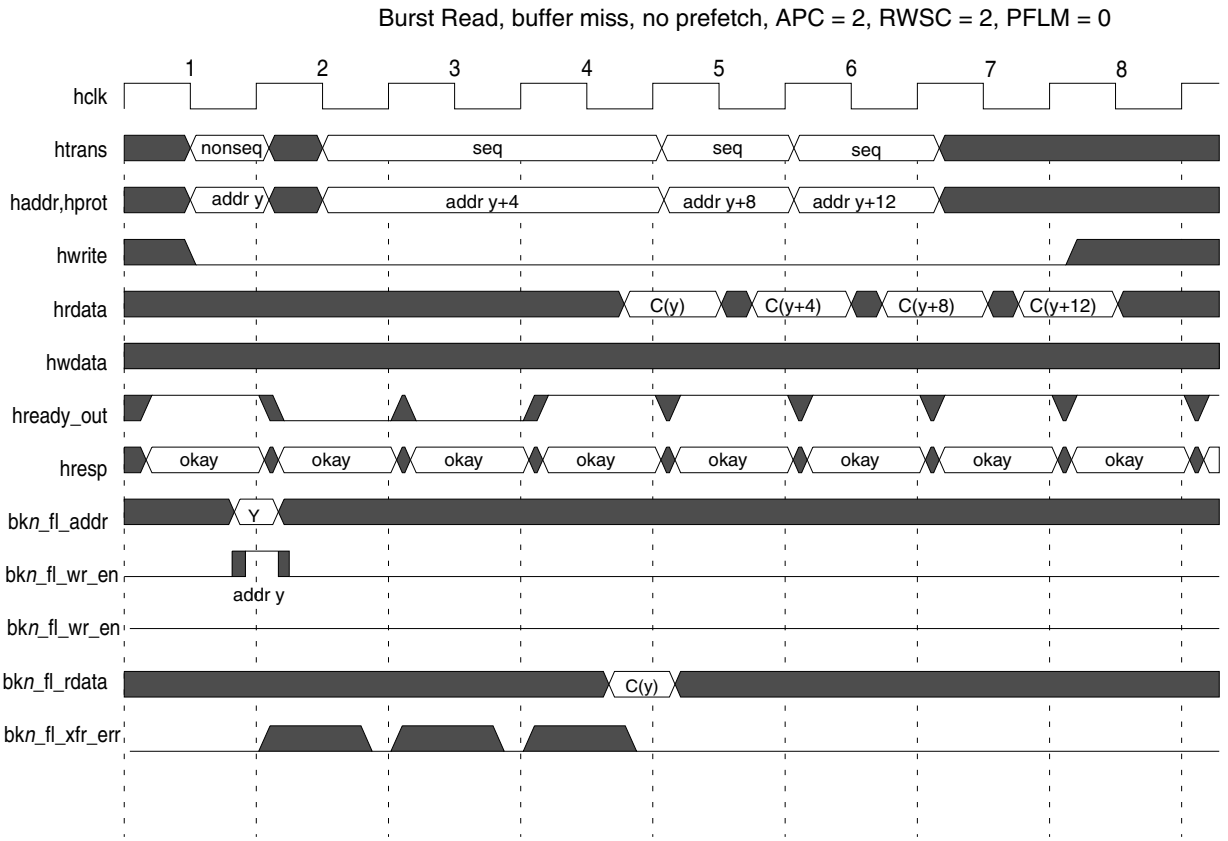


Figure 147. 3-cycle access, no prefetch, buffering enabled

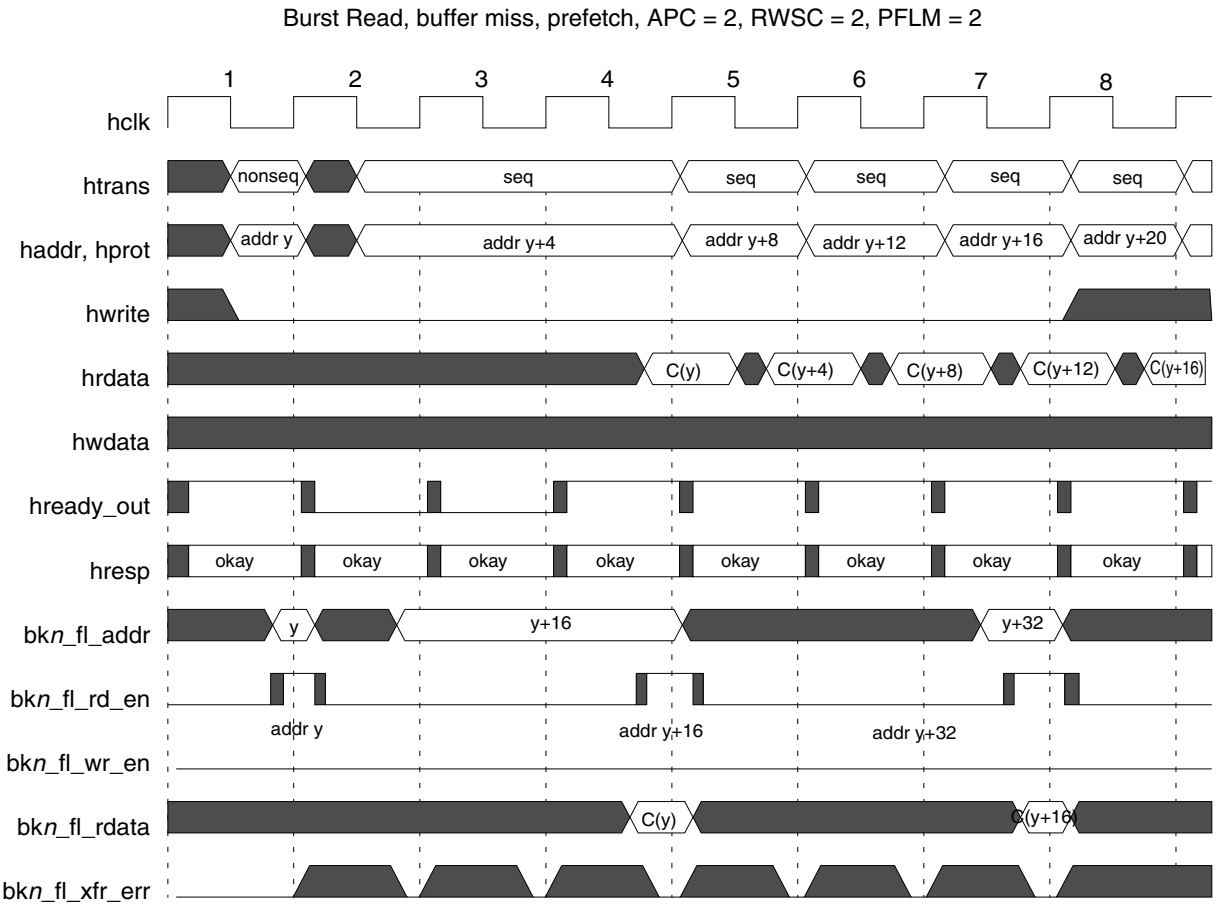


Figure 148. 3-cycle access, prefetch and buffering enabled

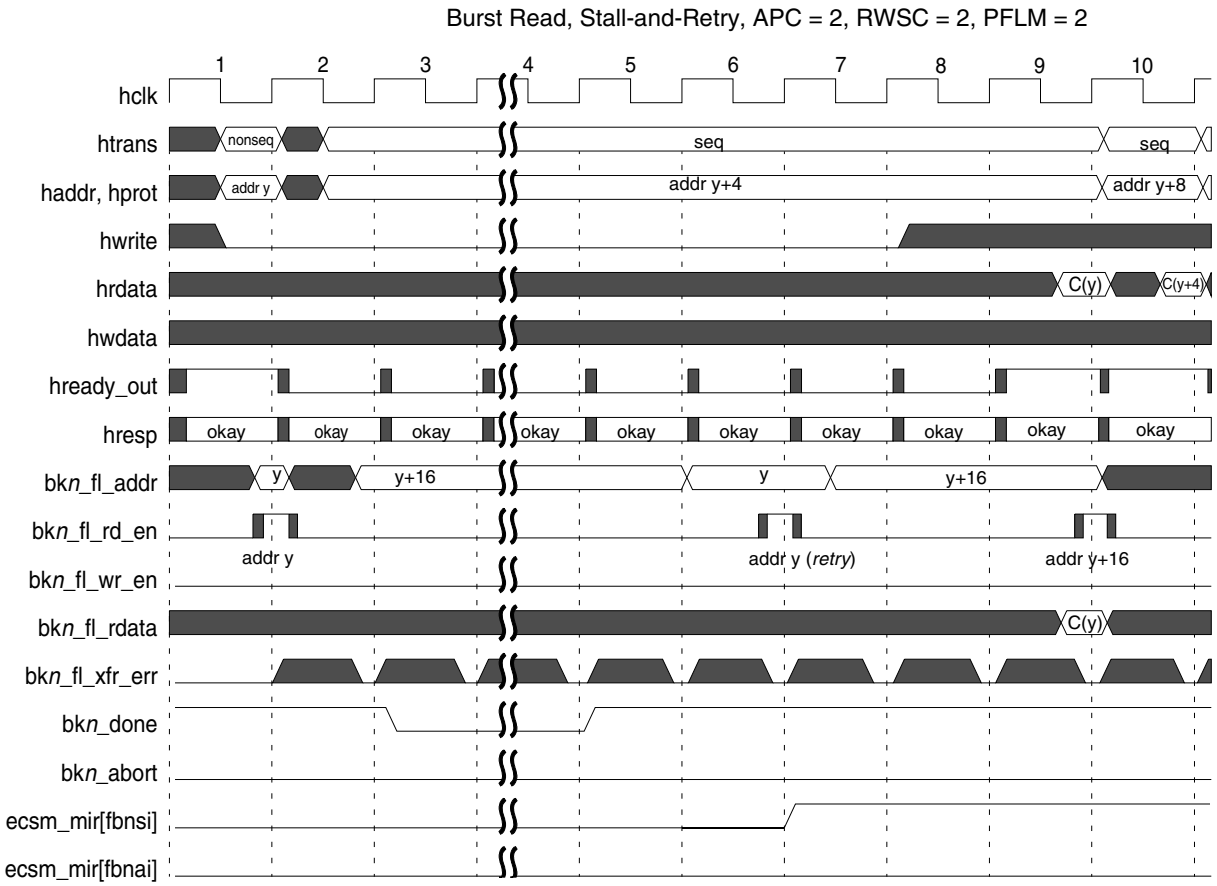


Figure 149. 3-cycle access, stall-and-retry with BK*n*_RWWC = 11*x*

As shown in *Figure 149*, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Eventually, this array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y + 4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal a stall notification interrupt (if BK*n*_RWWC = 110). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*SI] (Flash bank *n* stall interrupt).

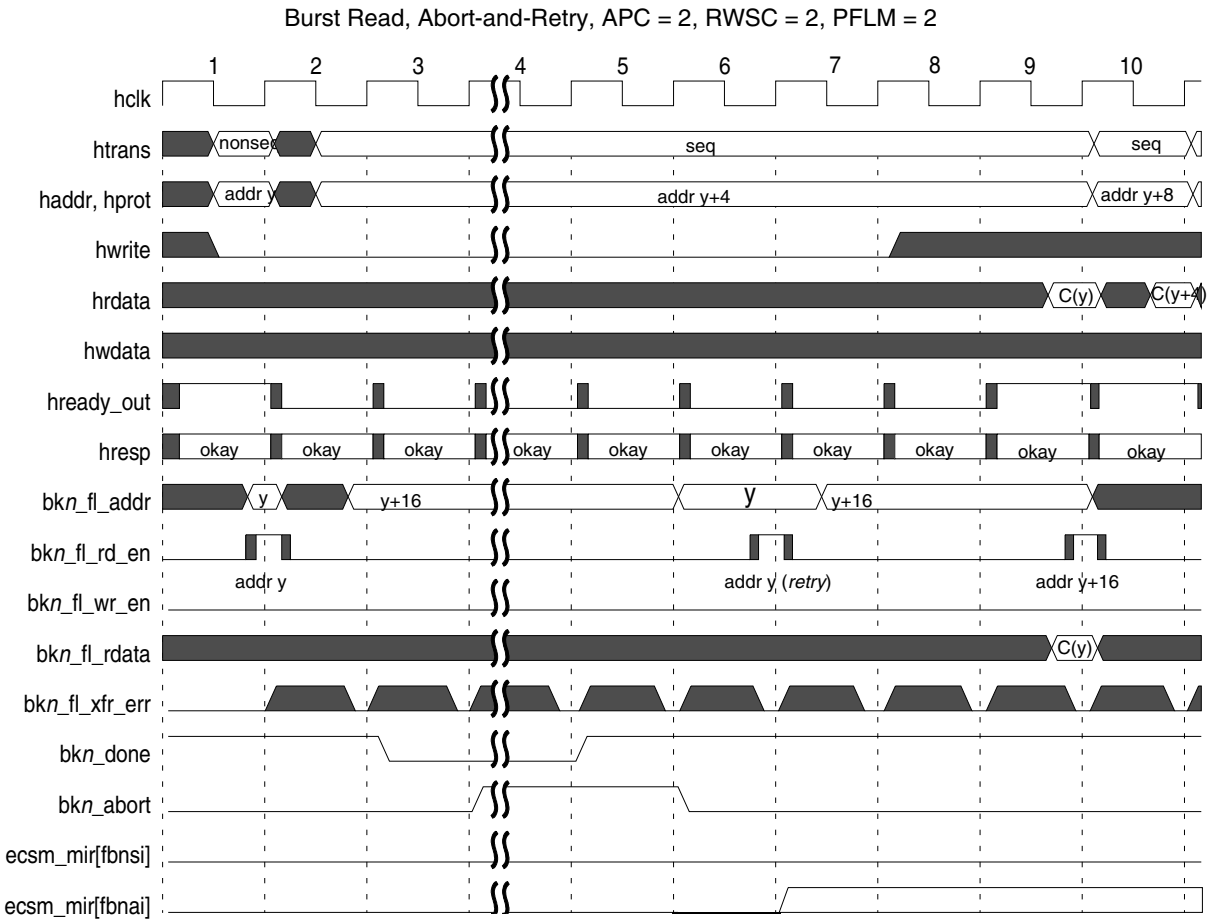


Figure 150. 3-cycle access, terminate-and-retry with BK*n*_RWWC = 10*x*

Figure 150 shows the terminate-while-write timing diagram. In this example, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Based on the setting of BK*n*_RWWC, once the bk*n*_done signal is detected as negated, the platform Flash controller asserts bk*n*_abort, which forces the Flash array to cancel the high-voltage program or erase event. The array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. It should be noted that the time spent in cycle 4 for *Figure 150* is considerably less than the time in the same cycle in *Figure 149* (because of the terminate operation). In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y+4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal an termination notification interrupt (if BK*n*_RWWC = 100). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*AI] (Flash bank *n* termination interrupt).

17.3 Flash memory

17.3.1 Introduction

The Flash module provides electrically programmable and erasable non-volatile memory (NVM), which may be used for instruction and/or data storage.

The Flash module is arranged as two functional units: the Flash core and the memory interface.

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

The memory interface contains the registers and logic which control the operation of the Flash core. The memory interface is also the interface between the Flash module and a bus interface unit (BIU), and may contain the ECC logic and redundancy logic. The BIU connects the Flash module to a system bus.

The SPC560P40/34 provides two Flash modules: one 256 KB code Flash module, and one 64 KB data module.

17.3.2 Main features

- High read parallelism (128 bits)
- Error Correction Code (SEC-DED) to enhance data retention
- Double word program (64 bits)
- Sector erase
- Single bank architecture
  - Read-While-Modify not available within an individual module
  - Read-While-Modify can be performed between the two Flash modules
- Erase suspend available (program suspend not available)
- Software programmable program/erase protection to avoid unwanted writes
- Censored mode against piracy
- Shadow Sector available on code Flash module

17.3.3 Block diagram

Data Flash

The data Flash 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 registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 151* shows the data Flash module structure.

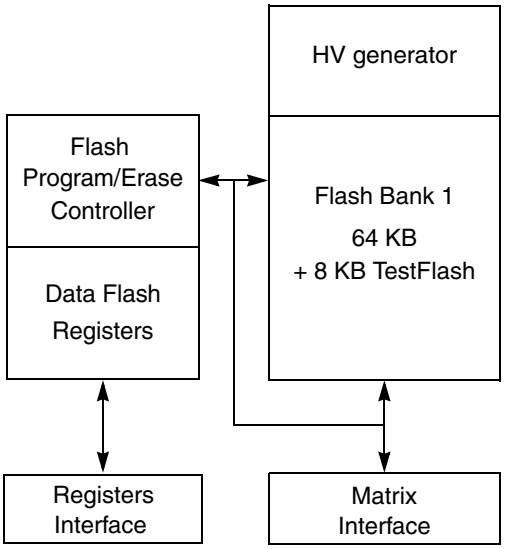


Figure 151. Data Flash module structure

Code Flash

The code Flash module contains the matrix modules 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 128 bits wide, while the Flash registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 152* shows the code Flash module structure.

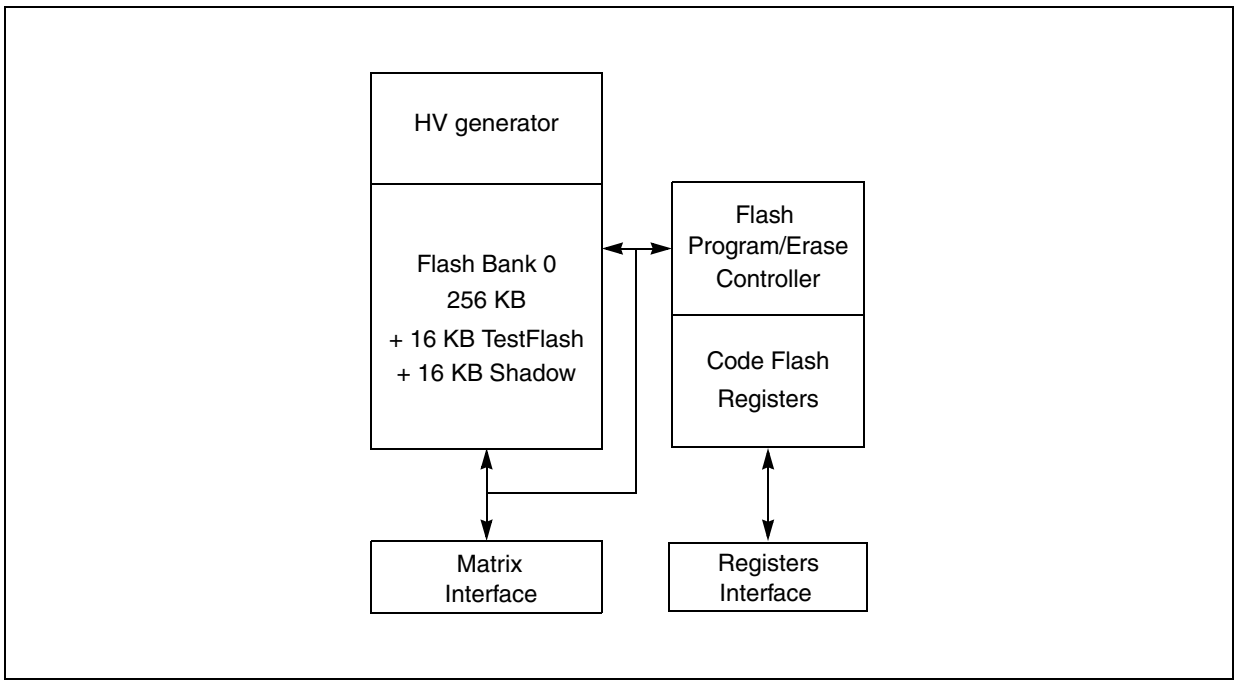


Figure 152. Code Flash module structure

17.3.4 Functional description

Macrocell structure

The Flash macrocell provides high density non-volatile memories with high-speed read access.

The Flash module is addressable by word (32 bits) or double-word (64 bits) for programming, and by page (128 bits) for reads. Reads done to the Flash always return 128 bits, although read page buffering may be done in the platform BIU.

Each read of the Flash 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 Flash page read architecture supports both cache and burst mode at the BIU level for high-speed read application.

The Flash module supports fault tolerance through Error Correction Code (ECC) and/or error detection. The ECC implemented within the Flash module will correct single bit failures and detect double bit failures.

The Flash module uses an embedded hardware algorithm implemented in the memory interface to program and erase the Flash 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 guarantee data integrity and reliability.

A programmed bit in the Flash module reads as logic level 0 (or low).

An erased bit in the Flash module reads as logic level 1 (or high).

Program and erase of the Flash module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be terminated.

Flash module sectorization

The code Flash module supports 256 KB of user memory, plus 16 KB of test memory (a portion of which is one-time programmable by the user). An extra 16 KB sector is available as Shadow space usable for user option bits or censorship.

The code Flash and data modules are each composed of a single bank: Bank 0 (code Flash) and Bank 1 (data Flash). Read-While-Modify within a module is not supported, but can be performed by reading from one module while writing to another.

The code Flash Bank 0 is divided in 8 sectors including a reserved sector named TestFlash, in which One Time Programmable (OTP) user data are stored, and a Shadow Sector in which user erasable configuration values can be stored (see *Table 142*).

The data Flash Bank 1 is divided in five sectors including a reserved sector named TestFlash (see *Table 143*).

Table 142. 288 KB code Flash module sectorization

Bank	Sector	Addresses	Size	Address space
B0	B0F0	0x0000_0000–0x0000_7FFF	32 KB	Low Address Space
B0	B0F1	0x0000_8000–0x0000_BFFF	16 KB	Low Address Space
B0	B0F2	0x0000_C000–0x0000_FFFF	16 KB	Low Address Space
B0	B0F3	0x0001_0000–0x0001_7FFF	32 KB	Low Address Space
B0	B0F4	0x0001_8000–0x0001_FFFF	32 KB	Low Address Space
B0	B0F5	0x0002_0000–0x0003_FFFF	128 KB	Low Address Space
B0	Reserved	0x0004_0000–0x0007_FFFF	256 KB	Mid Address Space
B0	Reserved	0x0008_0000–0x001F_FFFF	1536 KB	High Address Space
B0	B0SH	0x0020_0000–0x0020_3FFF	16 KB	Shadow Address Space
B0	Reserved	0x0020_4000–0x003F_FFFF	2032 KB	Shadow Address Space
B0	B0TF	0x0040_0000–0x0040_3FFF	16 KB	Test Address Space
B0	Reserved	0x0040_4000–0x007F_FFFF	4080 KB	Test Address Space

Table 143. 64 KB data Flash module sectorization

Bank	Sector	Addresses	Size (KB)	Address space
B1	B1F0	0x0080_0000 to 0x0080_3FFF	16	Low Address Space
B1	B1F1	0x0080_4000 to 0x0080_7FFF	16	Low Address Space
B1	B1F2	0x0080_8000 to 0x0080_BFFF	16	Low Address Space
B1	B1F3	0x0080_C000 to 0x0080_FFFF	16	Low Address Space

Table 143. 64 KB data Flash module sectorization (continued)

Bank	Sector	Addresses	Size (KB)	Address space
B1	Reserved	0x0081_0000 to 0x00C0_1FFF	4040	Reserved
B1	B1TF	0x00C0_2000 to 0x00C0_3FFF	8	Test Address Space
B1	Reserved	0x00C0_4000 to 0x00FF_FFFF	4080	Reserved

Each Flash module is divided into blocks to implement independent program/erase protection. A software mechanism is provided to independently lock/unlock each block in address space against program and erase.

TestFlash block

The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent TestFlash block is included also to support systems that require non-volatile memory for security and/or to store system initialization information.

A section of the TestFlash is reserved to store the non-volatile information related to redundancy, configuration, and protection.

ECC is also applied to TestFlash. The usage of reserved TestFlash sectors is detailed in *Table 144*.

Table 144. TestFlash structure

Name	Description	Addresses		Size (bytes)
		Code TestFlash	Data TestFlash	
—	Reserved	0x0040_0000–0x0040_1FFF	—	8192
—	Reserved	0x0040_2000–0x0040_3CFF	0x00C0_2000– 0x00C0_3CFF	7424
—	User Reserved	0x0040_3D00–0x0040_3DE7	0x00C0_3D00–0x00C0_3DE7	232
NVLML	Non-volatile Low/Mid address space block Locking register	0x0040_3DE8–0x0040_3DEF	0x00C0_3DE8–0x00C0_3DEF	8
—	User Reserved	0x0040_3DF0–0x0040_3DF7	0x00C0_3DF0–0x00C0_3DF7	8
NVSLL	Non-volatile Secondary Low/mid add space block Lock register	0x0040_3DF8–0x0040_3DFF	0x00C0_3DF8–0x00C0_3DFF	8
—	User Reserved	0x0040_3E00–0x0040_3EFF	0x00C0_3E00–0x00C0_3EFF	256
—	Reserved	0x0040_3F00–0x0040_3FFF	0x00C0_3F00–0x00C0_3FFF	256

Erase of the TestFlash block is always locked.

TestFlash block programming restrictions, in terms of how ECC is calculated, are similar to array programming restrictions. Only one program is allowed per 64-bit ECC segment.

Locations of the Code TestFlash block marked as reserved cannot be programmed by the user application. Locations of the Data TestFlash block marked as reserved cannot be programmed by the user application.

Shadow block

A Shadow block is present in each Code flash module, but not in the Data flash 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 set.

The Shadow block may be locked/unlocked against program or erase by using 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 sectors erase.

The Shadow block contains specified data that are needed for user features.

The user area of Shadow block may be used for user-defined functions (possibly to store boot code, other configuration words, or factory process codes).

The usage of the Shadow sector is detailed in *Table 145*.

Table 145. Shadow sector structure

Name	Description	Addresses	Size (bytes)
—	User Area	0x0020_0000–0x0020_3DCF	15824
—	Reserved	0x0020_3DD0–0x0020_3DD7	8
NVPWD0–1	Non-volatile private censorship password 0–1 registers	0x0020_3DD8–0x0020_3DDF	8
NVSCI0–1	Non-volatile system censorship information 0–1 registers	0x0020_3DE0–0x0020_3DE7	8
—	Reserved	0x0020_3DE8–0x0020_3DFF	24
NVBIU2–3	Non-volatile bus interface unit 2–3 registers	0x0020_3E00–0x0020_3E0F	16
—	Reserved	0x0020_3E10–0x0020_3E17	8
NVUSRO	Non-volatile user options register	0x0020_3E18–0x0020_3E1F	8
—	Reserved	0x0020_3E20–0x0020_3FFF	480

17.3.5 Operating modes

The following operating modes are available in the Flash module:

- Reset
- User mode
- Low-power mode
- Power-down mode

Reset

A reset is the highest priority operation for the Flash module and terminates all other operations.

The Flash module uses reset to initialize registers and status bits to their default reset values.

If the Flash 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 module into User mode ready to receive accesses.

Reset and power-down must not be used as a systematic way to terminate a program or erase operation.

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

User mode

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

The default state of the Flash module is the read state.

The main, shadow, and test address space can be read only in the read state.

The Flash registers are always available for reads. When the module is in power-down mode, most (but not all) registers are available for reads. The exceptions are documented.

The Flash 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 set and MCR[PGM] is cleared (Erase Suspend).

No Read-While-Modify is available within an individual module, although one module can be read while the other is being written or otherwise modified.

Flash core reads return 128 bits (1 page = 2 double words).

Register reads return 32 bits (1 word).

Flash core reads are done through the BIU.

Register reads to unmapped register address space return all 0s.

Register writes to unmapped register address space have no effect.

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

Simultaneous read cycles on the code Flash block and read/write cycles on the data Flash block are possible. However, simultaneous read/write accesses within a single block are not permitted.

Chip Select, Write Enable, addresses, and data input of registers are not internally latched and must be kept stable by the CPU for all the read/write access that lasts two clock cycles.

Low-power mode

The Low-power mode turns off most of the DC current sources within the Flash module.

The module (Flash core and registers) is not accessible for read or write operations once it has entered Low-power mode.

Wake-up time from Low-power mode is faster than wake-up time from Power-down mode.

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

When exiting from Low-power mode, the Flash module returns to its previous state in all cases, unless it was in the process of executing an erase high voltage operation at the time of entering Low-power mode.

If the Flash module is put into Low-power mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits from Low-power mode by clearing MCR[ESUS]. MCR[EHV] must be set to resume the erase operation.

If the Flash module is put in Low-power mode during a program operation, the operation is completed in all cases. Low-power mode is entered only after the programming ends.

Power-down mode cannot be entered when Low-power mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

Power-down mode

The Power-down mode allows turning off all Flash DC current sources so that power dissipation is limited only to leakage in this mode.

In Power-down mode, no reads from or writes to the Flash are possible.

When enabled, the Flash module returns to its previous state in all cases, unless in the process of executing an erase high voltage operation at the time of entering Power-down mode. If the Flash module is put into Power-down mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits Power-down mode by clearing the MCR[ESUS] bit. MCR[EHV] must be set to resume the erase operation.

If the Flash module is placed in Power-down mode during a program operation, the operation will be completed in any case and the Power-down mode is entered only after the programming is completed.

If the Flash module is put in Power-down mode and the vector table remains mapped in the Flash address space, the user must observe than the Flash module will require a longer interrupt response time. This should be accomplished by adding several wait states.

Low-power mode cannot be entered when Power-down mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

17.3.6 Registers description

The Flash user registers mapping is shown in *Table 146*. Except as noted, registers and offsets are identical for the code Flash and data Flash blocks.

Table 146. Flash registers

Offset from xxxx_BASE (0xFFFE_C000)	Register	Location
0x0000		*on page 17-342*
0x0004	Low/mid Address Space Block Locking Register (LML)	*on page 17-346*
0x0008	Reserved	
0x000C	Secondary Low/mid Address Space Block Lock Register (SLL)	*on page 17-349*
0x0010	Low/mid Address Space Block Select Register (LMS)	*on page 17-351*
0x0014	Reserved	
0x0018	Address Register (ADR)	*on page 17-352*
0x001C	Platform Flash Configuration Register 0 (PFCR0)(1)	*on page 17-353*
0x0020	Platform Flash Configuration Register 1 (PFCR1)(1)	*on page 17-357*
0x0024	Platform Flash Access Protection Register (PFAPR)(1)	*on page 17-359*
0x0028	Reserved	
0x003C	User Test Register 0 (UT0)	*on page 17-360*
0x0040	User Test Register 1 (UT1)	*on page 17-362*
0x0044	User Test Register 2 (UT2)(1)	*on page 17-363*
0x0048	User Multiple Input Signature Register 0 (UMISR0)	*on page 17-363*
0x004C	User Multiple Input Signature Register 1 (UMISR1)	*on page 17-364*
0x0050	User Multiple Input Signature Register 2 (UMISR2)(1)	*on page 17-365*
0x0054	User Multiple Input Signature Register 3 (UMISR3)(1)	*on page 17-365*
0x0058	User Multiple Input Signature Register 4 (UMISR4)(1)	*on page 17-366*
0x005C–0x3FFF	Reserved	

1. This register is not implemented on the data Flash block.

17.3.7 Register map

Table 147. Flash 256 KB bank0 register map

Address offset	Register 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
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV

Table 147. Flash 256 KB bank0 register map (continued)

Address offset	Register 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
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	0	0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	Reserved																
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	0	0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	Reserved																
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C	PFCR0	BI031	BI030	BI029	BI028	BI027	BI026	BI025	BI024	BI023	BI022	BI021	BI020	BI019	BI018	BI017	BI016
		BI015	BI014	BI013	BI012	BI011	BI010	BI009	BI008	BI007	BI006	BI005	BI004	BI003	BI002	BI001	BI000
0x20	PFCR1	BI131	BI130	BI129	BI128	BI127	BI126	BI125	BI124	BI123	BI122	BI121	BI120	BI119	BI118	BI117	BI116
		BI115	BI114	BI113	BI112	BI111	BI110	BI109	BI108	BI107	BI106	BI105	BI104	BI103	BI102	BI101	BI100
0x24	PFAPR	BI231	BI230	BI229	BI228	BI227	BI226	BI225	BI224	BI223	BI222	BI221	BI220	BI219	BI218	BI217	BI216
		BI215	BI214	BI213	BI212	BI211	BI210	BI209	BI208	BI207	BI206	BI205	BI204	BI203	BI202	BI201	BI200
0x28	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
0x44	UT2	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
		DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50	UMISR2	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
		MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064
0x54	UMISR3	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
		MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096
0x58	UMISR4	MS159	MS158	MS157	MS156	MS155	MS154	MS153	MS152	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
		MS143	MS142	MS141	MS140	MS139	MS138	MS137	MS136	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128

Table 148. Flash 64 KB bank1 register map

Address offset	Register 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
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	0	0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	Reserved																
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	0	0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	Reserved																
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C–0x3B	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
0x44–0x47	Reserved																
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50–0x5B	Reserved																

In the following sections, some non-volatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash or Shadow sectors with a special meaning.

During the Flash initialization phase, the FPEC reads these non-volatile registers and updates their related volatile registers. When the FPEC detects ECC double errors in these special locations, it behaves in the following way:

- In case of a failing system locations (configurations, device options, redundancy, EmbAlgo firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, BIU, ...), the volatile registers are filled with all 1s and the Flash initialization ends by clearing the MCR[PEG] bit.

Module Configuration Register (MCR)

The Module Configuration Register enables and monitors all the modify operations of each Flash module. Identical MCRs are provided in the code Flash and the data Flash blocks.

Figure 153. Module Configuration Register (MCR)

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	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
W	r1c															
Reset	0	0	0	0	0	—(1)	—(1)	—(1)	0	—(1)	1	0	0	0	0	0

	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	r1c	r1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

1. The value for this bit is different between the code and data Flash modules. See the bitfield description.

Table 149. MCR field descriptions

Field	Description
EDC 0	ECC Data Correction EDC provides information on previous reads. If a ECC Single 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 will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Double Error detection, this bit is not set. If EDC is not set, or remains 0, this 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 will have no effect. 0 Reads are occurring normally. 1 An ECC Single Error occurred and was corrected during a previous read.
1:4	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
SIZE[2:0] 5:7	Array space SIZE 2–0 The value of SIZE field depends on the size of the Flash module: 000 128 KB 001 256 KB (the value for the SPC560P40/34 device in the code Flash module) 010 512 KB 011 Reserved (1024 KB) 100 Reserved (1536 KB) 101 Reserved (2048 KB) 110 64 KB (the value for the device in the data Flash module) 111 Reserved The value for this bitfield is different between the code and data Flash modules.
8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 149. MCR field descriptions (continued)

Field	Description
LAS[2:0] 9:11	Low Address Space 2–0 The value of the LAS field corresponds to the configuration of the Low Address Space: 000 Reserved 001 Reserved 010 32 KB + (2 × 16 KB) + (2 × 32 KB) + 128 KB (the value for the SPC560P40/34device in the code Flash module) 011 Reserved 100 Reserved 101 Reserved 110 4 × 16 KB (the value for the SPC560P40/34device in the data Flash module) 111 Reserved The value for this bitfield is different between the code and data Flash modules.
12:14	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MAS 15	Mid Address Space The value of the MAS field corresponds to the configuration of the Mid Address Space: 0 0 KB or 2 × 128 KB 1 Reserved
EER 16	ECC Event Error EER provides information on previous reads. When an ECC Double Error detection occurs, the EER bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Single Error detection and correction, this bit will not be set. If EER is not set, or remains 0, this 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. A write of 0 will have no effect. 0 Reads are occurring normally. 1 An ECC Double Error occurred during a previous read.
RWE 17	Read-while-Write event Error RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the Flash 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 will return to a 0 state. This bit may not be set to 1 by the user. If RWE is not set, or remains 0, this 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 will have no effect. 0 Reads are occurring normally. 1 A RWW Error occurred during a previous read. If stall/terminate-while-write is used, the software should ignore the setting of the RWE flag and should clear this flag after each erase operation. If stall/terminate-while-write is not used, software can handle the RWE error normally.
18:19	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 149. MCR field descriptions (continued)

Field	Description
PEAS 20	Program/Erase Access Space PEAS 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 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 21	Modify Operation Done DONE indicates if the Flash module is performing a high voltage operation. DONE is set to 1 on termination of the Flash module reset. 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. DONE is set to 1 at the end of program and erase high voltage sequences. DONE is set to 1 (within t_{PABT} or t_{EABT}, equal to P/E Abort Latency) after a 1-to-0 transition of EHV, which terminates a high voltage program/erase operation. DONE is set to 1 (within t_{ESUS}, time equal to Erase Suspend Latency) after a 0-to-1 transition of ESUS, which suspends an erase operation. 0 Flash is executing a high voltage operation. 1 Flash is not executing a high voltage operation.
PEG 22	Program/Erase Good The PEG bit indicates the completion status of the last Flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the Flash module is reset, unless a Flash 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 a termination or the completion of a program/erase 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 are attempted on blocks that are locked, the response is PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation. If a program operation tries to program at 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 Margin Mode, 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 UMIRS[0:1]. 0 Program or erase operation failed; or program, erase, Array Integrity Check, or Margin Mode was terminated. 1 Program or erase operation successful; or Array Integrity Check or Margin Mode completed successfully.

Table 149. MCR field descriptions (continued)

Field	Description
23:26	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
PGM 27	Program PGM sets up the Flash 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. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence.
PSUS 28	Program Suspend Writing to this bit has no effect, but the written data can be read back.
ERS 29	Erase ERS sets up the Flash 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. ERS is cleared on reset. 0 Flash is not executing an Erase sequence. 1 Flash is executing an Erase sequence.
ESUS 30	Erase Suspend ESUS indicates that the Flash module is in Erase Suspend or in the process of entering a Suspend state. The Flash 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 in erase suspend. The Flash 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 module cannot exit Erase Suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.

Table 149. MCR field descriptions (continued)

Field	Description
EHV 31	Enable High Voltage The EHV bit enables the Flash module for a high voltage program/Erase operation. EHV is cleared on reset. 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 (ERS = 1, ESUS = 0, UT0[AIE] = 0) – Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0) In normal operation, a 1-to-0 transition of EHV with DONE high and 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 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 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 ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low. 0 Flash is not enabled to perform an high voltage operation. 1 Flash 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 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. *Table 150* shows the bit changing priorities.

Table 150. 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.

Low/Mid Address Space Block Locking register (LML)

The Low/Mid Address Space Block Locking register 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. Identical LML registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the LML register has a related Non-Volatile Low/Mid Address Space Block Locking register (NVLML) located in TestFlash that contains the default reset value for LML. The NVLML register is read during the reset phase of the Flash module and loaded into the LML. The reset value is 0x00XX_XXXX, initially determined by the NVLML value from test sector.

Figure 154. Low/Mid Address Space Block Locking register (LML)

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	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

Figure 155. Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

Address: Base + 0x40_3DE8 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	TSLK	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK	LLK
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The NVLML register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVLML registers are provided in the code Flash and the data Flash blocks.

Table 151. LML and NVLML field descriptions

Field	Description
LME(1) 0	Low/Mid Address Space Block Enable This bit enables the Lock registers (TSLK and LLK[15:0]) to be set or cleared by registers 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 0xA1A11111 must be written to the LML register. 0 Low Address Locks are disabled: TSLK and LLK[15:0] cannot be written. 1 Low Address Locks are enabled: TSLK and LLK[15:0] can be written.
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 151. LML and NVLML field descriptions (continued)

Field	Description
TSLK 11	Test/Shadow Address Space Block Lock This bit locks the block of Test and Shadow 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/Shadow block is locked for program and Erase. A value of 0 in the TSLK register signifies that the Test/Shadow 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. Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also SLL[STSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
14:15	Reserved
LLK[15:0] 16:31	Low Address Space Block Lock 15-0 These bits lock the blocks of Low Address Space from program and Erase. For code Flash, LLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 142* for more information. For data Flash, LLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 143* for more information. A value of 1 in a bit of the LLK bitfield signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the LLK bitfield signifies that the corresponding block is available to receive program and Erase pulses. The LLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK bitfield is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK bitfields. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value is always 1 (independent of the TestFlash block), and register writes have no effect. In the code Flash macrocell, bits LLK[15:6] are read-only and locked at 1. In the data Flash macrocell, bits LLK[15:4] are read-only and locked at 1. 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.

1. This field is present only in LML

Secondary Low/Mid Address Space Block Locking register (SLL)

The Secondary Low/Mid Address Space Block Locking register 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. Identical SLL registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the SLL register has a related Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL) located in TestFlash that contains the default reset value for SLL. The reset value is 0x00XX_XXXX, initially determined by NVSLL.

The NVSLL register is read during the reset phase of the Flash module and loaded into the SLL.

Figure 156. Secondary Low/mid address space block Locking reg (SLL)

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 LK	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK 15	SLK 14	SLK 13	SLK 12	SLK 11	SLK 10	SLK 9	SLK 8	SLK 7	SLK 6	SLK 5	SLK 4	SLK 3	SLK 2	SLK 1	SLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

The NVSLL register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVSLL registers are provided in the code Flash and the data Flash blocks.

Figure 157. Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

Address: Base + 0x40_3DF8 — 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	STS LK	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK 15	SLK 14	SLK 13	SLK 12	SLK 11	SLK 10	SLK 9	SLK 8	SLK 7	SLK 6	SLK 5	SLK 4	SLK 3	SLK 2	SLK 1	SLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 152. SLL and NVSLL field descriptions

Field	Description
SLE[1] 0	Secondary Low/Mid Address Space Block Enable This bit enables the Lock registers (STSLK and SLK[15:0]) to be set or cleared by registers 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 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[15:0] cannot be written. 1 Secondary Low/Mid Address Locks are enabled: STSLK and SLK[15:0] can be written.
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
STSLK 11	Secondary Test/Shadow address space block LocK This bit is used as an alternate means to lock the block of Test and Shadow Address Space from program and Erase (Erase is any case forbidden for Test block). A value of 1 in the STSLK bitfield signifies that the Test/Shadow block is locked for program and Erase. A value of 0 in the STSLK register signifies that the Test/Shadow 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. Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also LML[TSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
14:15	Reserved

Table 152. SLL and NVSLL field descriptions (continued)

Field	Description
SLK[15:0] 16:31	Secondary Low Address Space Block Lock 15–0 These bits are used as an alternate means to lock the blocks of Low Address Space from program and Erase. For code Flash, SLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 142* for more information. For data Flash, SLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 143* for more information. A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and Erase pulses. The SLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset causes the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SLK bits default to locked, and are not writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the code Flash macrocell, bits SLK[15:6] are read-only and locked at 1. . In the data Flash macrocell, bits SLK[15:4] are read-only and locked at 1. 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.

1. This field is present only in SLL

Low/Mid Address Space Block Select register (LMS)

The Low/Mid Address Space Block Select register provides a means to select blocks to be operated on during erase. Identical LMS registers are provided in the code Flash and the data Flash blocks.

Figure 158. Low/Mid Address Space Block Select register (LMS)

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

Table 153. LMS field descriptions

Field	Description
0:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
14:15	Reserved
LSL[15:0] 16:31	Low Address Space Block Select 15–0 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. For code Flash, LSL[5:0] are related to sectors B0F[5:0], respectively. See *Table 142* for more information. For data Flash, LSL[3:0] are related to sectors B1F[3:0], respectively. See *Table 143* for more information. 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 LSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. In the code Flash macrocell, bits LSL[15:6] are read-only and locked at 0. In the data Flash macrocell, bits LSL[15:4] are read-only and locked at 0. 0 Low Address Space Block is unselected for Erase. 1 Low Address Space Block is selected for Erase.

Address Register (ADR)

The Address Register provides the first failing address in the event module failures (ECC, RWW, or FPEC) or the first address at which a ECC single error correction occurs.

Figure 159. Address Register (ADR)

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	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 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	AD 15	AD 14	AD 13	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 154. ADR field descriptions

Field	Description
0:8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 154. ADR field descriptions (continued)

Field	Description
AD[22:3] 9:28	Address 22–3 ADR 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). ADR also provides the first address at which a ECC single error correction occurs (MCR[EDC] set). The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error, and the ECC single error correction. When accessed ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in *Table 155*. This address is always a double word address that selects 64 bits. In case of a simultaneous ECC double error detection on both double words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC single error correction on both double words of the same page. In User mode, ADR is read only.
29:31	*Reserved* Write these bits has no effect and read these bits always outputs 0.

Table 155. ADR content: priority list

Priority level	Error flag	ADR content
1	MCR[EER] = 1	Address of first ECC Double Error
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 Error Correction

Platform Flash Configuration Register 0 (PFCR0)

The Platform Flash Configuration Register 0 (PFCR0) defines the configuration associated with Flash memory bank0, which corresponds to the code Flash. The register is described in *Figure 160* and *Table 156*.

Note: This register is not implemented on the data Flash block.

Figure 160. Platform Flash Configuration Register 0 (PFCR0)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BK0_APC				BK0_WWSC				BK0_RWSC							BK0_RWWC
W																
Reset	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BK0_RWWC	0	0	0	0	0	0	0	BK0_RWWC	B0_P0_BCFG		B0_P0_DPFE	B0_P0_IPFE	B0_P0_PFLM		B0_P0_BFE
W																
Reset	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1

Table 156. PFCR0 field descriptions

Field	Description
0-4 BK0_APC	Bank0 Address Pipelining Control This field controls the number of cycles between Flash 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 operation. This field is set to 0b00010 by hardware reset. 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.
5-9 BK0_WWSC	Bank0 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. 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 operation. This field is set to an appropriate value by hardware reset. This field is set to 0b00010 by hardware reset. 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.

Table 156. PFCR0 field descriptions (continued)

Field	Description
10-14 BK0_RWSC	Bank0 Read Wait State Control This field controls the number of wait states to be added to the Flash 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. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 23 MHz APC = RWSC = 0. 23 MHz, < 45 MHz APC = RWSC = 1. 45 MHz, < 68 MHz APC = RWSC = 2. 68 MHz, < 90 MHz APC = RWSC = 3. This field is set to 0b00010 by hardware reset. 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.
15-16,24 BK0_RWWC	Bank0 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation termination and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the 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. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.
17-23	Reserved

Table 156. PFCR0 field descriptions (continued)

Field	Description
25-26 B0_P0_BCFG	Bank0, 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 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 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 0,1,2 allocated for instruction fetches and buffer 3 for data accesses. This field is set to 2b11 by hardware reset.
27 B0_P0_DPFE	Bank0, Port 0 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 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.
28 B0_P0_IPFE	Bank0, Port 0 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. 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.
29-30 B0_P0_PFLM	Bank0, 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. This field is set to 2b10 by hardware reset. 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*.
31 B0_P0_BFE	Bank0, Port 0 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset. 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.

Platform Flash Configuration Register 1 (PFCR1)

The Platform Flash Configuration Register 1 (PFCR1) defines the configuration associated with Flash memory bank1. This corresponds to the data Flash. The register is described in *Figure 161* and *Table 157*.

Note: This register is not implemented on the data Flash block.

Figure 161. Platform Flash Configuration Register 1 (PFCR1)

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	BK1_APC				BK1_WWSC				BK1_RWSC				BK1_RWWC			
Reset	0	1	0	0	0	0	1	0	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	BK1_RWWC	0	0	0	0	0	0	B1_P1_BFE	BK1_RWWC	0	0	0	0	0	0	B1_P0_BFE
W																
Reset	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Table 157. PFCR1 field descriptions

Field	Description
0-4 BK1_APC	Bank1 Address Pipelining Control This field controls the number of cycles between Flash 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 operation. This field is set to 0b00010 by hardware reset. 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.
5-9 BK1_WWSC	Bank1 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. 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 operation. This field is set to an appropriate value by hardware reset. This field is set to 0b00010 by hardware reset. 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.

Table 157. PFCR1 field descriptions (continued)

Field	Description
10-14 BK1_RWSC	Bank1 Read Wait State Control This field controls the number of wait states to be added to the Flash 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. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 23 MHz APC = RWSC = 0. 23 MHz, < 45 MHz APC = RWSC = 1. 45 MHz, < 68 MHz APC = RWSC = 2. 68 MHz, < 90 MHz APC = RWSC = 3. This field is set to 0b00010 by hardware reset. 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.
15-16,24 BK1_RWWC	Bank1 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the 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. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.
17-22	Reserved, should be cleared.
23 B1_P1_BFE	Bank1, Port 1 Buffer Enable This bit enables or disables read hits from the 128-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.
25-30	Reserved, should be cleared.

Table 157. PFCR1 field descriptions (continued)

Field	Description
31 B1_P0_PFE	Bank1, Port 0 Buffer Enable This bit enables or disables read hits from the 128-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.

Platform Flash Access Protection Register (PFAPR)

The Platform Flash Access Protection Register (PFAPR) controls read and write accesses to the Flash based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode for controllers supporting two AHB ports. The register is described in *Figure 162* and *Table 158*.

The contents of the register are loaded from location 0x20_3E00 of the shadow region in the code Flash (bank0) array 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 0x20_3E00 of the shadow region in the Flash array must be programmed using the normal sequence of operations. The reset value shown in *Table 162* reflects an erased or unprogrammed value from the shadow region.

Note: This register is not implemented on the data Flash block.

Figure 162. Platform Flash Access Protection Register (PFAPR)

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	ARBM		M7 PFD	M6 PFD	M5 PFD	M4 PFD	M3 PFD	M2 PFD	M1 PFD	M0 PFD
W																
Reset	0	0	0	0	0	0	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	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 158. PFAPR field descriptions

Field	Description
0-5	Reserved, should be cleared.
6-7 ARBM	Arbitration Mode This 2-bit field controls the arbitration for PFlash controllers supporting 2 AHB ports. 00 Fixed priority arbitration with AHB p0 > p1. 01 Fixed priority arbitration with AHB p1 > p0. 1*x* Round-robin arbitration.

Table 158. PFAPR field descriptions (continued)

Field	Description
8-15 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 PFCR0[B0_P*x*_DPFE, B0_P*x*_IPFE, B*x*_P*y*_BFE] bits. 0 Prefetching may be triggered by this master. 1 No prefetching may be triggered by this master.
16-31 M*x*AP	Master *x* Access Protection (*x* = 0,1,2,...,7) These fields control whether read and write accesses to the Flash 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.

User Test 0 register (UT0)

The User Test feature gives the user of the Flash module the ability to perform test features on the Flash. The User Test 0 register allows controlling the way in which the Flash content check is done.

The UT0[MRE], UT0[MRV], UT0[AIS], UT0[EIE], and DSI[7:0] bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 163. User Test 0 register (UT0)

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	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W	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	X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 159. UT0 field descriptions

Field	Description
UTE 0	User Test Enable This status bit indicates when User Test is enabled. All bits in UT0–2 and UMISR0–4 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. 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.
1:7	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
DSI7-0 8:15	Data Syndrome Input 7–0 These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. The DSI7–0 bits correspond to the 8 syndrome bits on a double 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.
16:24	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
25	*Reserved* (Read/Write) This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.
MRE 26	Margin Read Enable MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads. Margin reads are only active during Array Integrity Checks; Normal user reads are not affected by MRE. 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 disabled. All reads are User mode reads. 1 Margin reads are enabled.
MRV 27	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 28	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. 0 ECC Logic Check is disabled. 1 ECC Logic Check is enabled.

Table 159. UT0 field descriptions (continued)

Field	Description
AIS 29	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks or Margin Mode. 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. In Margin Mode only the linear sequence (AIS = 1) is allowed, while the proprietary sequence (AIS = 0) is forbidden. 0 Array Integrity sequence is a proprietary sequence. 1 Array Integrity or Margin Mode sequence is sequential.
AIE 31	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 (UMISR0–4) 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 disabled. 1 Array Integrity Checks are enabled.
AID 31	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 (UMISR0–4) can be checked. 0 Array Integrity Check is on-going. 1 Array Integrity Check is done.

User Test 1 register (UT1)

The User Test 1 register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 164. User Test 1 register (UT1)

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	DAI 31	DAI 30	DAI 29	DAI 28	DAI 27	DAI 26	DAI 25	DAI 24	DAI 23	DAI 22	DAI 21	DAI 20	DAI 19	DAI 18	DAI 17	DAI 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	DAI 15	DAI 14	DAI 13	DAI 12	DAI 11	DAI 10	DAI 9	DAI 8	DAI 7	DAI 6	DAI 5	DAI 4	DAI 3	DAI 2	DAI 1	DAI 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 160. UT1 field descriptions

Field	Description
DAI[31:0] 0:31	Data Array Input 31–0 These bits represent the input of the even word of ECC logic used in the ECC Logic Check. The DAI[31:0] bits correspond to the 32 array bits representing Word 0 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

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.

The User Test 2 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 165. User Test 2 register (UT2)

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	DAI 63	DAI 62	DAI 61	DAI 60	DAI 59	DAI 58	DAI 57	DAI 56	DAI 55	DAI 54	DAI 53	DAI 52	DAI 51	DAI 50	DAI 49	DAI 48
Reset	0	0	0	0	0	0	0	0	0	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	DAI 47	DAI 46	DAI 45	DAI 44	DAI 43	DAI 42	DAI 41	DAI 40	DAI 39	DAI 38	DAI 37	DAI 36	DAI 35	DAI 34	DAI 33	DAI 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 161. UT2 field descriptions

Field	Description
DAI[63:32] 0:31	Data Array Input [63:32] These bits represent the input of the odd word of ECC logic used in the ECC Logic Check. The DAI[63:32] bits 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.

User Multiple Input Signature Register 0 (UMISR0)

The Multiple Input Signature Register 0 (UMISR0) provides a mean to evaluate the array integrity. UMISR0 represents the bits 31:0 of the whole 144-bit word (2 double words including ECC).

UMISR0 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Figure 166. User Multiple Input Signature Register 0 (UMISR0)

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	MS 031	MS 030	MS 029	MS 028	MS 027	MS 026	MS 025	MS 024	MS 023	MS 022	MS 021	MS 020	MS 019	MS 018	MS 017	MS 016
Reset	0	0	0	0	0	0	0	0	0	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	MS 015	MS 014	MS 013	MS 012	MS 011	MS 010	MS 009	MS 008	MS 007	MS 006	MS 005	MS 004	MS 003	MS 002	MS 001	MS 000
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 162. UMSIR0 field descriptions

Field	Description
MS[031:000] 0:31	Multiple input Signature 031–000 These bits represent the MISR value obtained by 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.

User Multiple Input Signature Register 1 (UMISR1)

The Multiple Input Signature Register 1 (UMISR1) provides a means to evaluate the array integrity. UMISR1 represents bits 63:32 of the whole 144-bit word (2 double words including ECC).

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

Figure 167. User Multiple Input Signature Register 1 (UMISR1)

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	MS 063	MS 062	MS 061	MS 060	MS 059	MS 058	MS 057	MS 056	MS 055	MS 054	MS 053	MS 052	MS 051	MS 050	MS 049	MS 048
Reset	0	0	0	0	0	0	0	0	0	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	MS 047	MS 046	MS 045	MS 044	MS 043	MS 042	MS 041	MS 040	MS 039	MS 038	MS 037	MS 036	MS 035	MS 034	MS 033	MS 032
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 163. UMISR1 field descriptions

Field	Description
MS[063:032] 0:31	Multiple input Signature 063–032 These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR1 register.

User Multiple Input Signature Register 2 (UMISR2)

The Multiple Input Signature Register (UMISR2) provides a mean to evaluate the array integrity. UMISR2 represents the bits 95-64 of the whole 144-bit word (2 double words including ECC).

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

Note: This register is not implemented on the data Flash block.

Figure 168. User Multiple Input Signature Register 2 (UMISR2)

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	MS 095	MS 094	MS 093	MS 092	MS 091	MS 090	MS 089	MS 088	MS 087	MS 086	MS 085	MS 084	MS 083	MS 082	MS 081	MS 080
Reset	0	0	0	0	0	0	0	0	0	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	MS 079	MS 078	MS 077	MS 076	MS 075	MS 074	MS 073	MS 072	MS 071	MS 070	MS 069	MS 068	MS 067	MS 066	MS 065	MS 064
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 164. UMISR2 field descriptions

Field	Description
MS[095:064] 0:31	Multiple input Signature 095–064 These bits represent the MISR value obtained by accumulating the bits 95:64 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR2 register.

User Multiple Input Signature Register 3 (UMISR3)

The Multiple Input Signature Register 3 (UMISR3) provides a means to evaluate the array integrity. UMISR3 represents bits 127:96 of the whole 144-bit word (2 double words including ECC).

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

Note: This register is not implemented on the data Flash block.

Figure 169. User Multiple Input Signature Register 3 (UMISR3)

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	MS 127	MS 126	MS 125	MS 124	MS 123	MS 122	MS 121	MS 120	MS 119	MS 118	MS 117	MS 116	MS 115	MS 114	MS 113	MS 112
Reset	0	0	0	0	0	0	0	0	0	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	MS 111	MS 110	MS 109	MS 108	MS 107	MS 106	MS 105	MS 104	MS 103	MS 102	MS 101	MS 100	MS 099	MS 098	MS 097	MS 096
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 165. UMISR3 field descriptions

Field	Description
MS[127:096] 0:31	Multiple Input Signature 127–096 These bits represent the MISR value obtained accumulating bits 127:96 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR3 register.

User Multiple Input Signature Register 4 (UMISR4)

The Multiple Input Signature Register 4 (UMISR4) provides a means to evaluate the array integrity. The UMISR4 represents the ECC bits of the whole 144-bit word (2 double words including ECC). 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 odd and even double words, respectively.

UMISR4 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Note: This register is not implemented on the data Flash block.

Figure 170. User Multiple Input Signature Register 4 (UMISR4)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 159	MS 158	MS 157	MS 156	MS 155	MS 154	MS 153	MS 152	MS 151	MS 150	MS 149	MS 148	MS 147	MS 146	MS 145	MS 144
Reset	0	0	0	0	0	0	0	0	0	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	MS 143	MS 142	MS 141	MS 140	MS 139	MS 138	MS 137	MS 136	MS 135	MS 134	MS 133	MS 132	MS 131	MS 130	MS 129	MS 128
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 166. UMISR4 field descriptions

Field	Description
MS[159:128] 0:31	Multiple Input Signature 159:128 These bits represent the MISR value obtained accumulating: – MS[135:128]—8 ECC bits for the even double word – MS138—Single ECC error detection for even double word – MS139—Double ECC error detection for even double word – MS[151:144]—8 ECC bits for the odd double word – MS154—Single ECC error detection for odd double word – MS155—Double ECC error detection for odd double word The MS can be seeded to any value by writing the UMISR4 register.

Non-Volatile Private Censorship Password 0 register (NVPWD0)

The Non-Volatile Private Censorship Password 0 register (NVPWD0) contains the 32 LSB of the password used to validate the Censorship information contained in NVSCI0–1 registers.

Note: This register is not implemented on the data Flash block.

Figure 171. Non-Volatile private Censorship Password 0 register (NVPWD0)

Address: 0x20_3DD8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD 31	PWD 30	PWD 29	PWD 28	PWD 27	PWD 26	PWD 25	PWD 24	PWD 23	PWD 22	PWD 21	PWD 20	PWD 19	PWD 18	PWD 17	PWD 16
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	PWD 15	PWD 14	PWD 13	PWD 12	PWD 11	PWD 10	PWD 9	PWD 8	PWD 7	PWD 6	PWD 5	PWD 4	PWD 3	PWD 2	PWD 1	PWD 0
Reset	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0

Table 167. NVPWD0 field descriptions

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

Non-Volatile Private Censorship Password 1 register (NVPWD1)

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

Note: This register is not implemented on the data Flash block.

Figure 172. Non-Volatile Private Censorship Password 1 register (NVPWD1)

Address: 0x20_3DDC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
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	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD	PWD
W	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reset	1	0	1	1	1	1	1	0	1	1	1		1	1	1	1

Table 168. NVPWD1 field descriptions

Field	Description
0:31	**PWD63–32**: *PassWorD 63–32* The PWD63–32 registers represent the 32 MSB of the Private Censorship Password.

Non-Volatile System Censoring Information 0 register (NVSCI0)

The Non-Volatile System Censoring Information 0 register (NVSCI0) stores the 32 LSB of the Censorship Control Word of the device.

NVSCI0 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Figure 173. Non-Volatile System Censoring Information 0 register (NVSCI0)

Address: 0x20_3DE0 Access: User read/write

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

Table 169. NVSCI0 field descriptions

Field	Description
SC[15:0] 0:15	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.

Table 169. NVSCI0 field descriptions (continued)

Field	Description
CW[15:0] 16:31	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.

Non-Volatile System Censoring Information 1 register (NVSCI1)

The Non-Volatile System Censoring Information 1 register (NVSCI1) stores the 32 MSB of the Censorship Control Word of the device.

NVSCI1 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Figure 174. Non-Volatile System Censoring Information 1 register (NVSCI1)

Address: 0x20_3DE4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC 31	SC 30	SC 29	SC 28	SC 27	SC 26	SC 25	SC 24	SC 23	SC 22	SC 21	SC 20	SC 19	SC 18	SC 17	SC 16
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	CW 31	CW 30	CW 29	CW 28	CW 27	CW 26	CW 25	CW 24	CW 23	CW 22	CW 21	CW 20	CW 19	CW 18	CW 17	CW 16
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Table 170. NVSCI1 field descriptions

Field	Description
SC[32:16] 0:15	Serial Censorship control word 32–16 These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Public Access is disabled. If SC[32:16] ≠ 0x55AA or NVSCI1≠ NVSCI0, the Public Access is enabled.
CW[32:16] 16:31	Censorship control Word 32–16 These bits represent the 16 MSB of the Censorship Control Word (CCW). CW[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Censored mode is disabled. CW[32:16]≠ 0x55AA or NVSCI1≠ NVSCI0, the Censored mode is enabled.

Non-Volatile User Options register (NVUSRO)

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Note: This register is not implemented on the data Flash block.

Figure 175. Non-Volatile User Options register (NVUSRO)

Address: 0x20_3E18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WATCHDOG_EN	UO30	PAD3V5V	UO28	UO27	UO26	UO25	UO24	UO23	UO22	UO21	UO20	UO19	UO18	UO17	UO16
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UO15	UO14	UO13	UO12	UO11	UO10	UO9	UO8	UO7	UO6	UO5	UO4	UO3	UO2	UO1	UO0
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Table 171. NVUSRO field descriptions

Field	Description
UO[28:0] 3:31	User Options 28–0 The UO[28:0] bits are reset based on the information stored in NVUSRO.
PAD3V5V 2	PAD3V5V 0 High Voltage supply is 5.0 V. 1 High Voltage supply is 3.3 V. Default manufacturing value before Flash initialization is '1' (3.3 V), which should ensure correct minimum slope for boundary scan.
UO[30] 1	User Option 30 The UO[30] bit is reset based on the information stored in NVUSRO.
WATCHDOG_EN 0	Watchdog Enable 0Disable after reset. 1Enable after reset. Default manufacturing value before Flash initialization is '1'

17.3.8 Code Flash programming considerations

Modify operation

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

All the sectors of the Flash 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 modify operation, any attempt to read any Flash location will output invalid data and the MCR[RWE] bit will be automatically set. This means that the Flash block is not fetchable when a modify operation is active and these commands must be executed from another memory (internal RAM or another Flash block).

If a reset occurs during a modify operation, the operation is interrupted and the block is reset to read mode. The data integrity of the Flash section where the modify operation has been terminated is not guaranteed. The interrupted Flash modify operation must be repeated.

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 (MCR[PGM] or MCR[ERS]) or UT0 (UT0[MRE] or UT0[EIE]).
2. The second step defines 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 MCR[EHV] or UT0[AIE].

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available Flash modify operations are shown in *Table 172*.

Table 172. Flash 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 MCR[EHV] (or UT0[AIE]) is set, the operands cannot be modified until MCR[DONE] (or UT0[AID]) is high.

In general, each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR[DONE] (or UT0[AID]) to go high.
2. Check operation result: check bit MCR[PEG] (or compare UMISR0–4 with expected value).
3. Switch off FPEC by resetting MCR[EHV] (or UT0[AIE]).
4. Deselect current operation by clearing MCR[PGM] and MCR[ERS] (or UT0[MRE] and UT0[EIE]).

If a modify operation is on-going in one of the Flash blocks, it is forbidden to start any other modify operation on the other Flash block.

In the following sections, all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

Double word program

A Flash program sequence operates on any double word within the Flash core.

One or both words within a double word may be altered in a single program operation.

Whenever the Flash is programmed, ECC bits are also 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 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the double word.

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/disabled blocks cannot be programmed.

The user may program the values in any or all of 2 words of a double word with a single program sequence.

Double word-bound words have addresses that differ only in address bit 2.

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 module latches address bits (22:3) at this time.
   c) The Flash module latches data written as well.
   d) This write is referred to as a program data interlock write. An interlock write may be as large as 64 bits, and as small as 32 bits (depending on the CPU bus).
3. If more than 1 word is to be programmed, write the additional address in the double word with data to be programmed. This is referred to as a program data write.
   The Flash module ignores address bits (22:3) for program data writes.
   The eventual unwritten data word defaults to 0xFFFF_FFFF.
4. Set MCR[EHV] to 1 to start the internal program sequence, or skip to step 9 to terminate.
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 addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

A program operation 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 whether the shadow, 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 at the word location determined by address bit 2. Unwritten locations default to a data value of 0xFFFF_FFFF. 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 MCR[EHV], resulting in a program terminate. A program termination forces the module to step 8 of the program sequence.

A terminated program results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The data space being operated on before the termination will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

Example 1 Double word program of data 0x55AA_55AA at address 0x00_AAA8 and data 0xAA55_AA55 at address 0x00_AAAC

```
MCR        = 0x00000010;        /* Set PGM in MCR: Select Operation */
(0x00AAA8) = 0x55AA55AA;        /* Latch Address and 32 LSB data */
(0x00AAAC) = 0xAA55AA55;        /* Latch 32 MSB 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 */
```

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) in the low or mid address space, or the shadow block (if available). The test block cannot be erased.

The erase sequence is fully automated within the Flash. 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 1s to the LMS register. If the shadow block is to be erased, this step may be skipped, and LMS is ignored.

Note: Lock and Select are independent. If a block is selected and locked, no erase will occur.

3. Write to any address in Flash. This is referred to as an erase interlock write.
4. Set MCR[EHV] to start the internal erase sequence, or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Clear 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 MCR[ERS] before setting MCR[EHV].

An erase operation may be terminated by clearing MCR[EHV], assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low.

An erase termination forces the Module to step 8 of the erase sequence.

A terminated erase results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The block(s) being operated on before the termination contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not terminate an erase sequence while in erase suspend.

Example 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 */
```

The erase sequence may be suspended to allow read access to the Flash 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 core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 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 4 Sector Erase Resume

```
MCR              = 0x00000005;                    /* Reset ESUS in MCR:
Erase Resume */
```

User Test mode

User Test mode is a customer-accessible mode that can be used to perform specific tests to check the integrity of the Flash 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 generate a Read-While-Write Error (the MCR[RWE] bit is set).

User Test operations are not allowed on the Test and Shadow blocks.

Array integrity self check

Array integrity is checked using a predefined 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 MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128-bit data, the 16-bit ECC data, and the single and double ECC errors of the two double words are therefore captured by the MISR through five different read accesses at the same location.

The whole check is done through five complete scans of the memory address space:

1. The first pass scans only bits 31:0 of each page.
2. The second pass scans only bits 63:32 of each page.
3. The third pass scans only bits 95:64 of each page.
4. The fourth pass scans only bits 127:96 of each page.
5. The fifth pass scans only the ECC bits (8 + 8) and the single and double ECC errors (2 + 2) of both double words of each page.

The 128-bit data and the 16-bit ECC data are sampled before the eventual ECC correction, while the single and double 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 UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit is set.
6. Compare the contents of the UMISR0–4 registers with the expected results.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

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.

Note: During the execution of the Array Integrity Check operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

While UT0[AID] is low and UT0[AIE] is high, the user may clear UT0[AIE], resulting in an Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 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*/
data2      = UMISR2;       /* Read UMISR2 content*/
data3      = UMISR3;       /* Read UMISR3 content*/
data4      = UMISR4;       /* Read UMISR4 content*/
UT0              = 0x00000000;              /* Reset UTE and AIE in UT0:
Operation End */
```

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 respect 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 by comparing the checksum value in UMISR[0:4].

Since Margin reads are done at voltages that differ than the normal read voltage, the lifetime expectancy of the Flash macrocell is impacted by the execution of Margin reads.

Repeated Margin reads will result in degradation of the Flash array, and will shorten the expected lifetime experienced at normal read levels.

For these reasons, Margin reads are allowed only at the factory. Margin reads are forbidden for use by user applications.

In any case the charge losses detected through the Margin mode cannot be considered failures of the device and no failure analysis will be opened on them.

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 LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set 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 0s margin, UT0[MRV] = 1 for 1s margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR[0:4] content with the expected result.
9. Write a logic 0 to the UT0[AIE], UT0[MRE] and UT0[MRV] bits.
10. If more blocks are to be checked, return to step 2.

It is mandatory to leave UT0[AIS] at 1 and use the linear address sequence (that also takes less time).

During the execution of the Margin Mode operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of wait states to guarantee the correctness of the result.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in a Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 6 Margin Read Check versus 1s

```
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000004; /* Set AIS in UT0: Select Operation */
UT0 = 0x80000024; /* Set MRE in UT0: Select Operation */
UT0 = 0x80000034; /* Set MRV in UT0: Select Margin versus 1's */
UT0 = 0x80000036; /* 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*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
```

```
data4 = UMISR4; /* Read UMISR4 content*/
UT0 = 0x80000034; /* Reset AIE in UT0: Operation End */
UT0 = 0x00000000; /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */
```

ECC logic check

ECC logic can be checked by forcing the input of ECC logic: The 64 bits of data and the 8 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the whole page (2 double words).

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 UT0[UTE] by writing the related password in UT0.
2. Write the double word input value to UT1[DAI31–0] and UT2[DAI[63–32].
3. Write the Syndrome Input value to UT0[DSI7–0].
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 the contents of the UMISR0–4 registers with the expected results.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] = 0, the UMISR0–4 and UT1–2 registers and the UT0[MRE], UT0[MRV], UT0[EIE], UT0[AIS], and UT0[DSI7–0] bits are not accessible. Reads return indeterminate data; writes have no effect.

Example 7ECC logic check

```
UT0        = 0xF9F99999;   /* Set UTE in UT0: Enable User Test */
UT1        = 0x55555555;   /* Set DAI31-0 in UT1: Even Word Input Data */
UT2        = 0xAAAAAAAA;   /* Set DAI63-32 in UT2: Odd Word Input Data */
UT0        = 0x80FF0000;   /* Set DSI7-0 in UT0: Syndrome Input Data */
UT0        = 0x80FF0008;   /* Set EIE in UT0: Select ECC Logic Check */
UT0        = 0x80FF000A;   /* 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) */
data1      = UMISR1;       /* Read UMISR1 content (expected 0xAAAAAAAA) */
data2      = UMISR2;       /* Read UMISR2 content (expected 0x55555555) */
data3      = UMISR3;       /* Read UMISR3 content (expected 0xAAAAAAAA) */
data4      = UMISR4;       /* Read UMISR4 content (expected 0x00FF00FF) */
UT0              = 0x00000000;             /* Reset UTE, AIE and EIE in
UT0: Operation End */
```

Error Correction Code (ECC)

The Flash macrocell provides a method to improve the reliability of the data stored in Flash: the usage of an Error Correction Code. The word size is fixed at 64 bits.

Each double word of 64 bits has an associated 8 ECC bits that are programmed in such a way to guarantee a Single Error Correction and a Double Error Detection (SEC-DED).

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 on product reliability.

ECC algorithms

The Flash macrocell supports the ECC algorithm “All 1s No Error”.

All 1s No Error

The All 1s No Error algorithm detects as valid any double word read on a just erased sector (all the 72 bits are 1s).

This option allows performing a Blank Check after a Sector Erase operation.

EEPROM emulation

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 EEPROM emulation.

As an example the chosen ECC algorithm allows to start from an All ‘1’s Double Word value and rewrite whichever of its four 16-bit Half-Words to an All ‘0’s content by keeping the same ECC value.

Table 173 shows a set of Double Words sharing the same ECC value.

Table 173. Bits manipulation: double words with the same ECC value

Double word	ECC value – All 1s No Error
0xFFFF_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_FFFF_0000	0xFF
0xFFFF_FFFF_0000_FFFF	0xFF
0xFFFF_0000_FFFF_FFFF	0xFF
0x0000_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_0000_0000	0xFF
0xFFFF_0000_FFFF_0000	0xFF
0x0000_FFFF_FFFF_0000	0xFF
0xFFFF_0000_0000_FFFF	0xFF
0x0000_FFFF_0000_FFFF	0xFF
0x0000_0000_FFFF_FFFF	0xFF
0xFFFF_0000_0000_0000	0xFF
0x0000_FFFF_0000_0000	0xFF
0x0000_0000_0000_0000	0xFF

When some Flash sectors are used to perform an EEPROM emulation, it is recomended for safety reasons to reserve at least three sectors for this purpose.

Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in Flash sectors, and Censored mode to avoid piracy.

Modify protection

The Flash modify protection information is stored in non-volatile Flash cells located in the TestFlash. This information is read once during the Flash 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 non-volatile modify protection registers can be programmed through a normal double word program operation at the related locations in TestFlash.

The non-volatile modify protection registers cannot be erased.

- The non-volatile modify protection registers are physically located in TestFlash. Their bits can be programmed to 0 only once, after which they cannot be restored to 1.
- The volatile modify protection registers are read/write registers containing bits that can be written at 0 or 1 by the user application.

A software mechanism is provided to independently lock/unlock each Low or Mid Address Space block against program and erase.

Software locking is done through the LML (Low/Mid Address Space Block Lock Register) 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 non-volatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash non-volatile image is at all 1s, meaning all sectors are locked.

By programming the non-volatile locations in TestFlash, the selected sectors can be unlocked.

Because the TestFlash is one-time programmable (that is, not erasable), once the sectors have been unlocked, they 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.

Censored mode

The Censored mode information is stored in non-volatile Flash cells located in the Shadow Sector. This information is read once during the Flash 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 censored mode registers is the protected state.

All the non-volatile censored mode registers can be programmed through a normal double word program operation at the related locations in the Shadow Sector.

The non-volatile censored mode registers can be erased by erasing the Shadow Sector.

- The non-volatile 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 block provides two levels of protection against piracy:

- If bits NVSCI0[CW[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Censored mode is disabled. All other possible values enable Censored mode.
- If bits NVSCI0[SC[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Public Access is disabled. All other possible values enable Public Access.

The parts are delivered to the user with Censored mode and public access disabled.

The chosen Flash ECC algorithm allows to modify the censorship status without erasing the Shadow sector, as shown in *Table 174*.

Table 174. 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

18 Enhanced Direct Memory Access (eDMA)

18.1 Introduction

This chapter describes the enhanced Direct Memory Access (eDMA) Controller, a second-generation module capable of performing complex data transfers with minimal intervention from a host processor.

18.2 Overview

The enhanced direct memory access (eDMA) controller hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with SRAM-based local memory containing the transfer control descriptors (TCD) for the channels.

Figure 176 is a block diagram of the eDMA module.

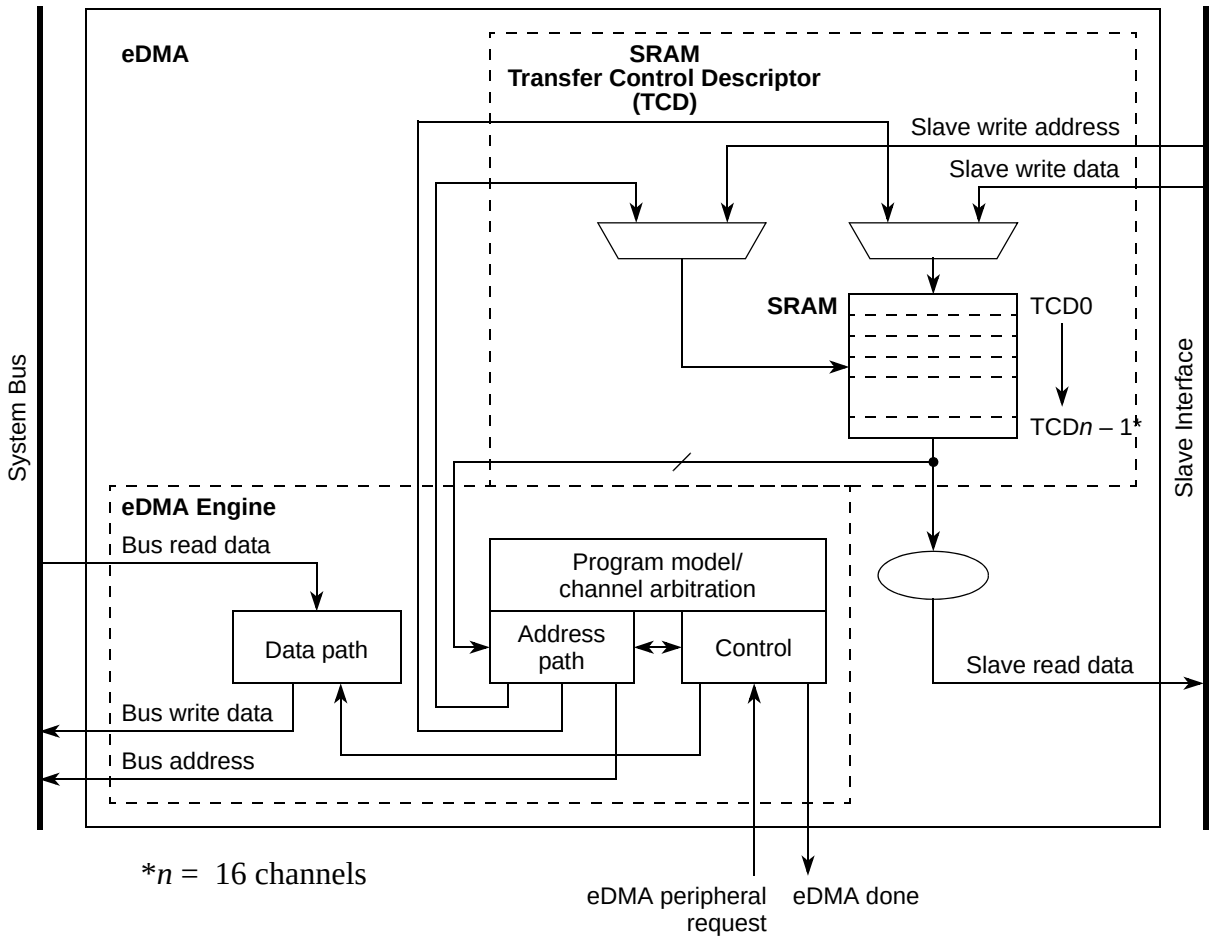


Figure 176. eDMA block diagram

18.3 Features

The eDMA is a highly programmable data transfer engine, which 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 module features:

- 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
- 16-channel implementation performs complex data transfers with minimal intervention from a host processor
  - 32 bytes of data registers, used as temporary storage to support burst transfers (refer to SSIZE bit)
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD per channel stored in local memory
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)

Note: For all three methods, one activation per execution of the minor loop is required.

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - 1 interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are enabled per channel, and logically summed together to form a single error interrupt.
- Support for scatter/gather DMA processing
- Any channel can be programmed so that it can be suspended by a higher priority channel's activation, before completion of a minor loop.

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).

18.4 Modes of operation

18.4.1 Normal mode

In normal mode, the eDMA transfers data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

18.4.2 Debug mode

If enabled by EDMA_CR[EDBG] and the CPU enters debug mode, the eDMA does not grant a service request when the debug input signal is asserted. If the signal asserts during a data block transfer as described by a minor loop in the current active channel's TCD, the eDMA continues the operation until the minor loop completes.

18.5 Memory map and register definition

18.5.1 Memory map

The eDMA programming model is partitioned into two regions:

Region 1 defines control registers; Region 2 defines the local transfer control for the descriptor memory.

Table 175 is a 32-bit view of the eDMA memory map.

Table 175. eDMA memory map

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0000	EDMA_CR—Control Register	*on page 18-386*
0x0004	EDMA_ESR—eDMA Error Status Register	*on page 18-386*
0x0008	Reserved	
0x000C	EDMA_ERQL—eDMA Enable Request Register	*on page 18-389*
0x0010	Reserved	
0x0014	EDMA_EEIRL—eDMA Enable Error Interrupt Register	*on page 18-390*
0x0018	EDMA_SERQR—eDMA Set Enable Request Register	*on page 18-391*
0x0019	EDMA_CERQR—eDMA Clear Enable Request Register	*on page 18-392*
0x001A	EDMA_SEEI—eDMA Set Enable Error Interrupt Register	*on page 18-392*
0x001B	EDMA_CEEI—eDMA Clear Enable Error Interrupt Register	*on page 18-393*
0x001C	EDMA_CIRQR—eDMA Clear Interrupt Request Register	*on page 18-393*
0x001D	EDMA_CER—eDMA Clear Error Register	*on page 18-394*
0x001E	EDMA_SSBR—eDMA Set START Bit Register	*on page 18-395*
0x001F	EDMA_CDSBR—eDMA Clear DONE Status Register	*on page 18-395*
0x0020	Reserved	
0x0024	EDMA_IRQRL—eDMA Interrupt Request Register	*on page 18-396*

Table 175. eDMA memory map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0028	Reserved	
0x002C	EDMA_ERL—eDMA Error Register	on page 18-397
0x0030	Reserved	
0x0034	EDMA_HRSL—eDMA Hardware Request Status Register	on page 18-397
0x0038–0x00FF	Reserved	
0x0100	EDMA_CPR0—eDMA Channel 0 Priority Register	on page 18-398
0x0101	EDMA_CPR1—eDMA Channel 1 Priority Register	on page 18-398
0x0102	EDMA_CPR2—eDMA Channel 2 Priority Register	on page 18-398
0x0103	EDMA_CPR3—eDMA Channel 3 Priority Register	on page 18-398
0x0104	EDMA_CPR4—eDMA Channel 4 Priority Register	on page 18-398
0x0105	EDMA_CPR5—eDMA Channel 5 Priority Register	on page 18-398
0x0106	EDMA_CPR6—eDMA Channel 6 Priority Register	on page 18-398
0x0107	EDMA_CPR7—eDMA Channel 7 Priority Register	on page 18-398
0x0108	EDMA_CPR8—eDMA Channel 8 Priority Register	on page 18-398
0x0109	EDMA_CPR9—eDMA Channel 9 Priority Register	on page 18-398
0x010A	EDMA_CPR10—eDMA Channel 10 Priority Register	on page 18-398
0x010B	EDMA_CPR11—eDMA Channel 11 Priority Register	on page 18-398
0x010C	EDMA_CPR12—eDMA Channel 12 Priority Register	on page 18-398
0x010D	EDMA_CPR13—eDMA Channel 13 Priority Register	on page 18-398
0x010E	EDMA_CPR14—eDMA Channel 14 Priority Register	on page 18-398
0x010F	EDMA_CPR15—eDMA Channel 15 Priority Register	on page 18-398
0x0110–0x0FFF	Reserved	
0x1000	TCD00—Transfer Control Descriptor 0	on page 18-399
0x1020	TCD01—Transfer Control Descriptor 1	on page 18-399
0x1040	TCD02—Transfer Control Descriptor 2	on page 18-399
0x1060	TCD03—Transfer Control Descriptor 3	on page 18-399
0x1080	TCD04—Transfer Control Descriptor 4	on page 18-399
0x10A0	TCD05—Transfer Control Descriptor 5	on page 18-399
0x10C0	TCD06—Transfer Control Descriptor 6	on page 18-399

Table 175. eDMA memory map (continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x10E0	TCD07—Transfer Control Descriptor 7	on page 18-399
0x1100	TCD08—Transfer Control Descriptor 8	on page 18-399
0x1120	TCD09—Transfer Control Descriptor 9	on page 18-399
0x1140	TCD10—Transfer Control Descriptor 10	on page 18-399
0x1160	TCD11—Transfer Control Descriptor 11	on page 18-399
0x1180	TCD12—Transfer Control Descriptor 12	on page 18-399
0x11A0	TCD13—Transfer Control Descriptor 13	on page 18-399
0x11C0	TCD14—Transfer Control Descriptor 14	on page 18-399
0x11E0	TCD15—Transfer Control Descriptor 15	on page 18-399
0x1200–0x3FFF	Reserved	

18.5.2 Register descriptions

Read operations on reserved bits in a register return undefined data. Do not write operations to reserved bits. Writing to reserved bits in a register can generate errors. The maximum register bit-width for this device is 16 bits wide.

eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in one group of 16 channels.

Arbitration can be configured to use either fixed-priority or round-robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

Refer to *Section , "eDMA Channel n Priority Registers (EDMA_CPRn).*

Figure 177. eDMA Control Register (EDMA_CR)

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	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 176. EDMA_CR field descriptions

Field	Description
0-28	Reserved.
29 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.
30 EDBG	Enable debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.
31	Reserved.

eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR 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. For either type of priority configuration error, the ERRCHN field is undefined. 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. When properly enabled, a scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion. 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 immediately stopped and the appropriate bus error flag is 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 minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write executes 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 executes before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the eDMA engine to stop the active channel, and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. 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. After the error status has been updated, the eDMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminates with the same error condition.

Figure 178. eDMA Error Status Register (EDMA_ESR)

Address: Base + 0x0004 Access: User read-only

	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	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	GPE	CPE	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

Table 177. EDMA_ESR field descriptions

Field	Description
0 VLD	Logical OR of all EDMA_ERH and EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
1–15	Reserved.
16 GPE	Group priority error. 0 No group priority error. 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
17 CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
18–23 ERRCHN[0:5]	Error channel number. Channel number of the last recorded error (excluding GPE and CPE errors). Do not rely on the number in the ERRCHN field for group and channel priority errors. Group and channel priority errors need to be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
24 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, indicating TCD.SADDR is inconsistent with TCD.SSIZE.

Table 177. EDMA_ESR field descriptions (continued)

Field	Description
25 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, indicating TCD.SOFF is inconsistent with TCD.SSIZE.
26 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, indicating TCD.DADDR is inconsistent with TCD.DSIZE.
27 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, indicating TCD.DOFF is inconsistent with TCD.DSIZE.
28 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, indicating the following conditions exist: – 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.
29 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, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.E_SG is enabled.
30 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
31 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 16 implemented channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR 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 EDMA_ERQRL.

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 eDMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request.

Figure 179. eDMA Enable Request Low Register (EDMA_ERQRL)

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	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 178. EDMA_ERQRL field descriptions

Field	Description
16–31 ERQ*n*	Enable DMA hardware service 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, there is a flag in the transfer control descriptor that can affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the DMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 15-0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR 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 EDMA_EEIRL.

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.

Figure 180. eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)

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 W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 179. EDMA_EEIRL field descriptions

Field	Description
16-31 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.

eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ*n*) provides a global set function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

Figure 181. eDMA Set Enable Request Register (EDMA_SERQR)

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		SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 180. EDMA_SERQR field descriptions

Field	Descriptions
0	Reserved.
1–7 SERQ[0:6]	Set enable request. 0–15 Set corresponding bit in EDMA_ERQRL 16–63Reserved 64–127Set all bits in EDMA_ERQRL Bit 2 (SERQ1) is not used.

eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ*n*) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes.

Figure 182. eDMA Clear Enable Request Register (EDMA_CERQR)

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		CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 181. EDMA_CERQR field descriptions

Field	Description
0	Reserved.
1–7 CERQ[0:6]	Clear enable request. 0–15 Clear corresponding bit in EDMA_ERQRL 16–63Reserved 64–127Clear all bits in EDMA_ERQRL Bit 2 (CERQ1) is not used.

eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI*n*) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

Figure 183. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

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		SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 182. EDMA_SEEIR field descriptions

Field	Description
0	Reserved.
1–7 SEEI[0:6]	Set enable error interrupt. 0–15 Set corresponding bit in EDMA_EIRRL 16–63 Reserved 64–127 Set all bits in EDMA_EEIRL Bit 2 (SEEI1) is not used.

eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI*n*) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

Figure 184. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

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		CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 183. EDMA_CEEIR field descriptions

Field	Description
0	Reserved.
1–7 CEEI[0:6]	Clear enable error interrupt. 0–15 Clear corresponding bit in EDMA_EEIRL 16–63 Reserved 64–127 Clear all bits in EDMA_EEIRL Bit 2 (CEEI1) is not used.

eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT*n*) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Figure 185. eDMA Clear Interrupt Request (EDMA_CIRQR)

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		CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 184. EDMA_CIRQR field descriptions

Field	Description
0	Reserved.
1–7 CINT[0:6]	Clear interrupt request. 0–15 Clear corresponding bit in EDMA_IRQRL 16–63 Reserved 64–127 Clear all bits in EDMA_IRQRL Bit 2 (CINT1) is not used.

eDMA Clear Error Register (EDMA_CERR)

The EDMA_CERR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CER*n*) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Figure 186. eDMA Clear Error Register (EDMA_CERR)

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		CERR[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 185. EDMA_CERR field descriptions

Field	Description
0	Reserved.
1–7 CER[0:6]	Clear error indicator. 0–15 Clear corresponding bit in EDMA_ERL 16–63 Reserved 64–127 Clear all bits in EDMA_ERL

eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR 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. Setting bit 1 (SSB*n*) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Figure 187. eDMA Set START Bit Register (EDMA_SSBR)

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		SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 186. EDMA_SSBR field descriptions

Field	Description
0	Reserved.
1–7 SSB[0:6]	Set START bit (channel service request). 0–15 Set the corresponding channel's TCD START bit 16–63 Reserved 64–127 Set all TCD START bits Bit 2 (SSB1) is not used.

eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR 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. Setting bit 1 (CDSB*n*) provides a global clear function, forcing all DONE bits to be cleared.

Figure 188. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

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		CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Table 187. EDMA_CDSBR field descriptions

Field	Description
0	Reserved.
1–7 CDSB[0:6]	Clear DONE status bit. 0–15 Clear the corresponding channel's DONE bit 16–63 Reserved 64–127 Clear all TCD DONE bits Bit 2 (CDSB1) is not used.

eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 15–0.

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 interrupt controller (INTC). 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 EDMA_CIRQR 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 EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR 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 EDMA_IRQRL.

Figure 189. eDMA Interrupt Request Low Register (EDMA_IRQRL)

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	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT
W	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 188. EDMA_IRQRL field descriptions

Field	Description
16–31 INT*n*	eDMA interrupt request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 15-0.

The eDMA 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 EDMA_EEIR, then logically summed across groups of 16 and 32 channels to form several group error interrupt requests that are then routed to the 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 EDMA_CERR 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 EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CERR. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CERR is provided so the error indicator for a *single* channel can easily be cleared.

Figure 190. eDMA Error Low Register (EDMA_ERL)

Address: Base + 0x002C 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	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 189. EDMA_ERL field descriptions

Field	Description
16–31 ERR*n*	eDMA Error *n.* 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

DMA Hardware Request Status (DMAHRSL)

The DMAHRSL registers provide a bit map for the implemented channels 16 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 eDMA's arbitration logic. This view into the hardware request signals may be used for debug purposes.

Figure 191. EDMA Hardware Request Status Register Low (EDMA_HRSL)

Address: Base + 0x0034 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	HRS 15	HRS 14	HRS 13	HRS 12	HRS 11	HRS 10	HRS 09	HRS 08	HRS 07	HRS 06	HRS 05	HRS 04	HRS 03	HRS 02	HRS 01	HRS 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 190. EDMA_HRSL field descriptions

Field	Description
16–31 HRSn	DMA Hardware Request Status 0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present. The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

eDMA Channel n Priority Registers (EDMA_CPRn)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[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; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software chooses to modify channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_CPR*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 EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR.

Refer to *Figure 177* and *Table 176* for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*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.

Figure 192. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

Address: Base + 0x100 + *n* Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	0	0	0	CHPRI			
W								
Reset	0	0	0	0	— [1]			

1. The reset value for the channel priority fields, GRPPRI[0–1] and CHPRI[0–3] is the channel number for the priority register; EDMA_CPR15[CHPRI] = 0b1111.

The following table describes the fields in the eDMA channel *n* priority register:

Table 191. EDMA_CPR*n* field descriptions

Field	Description
0 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.
1-3	Reserved.
4–7 CHPRI[0:3]	Channel *n* arbitration priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

Transfer Control Descriptor (TCD)

Each channel requires a 256-bit transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as 23 variable-length fields.

Table 192 defines the fields of the basic TCD structure.

Table 192. TCD*n* 32-bit memory structure

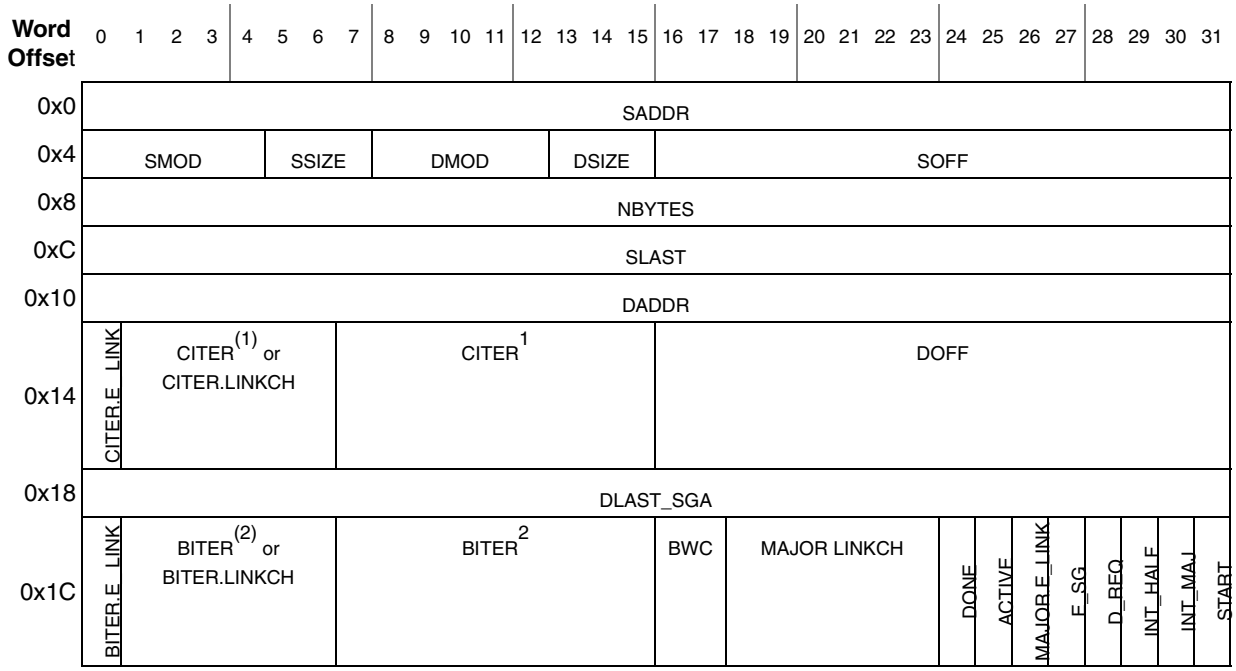
eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 0	32	Source address	SADDR	Word 0
0x1000 + (32 × *n*) + 32	5	Source address modulo	SMOD	Word 1
0x1000 + (32 × *n*) + 37	3	Source data transfer size	SSIZE	
0x1000 + (32 × *n*) + 40	5	Destination address modulo	DMOD	
0x1000 + (32 × *n*) + 45	3	Destination data transfer size	DSIZE	
0x1000 + (32 × *n*) + 48	16	Signed Source Address Offset	SOFF	
0x1000 + (32 × *n*) + 64	32	Inner minor byte count	NBYTES	Word 2
0x1000 + (32 × *n*) + 96	32	Last Source Address Adjustment	SLAST	Word 3
0x1000 + (32 × *n*) + 128	32	Destination Address	DADDR	Word 4

Table 192. TCD*n* 32-bit memory structure (continued)

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 160	1	Channel-to-channel Linking on Minor Loop Complete	CITER.E_LINK	Word 5
0x1000 + (32 × *n*) + 161	6	Current Major Iteration Count or Link Channel Number	CITER or CITER.LINKCH	
0x1000 + (32 × *n*) + 167	9	Current Major Iteration Count	CITER	
0x1000 + (32 × *n*) + 176	16	Destination Address Offset (Signed)	DOFF	
0x1000 + (32 × *n*) + 192	32	Last Destination Address Adjustment / Scatter Gather Address	DLAST_SGA	Word 6
0x1000 + (32 × *n*) + 224	1	Channel-to-channel Linking on Minor Loop Complete	BITER.E_LINK	Word 7
0x1000 + (32 × *n*) + 225	6	Starting Major Iteration Count or Link Channel Number	BITER or BITER.LINKCH	
0x1000 + (32 × *n*) + 231	9	Starting Major Iteration Count	BITER	
0x1000 + (32 × *n*) +240	2	Bandwidth Control	BWC	
0x1000 + (32 × *n*) + 242	6	Link Channel Number	MAJOR.LINKCH	
0x1000 + (32 × *n*) + 248	1	Channel Done	DONE	
0x1000 + (32 × *n*) + 249	1	Channel Active	ACTIVE	
0x1000 + (32 × *n*) + 250	1	Channel-to-channel Linking on Major Loop Complete	MAJOR.E_LINK	
0x1000 + (32 × *n*) + 251	1	Enable Scatter/Gather Processing	E_SG	
0x1000 + (32 × *n*) + 252	1	Disable Request	D_REQ	
0x1000 + (32 × *n*) + 253	1	Channel Interrupt Enable When Current Major Iteration Count is Half Complete	INT_HALF	
0x1000 + (32 × *n*) + 254	1	Channel Interrupt Enable When Current Major Iteration Count Complete	INT_MAJ	
0x1000 + (32 × *n*) + 255	1	Channel Start	START	

Figure 193 defines the fields of the TCD*n* structure.

Figure 193. TCD structure



1. If channel linking on minor link completion is disabled, TCD bits [161:175] form a 15-bit CITER field; if channel-to-channel linking is enabled, CITER becomes a 9-bit field.
2. If channel linking on minor link completion is disabled, TCD bits [225:239] form a 15-bit BITER field; if channel-to-channel linking is enabled, BITER becomes a 9-bit field.

Note: The TCD structures for the eDMA channels shown in Figure 193 are implemented in internal SRAM. These structures are not initialized at reset. Therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 193 gives a detailed description of the TCN*n* fields.

Table 193. TCD*n* field descriptions

Bits Word Offset [n:n]	Field Name	Description
0–31 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. not 0 This value defines a specific address range that is specified to be either the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, start the queue at a 0-modulo-size address and set the SMOD field to the value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue 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.

Table 193. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
37–39 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 32-bit 101 32-byte burst (64-bit x 4) 110 Reserved 111 Reserved The attempted specification of a ‘reserved’ encoding causes a configuration error.
40–44 0x4 [8:12]	DMOD [0:4]	Destination address modulo. Refer to the SMOD[0:5] definition.
45–47 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. Refer to the SSIZE[0:2] definition.
48–63 0x4 [16:31]	SOFF [0:15]	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.
64–95 0x8 [0:31]	NBYTES [0:31]	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. Once 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 of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a four GB transfer.
96–127 0xC [0:31]	SLAST [0:31]	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.
128–159 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. 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. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. This bit must be equal to the BITER.E_LINK bit otherwise a configuration error is reported.

Table 193. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
161–166 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] form a 15-bit CITER field. otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s TCD.START bit.
167–175 0x14 [7:15]	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. After 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 must be 0x0001.
176–191 0x14 [16:31]	DOFF [0:15]	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.
192–223 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If scatter/gather processing for the channel is disabled (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. Otherwise – This address points to the beginning of a 0-modulo-32 byte 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 byte, otherwise a configuration error is reported.

Table 193. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
224 0x1C [0]	BITER.E_LINK	Enables 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 BITER.LINKCH[0:5]. 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 BITER 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. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 0x1C [1:6]	BITER [0:5] or BITER.LINKCH [0:5]	Beginning or starting “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] form a 15-bit BITER field. Otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel’s TCD.START bit. When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 0x1C [7:15]	BITER [6:14]	Beginning or starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
240–241 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). To minimize start-up latency, bandwidth control stalls are suppressed for the first two system bus cycles and after the last write of each minor loop. 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for four cycles after each r/w 11 eDMA engine stalls for eight cycles after each r/w

Table 193. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
242–247 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then: – No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise – After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 0x1C [24]	DONE	Channel done. This flag indicates the eDMA 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 hardware when the channel is activated (when the channel has begun to be processed by the eDMA engine, not when the first data transfer occurs). This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 0x1C [25]	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.
250 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: 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.
251 0x1C [27]	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 that is loaded as the transfer control descriptor into the local memory. NOTE: 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.
252 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.

Table 193. TCD*n* field descriptions (continued)

Bits Word Offset [n:n]	Field Name	Description
253 0x1C [29]	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 bit in the EDMA_ERQL when the current major iteration count reaches the halfway point. The eDMA engine performs the compare (CITER == (BITER >> 1)). This halfway point interrupt request supports double-buffered (aka ping-pong) schemes, or where the processor needs an early indication of the data transfer's progress during data movement. CITER = BITER = 1 with INT_HALF enabled generates an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 0x1C [30]	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 EDMA_ERQL 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.
255 0x1C [31]	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.

18.6 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

18.6.1 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 four submodules, as shown in the following list:

- eDMA engine
  - Address 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. After 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 EDMA_CPR*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 address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCDn.{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 TCDn.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 system read data bus is the primary input, and the system write data bus is the primary output.

  The address and data path modules directly support the 2-stage pipelined system bus. The address 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).

– Program model/channel arbitration: This module implements the first section of eDMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request 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.

  A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as the following:

```
if (SSIZE < DSIZE)
transfer size = destination transfer size (# of bytes)
else
transfer size = source transfer size (# of bytes)
```

  Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

  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.

- TCD 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 slave bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the slave 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.

18.6.2 eDMA basic data flow

The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 194*, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same

basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed through the eDMA engine, first through the control module, then into the program model/channel arbitration 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 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 eDMA engine address 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 eDMA engine address path channel{x,y} registers.

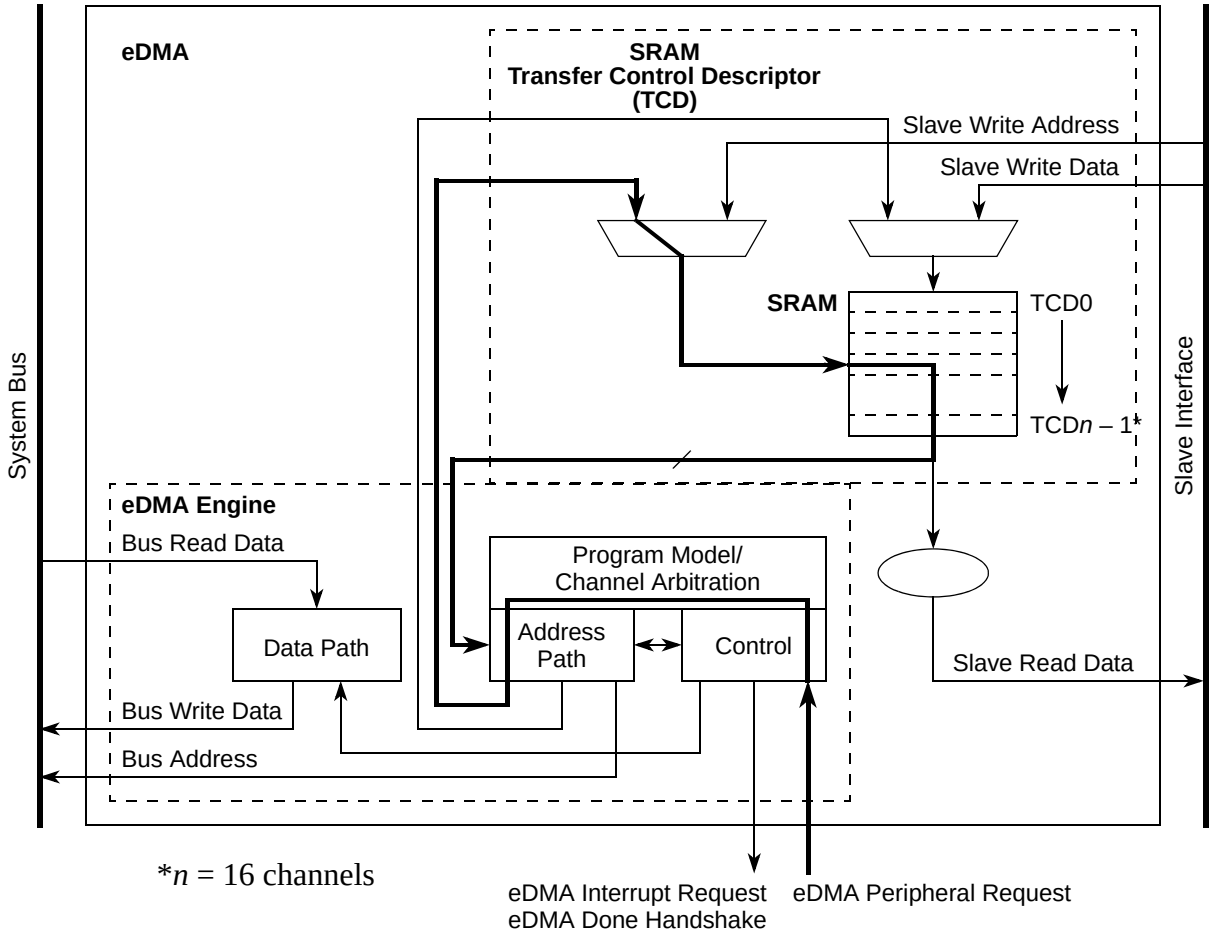


Figure 194. eDMA operation, part 1

In the second part of the basic data flow as shown in *Figure 195*, the modules associated with the data transfer (address 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 system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA Done Handshake signal is asserted at the end of the minor byte count transfer.

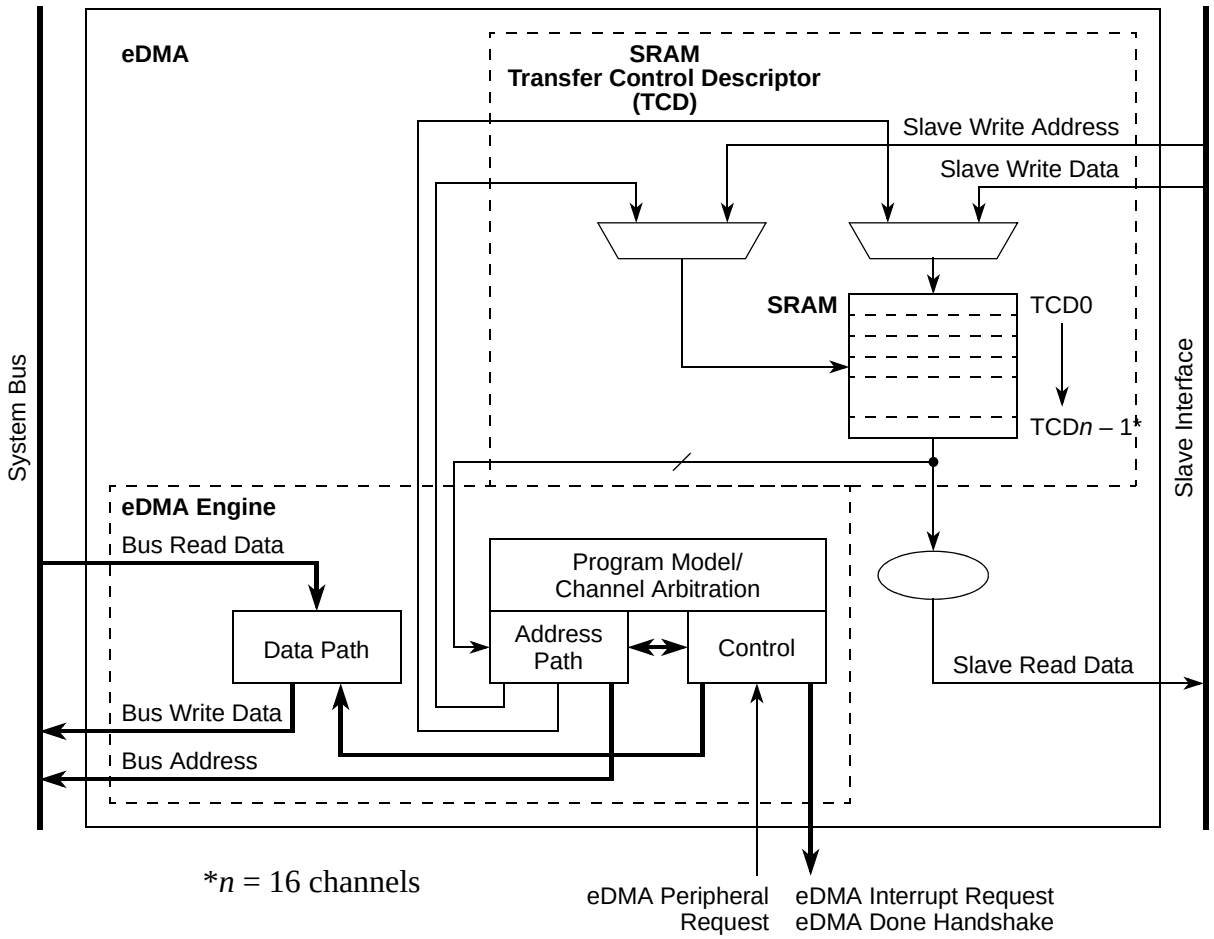


Figure 195. eDMA operation, part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD: for example., SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then 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 196*.

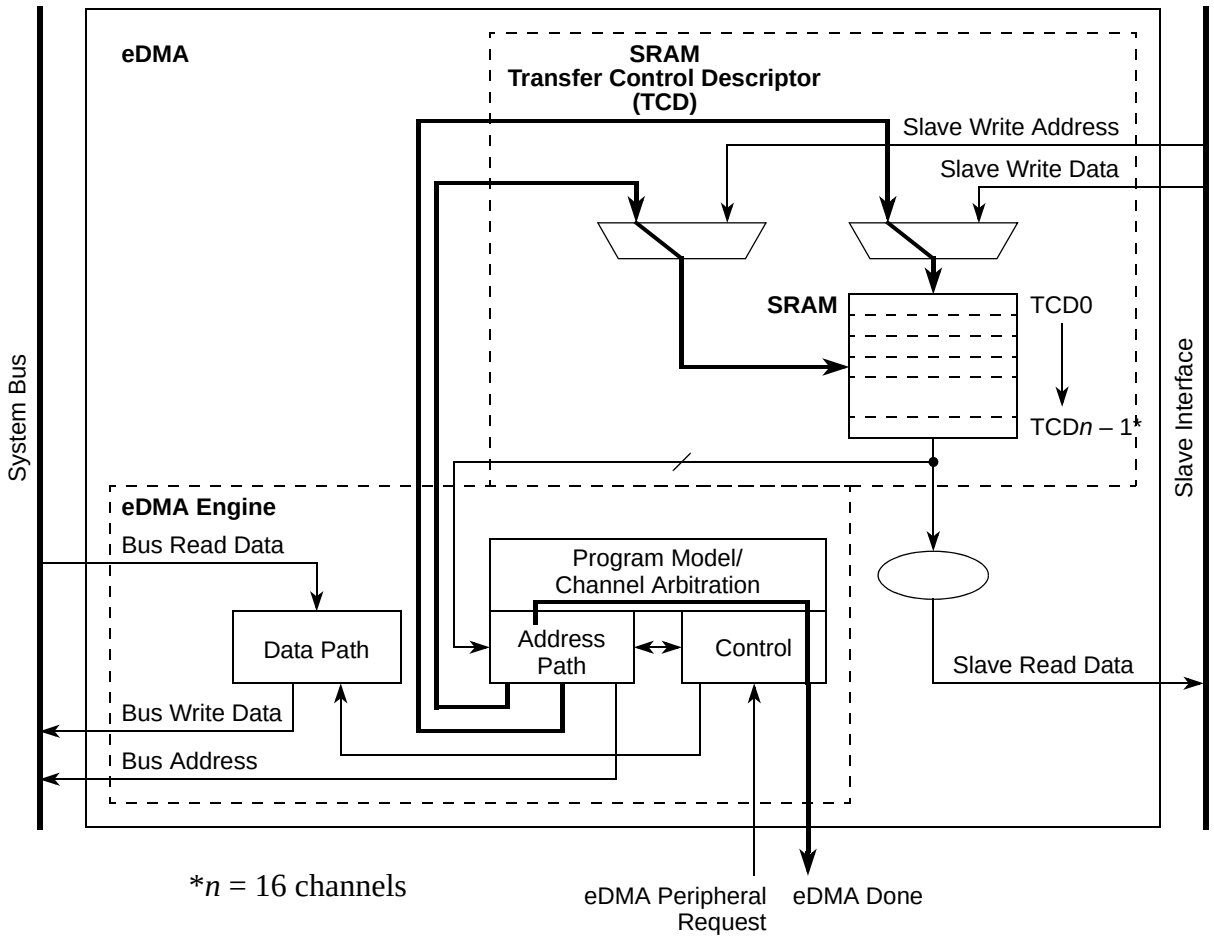


Figure 196. eDMA operation, part 3

18.6.3 eDMA 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. 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 useful metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in *Table 194*. The following assumptions apply to *Table 194* and *Table 195*:

- Internal SRAM can be accessed with zero wait-states when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- All slave accesses are 32-bits in size.

Table 194. eDMA peak transfer rates (MB/Sec)

System Speed, Transfer Size	Internal SRAM-to-Internal SRAM	32-bit Slave-to-Internal SRAM	Internal SRAM-to-32-bit Slave (buffering disabled)	Internal SRAM-to-32-bit Slave (buffering enabled)
66.7 MHz, 32-bit	66.7	66.7	53.3	88.7
66.7 MHz, 256-bit(1)	213.4	N/A(2)	N/A[2]	N/A[2]
66.7 MHz, 64-bit	133.3	66.7	53.3	88.7
83.3 MHz, 32-bit	83.3	83.3	66.7	110.8
83.3 MHz, 64-bit	166.7	83.3	66.7	110.8
83.3 MHz, 256-bit[1]	266.6	N/A[2]	N/A[2]	N/A[2]
100.0 MHz, 32-bit	100.0	100.0	80.0	133.0
100.0 MHz, 64-bit	200.0	100.0	80.0	133.0
100.0 MHz, 256-bit[1]	320.0	N/A[2]	N/A[2]	N/A[2]
132.0 MHz, 32-bit	132.0	132.0	105.6	175.6
132.0 MHz, 64-bit	264.0	132.0	105.6	175.6
132.0 MHz, 256-bit[1]	422.4	N/A[2]	N/A[2]	N/A[2]

1. A 256-bit transfer occurs as a burst of four 64-bit beats.
2. Not applicable: burst access to a slave port is not supported.

Table 194 presents a peak transfer rate comparison, measured in MBs per second where the internal-SRAM-to-internal-SRAM transfers occur at the core's datapath width; that is, either 32- or 64-bits per access. For all transfers involving the slave 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.

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 slave-mapped operand to/from internal 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: eDMA peripheral request is asserted.
- Cycle 2: The eDMA peripheral request is registered locally in the eDMA module and qualified. (TCD.START bit initiated requests start at this point with the registering of the slave write to TCD bit 255).
- 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 system bus 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 crossbar switch, arbitration at the system bus can insert an additional cycle of delay here.
- Cycle 8 – *n*: The last part of the TCD is read in. This cycle represents the 1st 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 slave read and internal SRAM write, the combined data phase time is 4 cycles. For an SRAM read and slave write, it is 5 cycles.

- Cycle *n* + 1: This cycle represents the data phase of the last destination write.
- Cycle *n* + 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. The control/status fields at word offset 0x1C in TCD*n* are read. If the major loop is complete, the MAJOR.E_LINK and E_SG bits are checked and processed if enabled.
- Cycle *n* + 3: The appropriate fields in the first part of the TCD*n* are written back into the local memory.
- Cycle *n* + 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 *n* + 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 system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with slave-to-SRAM (4 cycles) and SRAM-to-slave (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 "*n* 5." The resulting peak request rate, as a function of the system frequency, is shown in *Table 195*. This metric represents millions of requests per second.

Table 195. eDMA peak request Rate (MReq/sec)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
66.6	7.4	5.8
83.3	9.2	7.2
100.0	11.1	8.7

Table 195. eDMA peak request Rate (MReq/sec) (continued)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
133.3	14.8	11.6
150.0	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

Equation 13

PEAKreq = freq / [entry + (1 + read_ws) + (1 + write_ws) + exit]

where:

PEAKreq — peak request rate

freq — system frequency

entry — channel startup (four 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 (three cycles)

For example: consider a system with the following characteristics:

- Internal SRAM can be accessed with one wait-state when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- System operates at 150 MHz.

For an SRAM to slave transfer,

Equation 14

PEAKreq = 150 MHz / [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For an slave to SRAM transfer,

Equation 15

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 is:

Equation 16

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 (no channel is executing, eDMA is idle) are the following:

- 11 cycles for a software (TCD.START bit) request
- 12 cycles for a hardware (eDMA peripheral request signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the eDMA peripheral request 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.

18.7 Initialization / application information

18.7.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers (optional).
4. Write the 32-byte TCD for each channel that can request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by either software (setting the TCD.START bit) or by hardware (slave device asserting its eDMA peripheral request 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, including the primary transfer control parameter shown in *Table 196*, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system 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.

Table 196. TCD primary control and status fields

TCD Field Name	Description
START	Control bit to explicitly start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (Cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for “throttling” bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature

Table 196. TCD primary control and status fields (continued)

TCD Field Name	Description
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 197 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

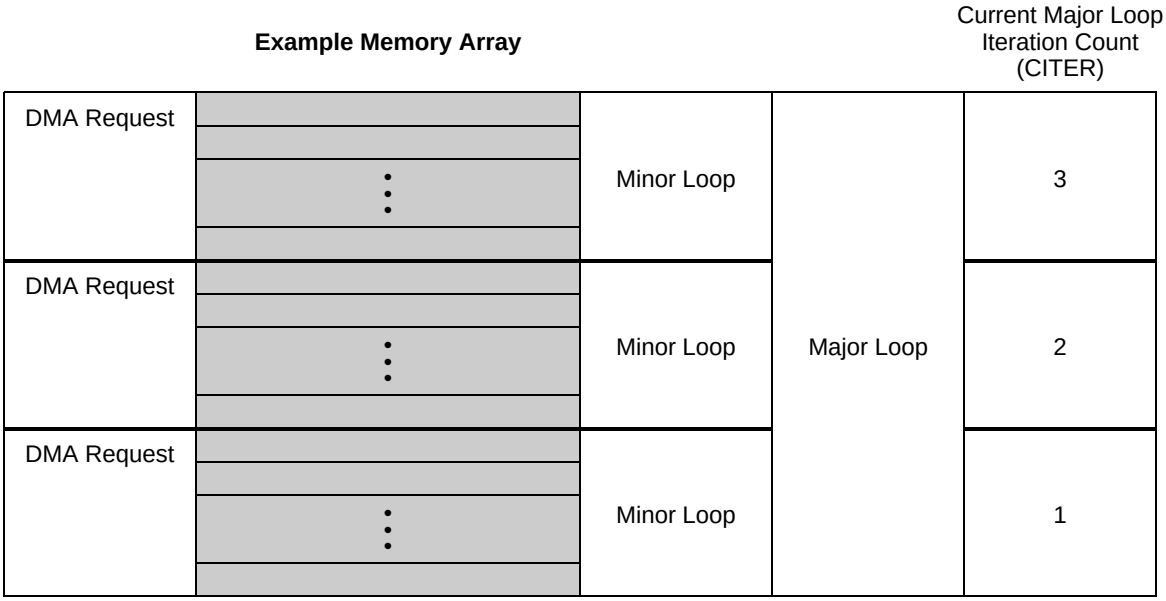


Figure 197. Example of multiple loop iterations

Figure 198 lists the memory array terms and how the TCD settings interrelate.

<table>
<tr><td>xADDR:
(Starting Address)</td><td>xSIZE:
(Size of one data transfer)
•
•
•</td><td>Minor Loop
(NBYTES in Minor Loop, often the same value as xSIZE)</td><td rowspan="3">Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)

Each DMA Source (S) and Destination (D) has its own:
• Address (xADDR)
• Size (xSIZE)
• Offset (xOFF)
• Modulo (xMOD)
• Last Address Adjustment (xLAST) where x = S or D

Peripheral queues typically have size and offset equal to NBYTES</td></tr>
<tr><td>•
•
•</td><td>•
•
•</td><td>Minor Loop</td></tr>
<tr><td>xLAST: Number of bytes added to current address after Major Loop
(Typically used to loop back)</td><td>•
•
•</td><td>Last Minor Loop</td></tr>
</table>

Figure 198. Memory array terms

18.7.2 DMA programming errors

The eDMA 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, or EDMA_ESR[GPE] and EDMA_ESR[CPE], respectively.

For all error types other than group or channel priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

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.

18.7.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in *Table 197*. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 197. DMA request summary for eDMA

DMA Request	Ch.	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source

Table 197. DMA request summary for eDMA (continued)

DMA Request	Ch.	Source	Description
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source

18.7.4 DMA arbitration mode considerations

Fixed-channel arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. 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.

Fixed-group arbitration, round-robin channel arbitration

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.

18.7.5 DMA transfer

Single request

To perform a simple transfer of '*n*' bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer completes, the TCD.DONE bit is set and an interrupt is generated if correctly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA 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.

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

TCD.START = 1 (Initialize all other fields before writing to this bit)

All other TCD fields = 0

This generates the following sequence of events:

1. Slave 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, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

Multiple requests

The next example is the same as previous 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 is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the

EDMA_ERQR, channel service requests are initiated by the slave device (set ERQR after TCD; TCD.START = 0).

TCD.CITER = TCD.BITER = 2

TCD.NBYTES = 16

TCD.SADDR = 0x1000

TCD.SOFF = 1

TCD.SSIZE = 0

TCD.SLAST = –32

TCD.DADDR = 0x2000

TCD.DOFF = 4

TCD.DSIZE = 2

TCD.DLAST_SGA = –32

TCD.INT_MAJ = 1

TCD.START = 0 (Initialize all other fields before writing this bit.)

All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) 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 execute 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 goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) 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 execute 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, EDMA_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

Modulo feature

The modulo feature of the eDMA provides the ability to easily implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for the source and destination in the TCD, and specifies which lower address bits are incremented from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. Clearing this field to 0 disables the modulo feature.

Table 198 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 198. Modulo feature example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

18.7.6 TCD status

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 can be extracted from the following sequence. 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 can be inconclusive because the active status can 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 (issued service request via software)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed 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. eDMA peripheral request asserts (issued service request via hardware)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed 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.

Active channel TCD reads

the eDMA reads 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.

Preemption status

Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes. A preempt-able situation is one in which 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 (or more exactly, 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.

18.7.7 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 determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last.

When the major loop is exhausted, only the major loop channel link fields are used to determine whether to make a channel link. For example, with the initial fields of:

TCD.CITER.E_LINK = 1

TCD.CITER.LINKCH = 0xC

TCD.CITER value = 0x4

TCD.MAJOR.E_LINK = 1

TCD.MAJOR.LINKCH = 0x7

channel linking executes 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 nine 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: After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be 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.

Table 199 summarizes how a DMA channel can “link” to another DMA channel, i.e, use another channel’s TCD, at the end of a loop.

Table 199. Channel linking parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of Minor Loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration)
	citer.linkch	Link channel number when linking at end of minor loop (current iteration)
Link at end of Major Loop	major.e_link	Enable channel-to-channel linking on major loop completion
	major.linkch	Link channel number when linking at end of major loop

18.7.8 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits 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 is set in the programmer's model, but it is unclear whether the link completed before the channel retired.

Use the following coherency model when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.MAJOR.E_LINK bit
2. Read 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.D
   b) If the bit is cleared, the channel had already retired before the dynamic link completed.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic 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 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 or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

19 DMA Channel Mux (DMA_MUX)

19.1 Introduction

19.1.1 Overview

The DMA Mux allows to route a configurable amount of DMA sources (slots) to a configurable amount of DMA channels. This is illustrated in *Figure 199*.

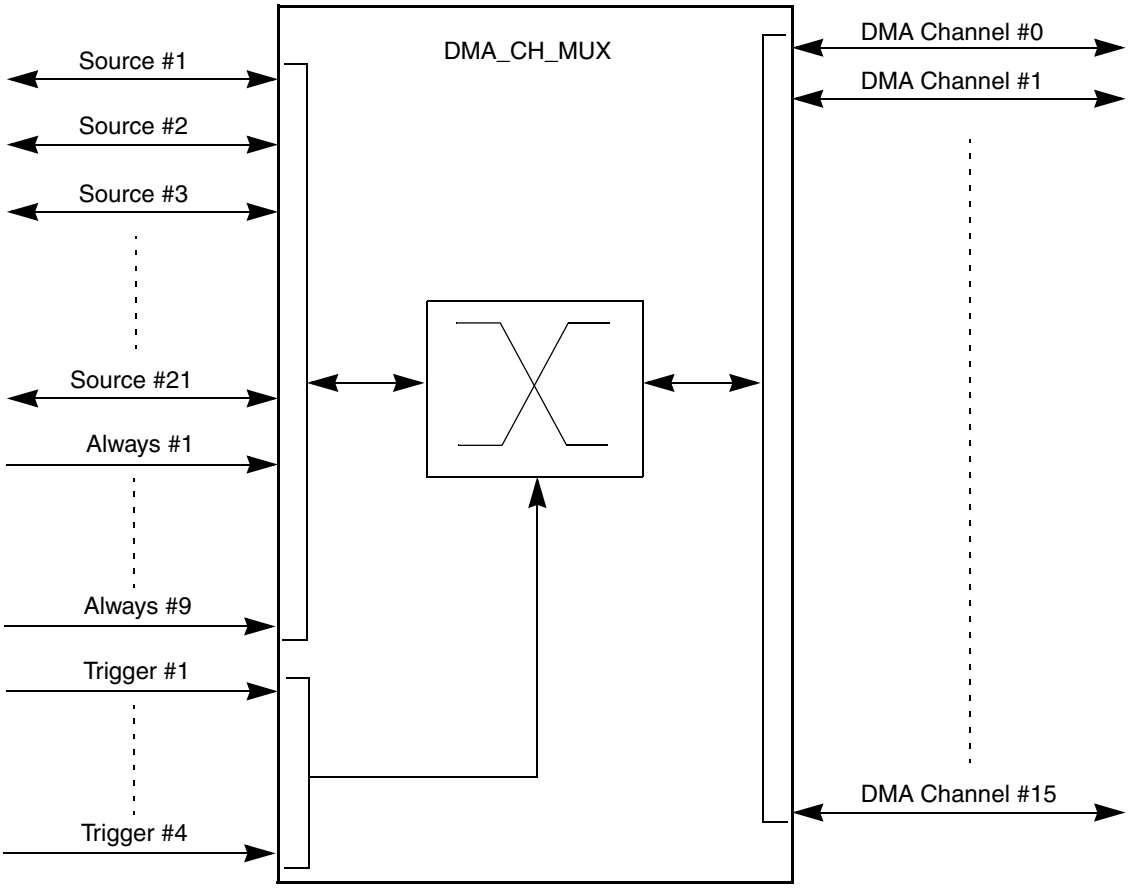


Figure 199. DMA Mux block diagram

19.1.2 Features

The DMA Mux has these major features:

- 16 independently selectable DMA channel routers
  - 4 channels with normal or periodic triggering capability
  - 12 channels with normal capability
- Each channel router can be assigned to 1 of 21 possible peripheral DMA sources

19.1.3 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 DMA Channel Mux. 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_0_TX or DSPI_0_RX example) is routed directly to the specified DMA channel. The operation of the DMA Mux 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).

DMA channels 0–3 may be used in all the modes listed above but channels 4–15 may be configured only to disabled or normal mode.

19.2 External signal description

19.2.1 Overview

The DMA Mux has no external pins.

19.3 Memory map and register definition

This section provides a detailed description of all memory-mapped registers in the DMA Mux.

19.3.1 Memory map

Table 200 shows the memory map for the DMA Mux. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA Mux.

Table 200. DMA_MUX memory map

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0000	Channel #0 Configuration (CHCONFIG0)	*on page 19-427*
0x0001	Channel #1 Configuration (CHCONFIG1)	*on page 19-427*
0x0002	Channel #2 Configuration (CHCONFIG2)	*on page 19-427*
0x0003	Channel #3 Configuration (CHCONFIG3)	*on page 19-427*
0x0004	Channel #4 Configuration (CHCONFIG4)	*on page 19-427*

Table 200. DMA_MUX memory map (continued)

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0005	Channel #5 Configuration (CHCONFIG5)	on page 19-427
0x0006	Channel #6 Configuration (CHCONFIG6)	on page 19-427
0x0007	Channel #7 Configuration (CHCONFIG7)	on page 19-427
0x0008	Channel #8 Configuration (CHCONFIG8)	on page 19-427
0x0009	Channel #9 Configuration (CHCONFIG9)	on page 19-427
0x000A	Channel #10 Configuration (CHCONFIG10)	on page 19-427
0x000B	Channel #11 Configuration (CHCONFIG11)	on page 19-427
0x000C	Channel #12 Configuration (CHCONFIG12)	on page 19-427
0x000D	Channel #13 Configuration (CHCONFIG13)	on page 19-427
0x000E	Channel #14 Configuration (CHCONFIG14)	on page 19-427
0x000F	Channel #15 Configuration (CHCONFIG15)	on page 19-427
0x0010	Channel #16 Configuration (CHCONFIG16)	on page 19-427
0x0011	Channel #17 Configuration (CHCONFIG17)	on page 19-427
0x0012	Channel #18 Configuration (CHCONFIG18)	on page 19-427
0x0013	Channel #19 Configuration (CHCONFIG19)	on page 19-427
0x0014	Channel #20 Configuration (CHCONFIG20)	on page 19-427
0x0015	Channel #21 Configuration (CHCONFIG21)	on page 19-427
0x0016	Channel #22 Configuration (CHCONFIG22)	on page 19-427
0x0017	Channel #23 Configuration (CHCONFIG23)	on page 19-427
0x0018	Channel #24 Configuration (CHCONFIG24)	on page 19-427
0x0019	Channel #25 Configuration (CHCONFIG25)	on page 19-427
0x001A	Channel #26 Configuration (CHCONFIG26)	on page 19-427
0x001B	Channel #27 Configuration (CHCONFIG27)	on page 19-427
0x001C	Channel #28 Configuration (CHCONFIG28)	on page 19-427
0x001D	Channel #29 Configuration (CHCONFIG29)	on page 19-427
0x001E	Channel #30 Configuration (CHCONFIG30)	on page 19-427
0x001F–0x3FFF	Reserved	

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 + 0x0000’, but performing a 32-bit access to address ‘Base + 0x0001’ is illegal.

19.3.2 Register descriptions

Channel Configuration Registers

Each of the DMA channels can be independently enabled/disabled and associated with one of the #SRC + #ALE total DMA sources in the system.

Figure 200. Channel Configuration Registers (CHCONFIG#*n*)

Address: Base + *#n* Access: User read/write

	7	6	5	4	3	2	1	0
R W	ENBL	TRIG	SOURCE					
Reset	0	0	0	0	0	0	0	0

Table 201. CHCONFIG#*x* field descriptions

Field	Description
7 ENBL	DMA Channel Enable ENBL enables the DMA Channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA Mux. The DMA has separate channel enables/disables that should be used to disable or reconfigure a DMA channel. 1 DMA channel is enabled.
6 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.
5–0 SOURCE	DMA Channel Source (slot) SOURCE specifies which DMA source, if any, is routed to a particular DMA channel. See *Table 203*.

Table 202. 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.

19.4 DMA request mapping

Table 203. DMA channel mapping

DMA_CH_MUX channel	Module	DMA requesting module	DMA Mux input #
1	DSPI_0	DSPI_0 TX	DMA MUX Source #1
2	DSPI_0	DSPI_0 RX	DMA MUX Source #2
3	DSPI_1	DSPI_1 TX	DMA MUX Source #3
4	DSPI_1	DSPI_1 RX	DMA MUX Source #4
5	DSPI_2	DSPI_2 TX	DMA MUX Source #5
6	DSPI_2	DSPI_2 RX	DMA MUX Source #6
7	Not connected		DMA MUX Source #7
8	Not connected		DMA MUX Source #8
9	CTU_0	CTU	DMA MUX Source #9
10	CTU_0	CTU FIFO 1	DMA MUX Source #10
11	CTU_0	CTU FIFO 2	DMA MUX Source #11
12	CTU_0	CTU FIFO 3	DMA MUX Source #12
13	CTU_0	CTU FIFO 4	DMA MUX Source #13
14	flexpwm_0	FlexPWM WR	DMA MUX Source #14
15	Not connected		DMA MUX Source #15
16	etimer_0	eTimer_0 CH0	DMA MUX Source #16
17	etimer_0	eTimer_0 CH1	DMA MUX Source #17
18	Not connected		DMA MUX Source #18
19	Not connected		DMA MUX Source #19
20	adc_0	ADC_0	DMA MUX Source #20
21	Not connected		DMA MUX Source #21
22	—	ALWAYS requestors	DMA MUX Source #22
23	—	ALWAYS requestors	DMA MUX Source #23
24	—	ALWAYS requestors	DMA MUX Source #24
25	—	ALWAYS requestors	DMA MUX Source #25
26	—	ALWAYS requestors	DMA MUX Source #26
27	—	ALWAYS requestors	DMA MUX Source #27
28	—	ALWAYS requestors	DMA MUX Source #28
29	—	ALWAYS requestors	DMA MUX Source #29
30	—	ALWAYS requestors	DMA MUX Source #30

19.5 Functional description

This section provides a complete functional description of the DMA Mux. The primary purpose of the DMA Mux is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA Mux is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in *Section 19.6.2, "Enabling and configuring sources* is followed, the configuration of the DMA Mux may be changed during the normal operation of the system.

Functionally, the DMA Mux channels may be divided into two classes: Channels that implement the normal routing functionality plus periodic triggering capability, and channels that implement only the normal routing functionality.

19.5.1 DMA channels with periodic triggering capability

Besides the normal routing functionality, the first four channels of the DMA Mux 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. Please refer to *30, "Periodic Interrupt Timer (PIT)* for more information on this topic.

Note: Because of the dynamic nature of the system (i.e., 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.

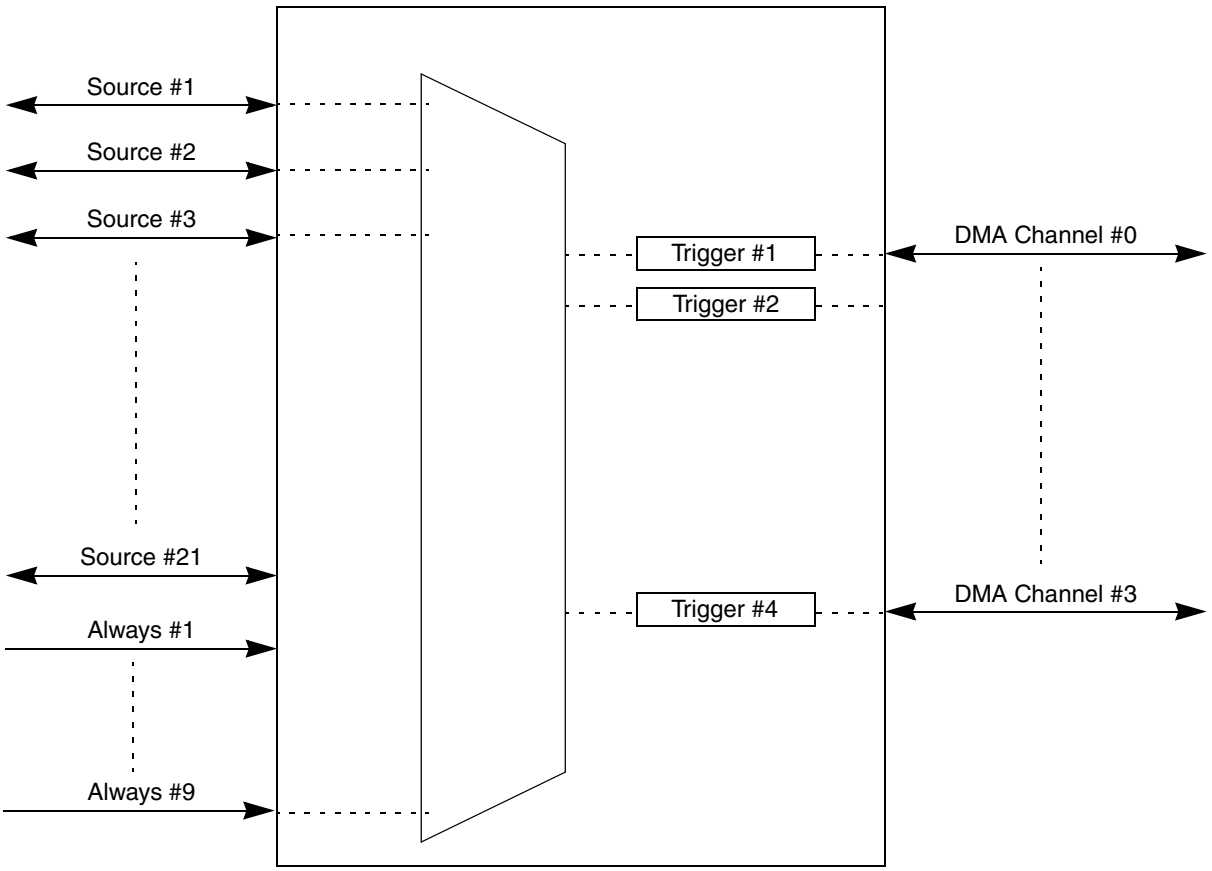


Figure 201. DMA mux triggered channels diagram

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 202*.

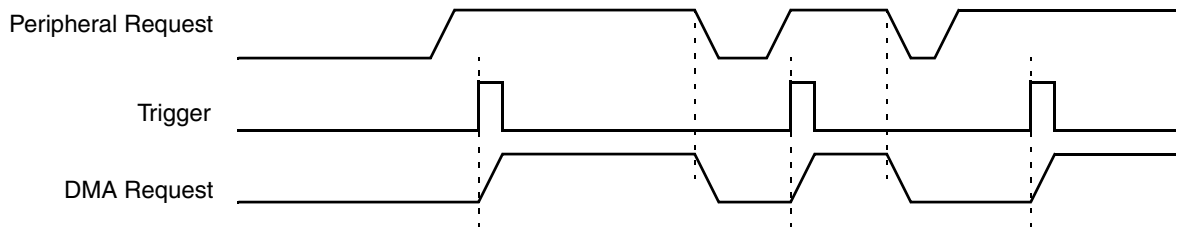


Figure 202. DMA mux channel triggering: normal operation

Once the DMA request has been serviced, the peripheral will negate 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 triggered will be ignored. This situation is illustrated in *Figure 203*.

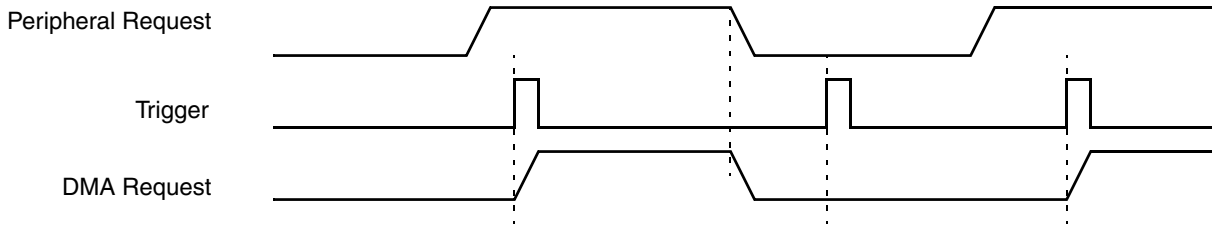


Figure 203. DMA mux channel triggering: ignored trigger

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 setup, the SPI will request 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 (for example, resolution, range of values, etc.) may be found in *30, "Periodic Interrupt Timer (PIT)*.

19.5.2 DMA channels with no triggering capability

Channels 4–15 of the DMA Mux provide the normal routing functionality as described in *Section 19.1.3, "Modes of operation.*

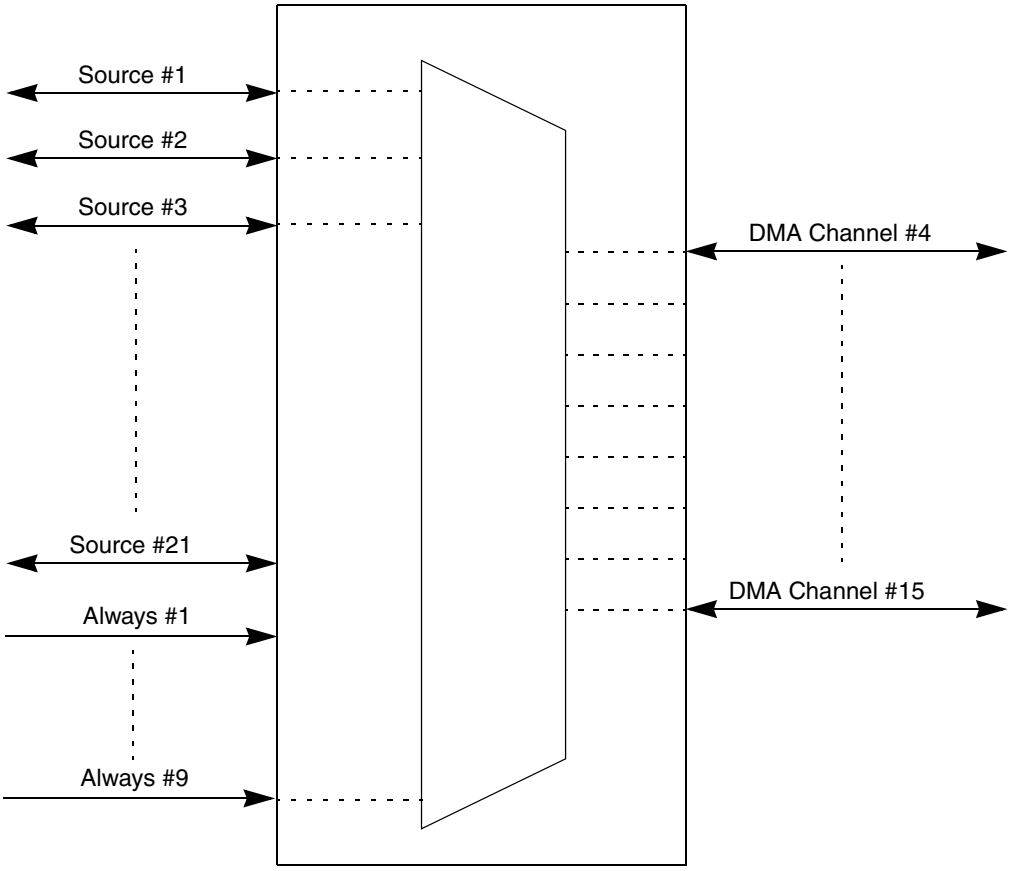


Figure 204. DMA mux channel 4–15 block diagram

19.6 Initialization/application information

19.6.1 Reset

The reset state of each individual bit is shown within the register description section (see *Section 19.3.2, "Register descriptions*). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

19.6.2 Enabling and configuring sources

Enabling a source with periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember 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. In the PIT, 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 8 Configure 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 Timer 3 in the Periodic Interrupt Timer (PIT) 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     0xFC084000/* 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);

In File main.c:
```

```
#include "registers.h"
      :
      :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Enabling a source without periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember that only DMA channels 0–7 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 9 Configure 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     0xFC084000/* 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);

In File main.c:
     #include "registers.h"
          :
          :
     *CHCONFIG2 = 0x00;
     *CHCONFIG2 = 0x85;
```

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.

Switching the source of a DMA channel:

1. Disable the DMA channel in the DMA and reconfigure 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 10 Switch DMA Channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and reconfigure it to handle the DSPI_0 transmits. This example assumes channel 0..7 have triggering capability (#TRG = 8).
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08).
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). In this case, setting the TRIG bit would have no effect, because channels 8 and above do not support the periodic triggering functionality.

The following code example illustrates steps 2 and 3 above:

```
In File registers.h:
     #define DMAMUX_BASE_ADDR     0xFC084000/* 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);

In File main.c:
     #include "registers.h"
          :
          :
     *CHCONFIG8 = 0x00;
     *CHCONFIG8 = 0x87;
```

20 Deserial Serial Peripheral Interface (DSPI)

20.1 Introduction

This chapter describes the deserial serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

The SPC560P40/34 implements the modules DSPI0, 1 and 2. The “x” appended to signal names signifies the module to which the signal applies. Thus CS0_x specifies that the CS0 signal applies to DSPI module 0, 1, etc.

20.2 Block diagram

A block diagram of the DSPI is shown in *Figure 205*.

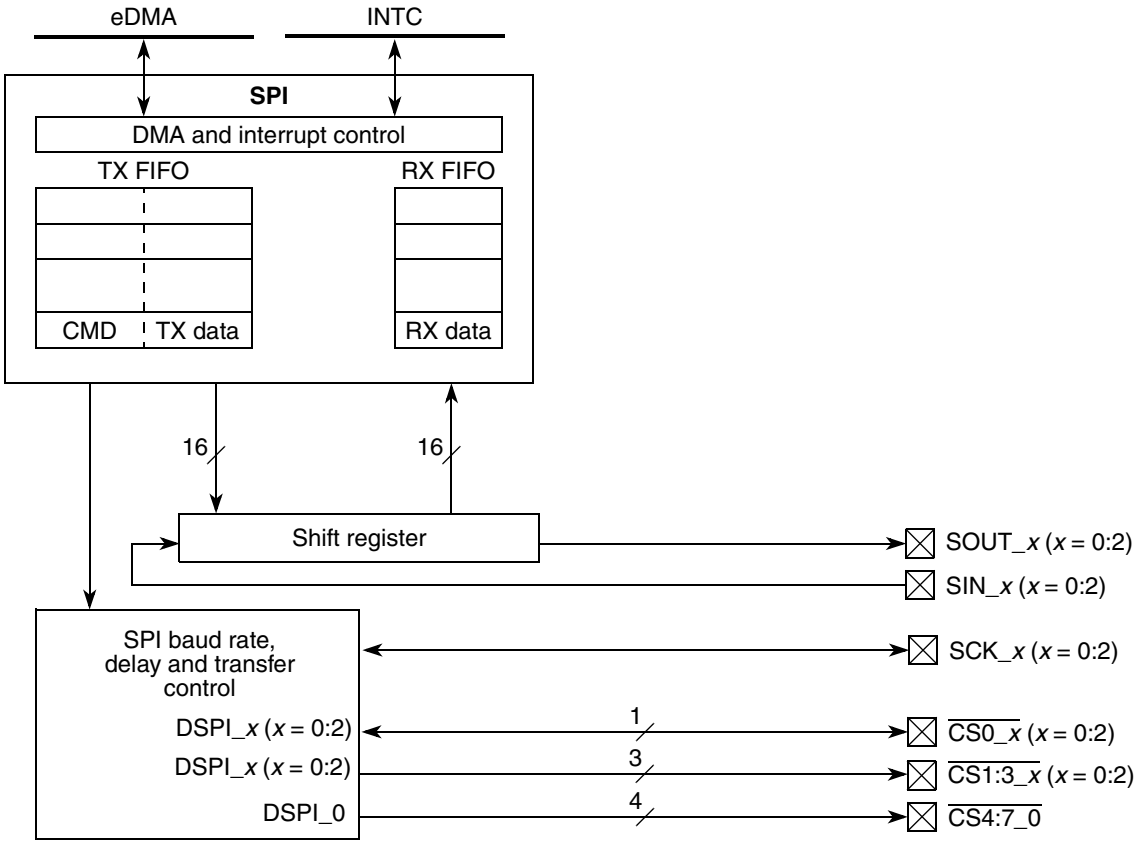


Figure 205. DSPI block diagram

20.3 Overview

The register content is transmitted using an SPI protocol. There are three DSPI modules (DSPI_0, DSPI_1, and DSPI_2) on the device. The modules are identical except that DSPI_0 has four additional chip select ($\overline{CS}$) lines.

For queued operations, the SPI queues reside in internal SRAM that is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

Figure 206 shows a DSPI with external queues in internal SRAM.

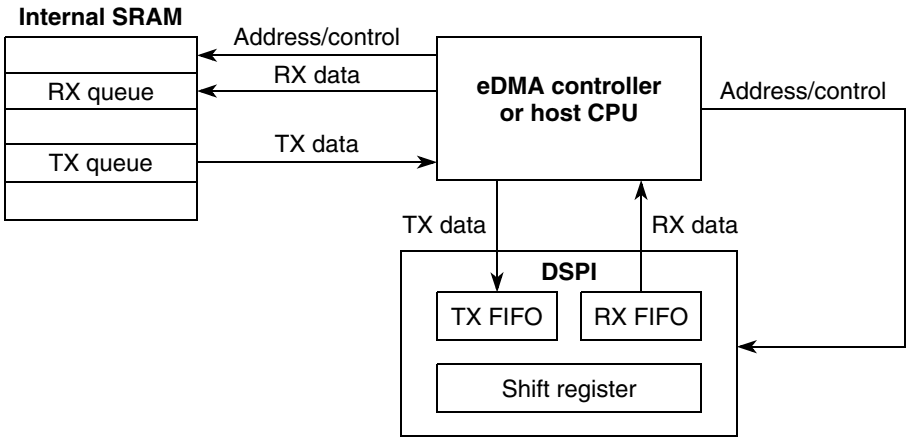


Figure 206. DSPI with queues and eDMA

20.4 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave modes
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 5 entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - 8 clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays

    CS to SCK delay

    SCK to CS delay

    Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- 8 peripheral chip selects, expandable to 64 with external demultiplexer
- Deglitching support for as many as 32 peripheral chip selects with external demultiplexer
- 2 DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)
  - FIFO under flow (slave only and SPI mode, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)
- Maximum baud rate:
  - Classic SPI Transfer Format—8 Mbit/s
  - Modified SPI Transfer Format—16 Mbit/s

20.5 Modes of operation

The DSPI has four modes of operation. These modes can be divided into two categories; module-specific modes such as master, slave, and module disable modes, and a second category that is an MCU-specific mode: debug mode. All four modes are implemented on this device.

The module-specific modes are entered by host software writing to a register. The MCU-specific mode is controlled by signals external to the DSPI. The MCU-specific mode is a mode that the entire device may enter, in parallel to the DSPI being in one of its module-specific modes.

20.5.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK, $\overline{CS}n$ and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, refer to *Section , "Master mode*.

20.5.2 Slave mode

Slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the CS0_*x* signal are configured as inputs and provided by a bus master. $\overline{CS0\_x}$ must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

For more information, refer to *Section , "Slave mode*.

20.5.3 Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

For more information, refer to *Section , "Module disable mode*.

20.5.4 Debug mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected.

For more information, refer to *Section , "Debug mode*.

20.6 External signal description

20.6.1 Signal overview

Table 204 lists off-chip DSPI signals.

Table 204. Signal properties

Name	I/O type	Function	
		Master mode	Slave mode
$\overline{CS0\_x}$	Output / input	Peripheral chip select 0	Slave select
$\overline{CS1:3\_x}$ (DSPI 0: $\overline{CS1:3\_0}$, $\overline{CS5\_0}$)	Output	Peripheral chip select 1–3	Unused(1)
$\overline{CS4\_x}$	Output	Peripheral chip select 4	Master trigger
$\overline{CS5\_x}$ (DSPI 0: $\overline{CS7\_0}$)	Output	Peripheral chip select 5 / Peripheral chip select strobe	Unused(1)
SIN_*x*	Input	Serial data in	Serial data in
SOUT_*x*	Output	Serial data out	Serial data out
SCK_*x*	Output / input	Serial clock (output)	Serial clock (input)

1. The SIUL allows you to select alternate pin functions for the device.

20.6.2 Signal names and descriptions

Peripheral Chip Select / Slave Select (CS_0)

In master mode, the CS_0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS_0 signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. CS_0 must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

Set the IBE and OBE bits in the PCR register for all CS_0 pins when the DSPI chip select or slave select primary function is selected for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit.

Refer to *Section , "Pad Configuration Registers (PCR[0:71])* for more information.

Peripheral Chip Selects 1–3 ($\overline{CS1:3}$)

$\overline{CS1:3}$ are peripheral chip select output signals in master mode. In slave mode these signals are not used. On DSPI_0, these are $\overline{CS1:3}$ and $\overline{CS5:6}$.

Peripheral Chip Select 4 ($\overline{CS4}$)

$\overline{CS4}$ is a peripheral chip select output signal in master mode.

Peripheral Chip Select 5/Peripheral Chip Select Strobe ($\overline{CS\_5}$)

$\overline{CS5}$ is a peripheral chip select output signal. When the DSPI is in master mode and PCSSE bit in the DSPI*x*_MCR is cleared, the $\overline{CS5}$ signal selects the slave device that receives the current transfer.

$\overline{CS5}$ is a strobe signal used by external logic for deglitching of the CS signals. When the DSPI is in master mode and the PCSSE bit in the DSPI*x*_MCR is set, the $\overline{CS5}$ signal indicates the timing to decode $\overline{CS0:4}$ signals, which prevents glitches from occurring.

$\overline{CS5}$ is not used in slave mode. On DSPI_0, this is $\overline{CS7}$.

Serial Input (SIN_*x*)

SIN_*x* is a serial data input signal.

Serial Output (SOUT_*x*)

SOUT_*x* is a serial data output signal.

Serial Clock (SCK_*x*)

SCK_*x* is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK_*x* is an input from an external bus master.

20.7 Memory map and registers description

20.7.1 Memory map

Table 205 shows the DSPI memory map.

Table 205. DSPI memory map

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2)	Register	Location
0x0000	DSPI_MCR—DSPI module configuration register	*on page 20-443*
0x0004	Reserved	
0x0008	DSPI_TCR—DSPI transfer count register	*on page 20-446*
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	*on page 20-447*
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	*on page 20-447*
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	*on page 20-447*
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	*on page 20-447*
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	*on page 20-447*
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	*on page 20-447*
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	*on page 20-447*
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	*on page 20-447*
0x002C	DSPI_SR—DSPI status register	*on page 20-453*

Table 205. DSPI memory map (continued)

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2)	Register	Location
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	*on page 20-455*
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	*on page 20-456*
0x0038	DSPI_POPR—DSPI pop RX FIFO register	*on page 20-458*
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	*on page 20-459*
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	*on page 20-459*
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	*on page 20-459*
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	*on page 20-459*
0x004C	DSPI_TXFR4—DSPI transmit FIFO register 4	*on page 20-459*
0x0050–0x007B	Reserved	
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	*on page 20-460*
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	*on page 20-460*
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	*on page 20-460*
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	*on page 20-460*
0x008C	DSPI_RXFR4—DSPI receive FIFO register 4	*on page 20-460*
0x0090–0x3FFF	Reserved	

20.7.2 Registers description

DSPI Module Configuration Register (DSPI*x*_MCR)

The DSPI*x*_MCR contains bits that configure attributes of the DSPI operation. The values of the HALT and MDIS bits can be changed at any time, but their effect begins on the next frame boundary. The HALT and MDIS bits in the DSPI*x*_MCR are the only bit values software can change while the DSPI is running.

Figure 207. DSPI Module Configuration Register (DSPI*x*_MCR)

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[0:1]		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	CLR_TXF	CLR_RXF	SMPL_PT[0:1]		0	0	0	0	0	0	0	HALT
W					w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 206. DSPI*x*_MCR field descriptions

Field	Description
0 MSTR	Master/slave mode select Configures the DSPI for master mode or slave mode. 0 DSPI is in slave mode. 1 DSPI is in master mode.
1 CONT_SCKE	Continuous SCK enable Enables the serial communication clock (SCK) to run continuously. Refer to *Section 20.8.6, "Continuous Serial communications clock* for details. 0 Continuous SCK disabled 1 Continuous SCK enabled If the FIFO is enabled with continuous SCK mode, before setting the CONT_SCKE bit, the TX-FIFO should be cleared and only CTAR0 register should be used for transfer attributes otherwise a change in SCK frequency occurs.
2–3 DCONF [0:1]	DSPI configuration The following table lists the DCONF values for the various configurations. **DCONF** / **Configuration** 00 / SPI 01 / Invalid value 10 / Invalid value 11 / Invalid value
4 FRZ	Freeze Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers. 1 Halt serial transfers.

Table 206. DSPI*x*_MCR field descriptions (continued)

Field	Description
5 MTFE	Modified timing format enable Enables a modified transfer format to be used. Refer to *Section , "Modified SPI transfer format (MTFE = 1, CPHA = 1)* for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
6 PCSSE	Peripheral chip select strobe enable Enables the $\overline{CS5\_x}$ to operate as a $\overline{CS}$ strobe output signal. Refer to *Section , "Peripheral Chip Select strobe enable (CS5_x)* for more information. 0 $\overline{CS5\_x}$ is used as the Peripheral chip select 5 signal. 1 $\overline{CS5\_x}$ is used as an active-low $\overline{CS}$ strobe signal.
7 ROOE	Receive FIFO overflow overwrite enable Enables an 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 that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. Refer to *Section , "Receive FIFO overflow interrupt request (RFOF)* for more information. 0 Incoming data is ignored. 1 Incoming data is put in the shift register.
8–9	Reserved, but implemented. These bits are writable, but have no effect.
10–15 PCSIS*n*	Peripheral chip select inactive state Determines the inactive state of the $\overline{CS0\_x}$ signal. $\overline{CS0\_x}$ must be configured as inactive high for slave mode operation. 0 The inactive state of $\overline{CS0\_x}$ is low. 1 The inactive state of $\overline{CS0\_x}$ is high. PCSIS7 and PSCIS6 are implemented only on DSPI_0.
16	Reserved
17 MDIS	Module disable Allows the clock to stop to the non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. Refer to *Section 20.8.8, "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.
18 DIS_TXF	Disable transmit FIFO Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section , "FIFO disable operation* for details. 0TX FIFO enabled 1TX FIFO disabled
19 DIS_RXF	Disable receive FIFO Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section , "FIFO disable operation* for details. 0 RX FIFO enabled 1 RX FIFO disabled

Table 206. DSPI*x*_MCR field descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20
CLR_TXF</td><td>Clear TX FIFO
Flushes the TX FIFO. Write a 1 to the CLR_TXF bit to clear the TX FIFO counter. The CLR_TXF bit is always read as 0.
0 Do not clear the TX FIFO counter.
1 Clear the TX FIFO counter.</td></tr>
<tr><td>21
CLR_RXF</td><td>Clear RX FIFO
Flushes the RX FIFO. Write a 1 to the CLR_RXF bit to clear the RX counter. The CLR_RXF bit is always read as 0.
0 Do not clear the RX FIFO counter.
1 Clear the RX FIFO counter.</td></tr>
<tr><td>22–23
SMPL_PT
[0:1]</td><td>Sample point
Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 222 shows where the master can sample the SIN pin. The following table lists the delayed sample points.
<table><tr><th>SMPL_PT</th><th>Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x</th></tr><tr><td>00</td><td>0</td></tr><tr><td>01</td><td>1</td></tr><tr><td>10</td><td>2</td></tr><tr><td>11</td><td>Invalid value</td></tr></table></td></tr>
<tr><td>24–30</td><td>Reserved</td></tr>
<tr><td>31
HALT</td><td>Halt
Provides a mechanism for software to start and stop DSPI transfers. Refer to Section 20.8.2, “Start and stop of DSPI transfers for details on the operation of this bit.
0 Start transfers.
1 Stop transfers.</td></tr>
</table>

DSPI Transfer Count Register (DSPI*x*_TCR)

The DSPI*x*_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPI*x*_TCR while the DSPI is running.

Figure 208. DSPI Transfer Count Register (DSPIx_TCR)

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[0:15]															
Reset	0	0	0	0	0	0	0	0	0	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

Table 207. DSPI*x*_TCR field descriptions

Field	Description
0–15 SPI_TCNT [0:15]	SPI transfer counter Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented 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,' incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved

DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTAR*n*)

The DSPI modules each contain eight clock and transfer attribute registers (DSPI*x*_CTAR*n*) that define different transfer attribute configurations. Each DSPI*x*_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPI*x*_CTARs support compatibility with the QSPI module used in certain members of the SPC56x family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI*x*_CTAR that contains the transfer's attributes. Do not write to the DSPI*x*_CTARs while the DSPI is running.

In master mode, the DSPI*x*_CTAR*n* 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 bit fields in the DSPI*x*_CTAR0 and DSPI*x*_CTAR1 registers sets the slave transfer attributes. Refer to the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI*x*_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPI*x*_CTAR0 register is used.

Figure 209. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTARn)

Address: Base + 0x000C (DSPI*x*_CTAR0) Base + 0x001C (DSPIx_CTAR4)
Base + 0x0010 (DSPI*x*_CTAR1) Base + 0x0020 (DSPI*x*_CTAR5)
Base + 0x0014 (DSPI*x*_CTAR2) Base + 0x0024 (DSPI*x*_CTAR6)
Base + 0x0018 (DSPI*x*_CTAR3) Base + 0x0028 (DSPI*x*_CTAR7)
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	LSB FE	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

Table 208. DSPI*x*_CTARn field descriptions

Field	Descriptions
0 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 209*. See the BR[0:3] field description for details on how to compute 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.
1–4 FMSZ[0:3]	Frame Size The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. *Table 210* lists the frame size encodings.
5 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.
6 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. Continuous SCK is only supported for CPHA = 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.

Table 208. DSPI*x*_CTARn field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>7
LSBFE</td><td>LSB First
The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode.
0 Data is transferred MSB first.
1 Data is transferred LSB first.</td></tr>
<tr><td>8–9
PCSSCK[0:1]</td><td>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. This field is only used in Master Mode. The table lists the prescaler values. See the CSSCK[0:3] field description for details on how to compute the PCS to SCK delay.
<table><tr><th>PCSSCK</th><th>PCS to SCK delay prescaler value</th></tr><tr><td>00</td><td>1</td></tr><tr><td>01</td><td>3</td></tr><tr><td>10</td><td>5</td></tr><tr><td>11</td><td>7</td></tr></table></td></tr>
<tr><td>10–11
PASC[0:1]</td><td>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. This field is only used in Master Mode. The table lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK delay.
<table><tr><th>PASC</th><th>After SCK delay prescaler value</th></tr><tr><td>00</td><td>1</td></tr><tr><td>01</td><td>3</td></tr><tr><td>10</td><td>5</td></tr><tr><td>11</td><td>7</td></tr></table></td></tr>
<tr><td>12–13
PDT[0:1]</td><td>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. The table lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.
<table><tr><th>PDT</th><th>Delay after transfer prescaler value</th></tr><tr><td>00</td><td>1</td></tr><tr><td>01</td><td>3</td></tr><tr><td>10</td><td>5</td></tr><tr><td>11</td><td>7</td></tr></table></td></tr>
</table>

Table 208. DSPI*x*_CTARn field descriptions (continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>14–15
PBR[0:1]</td><td>Baud Rate Prescale
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. The baud rate prescaler values are listed in the table. See the BR[0:3] field description for details on how to compute the baud rate.
<table><tr><th>PBR</th><th>Baud rate prescaler value</th></tr><tr><td>00</td><td>2</td></tr><tr><td>01</td><td>3</td></tr><tr><td>10</td><td>5</td></tr><tr><td>11</td><td>7</td></tr></table></td></tr>
<tr><td>16–19
CSSCK[0:3]</td><td>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 is the delay between the assertion of PCS and the first edge of the SCK. Table 211 lists 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:
Equation 17
$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$
See Section , “CS to SCK delay (t_{CSC}) for more details.</td></tr>
<tr><td>20–23
ASC[0:3]</td><td>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 is the delay between the last edge of SCK and the negation of PCS. Table 212 lists the scaler values. The After SCK delay is a multiple of the system clock period, and it is computed according to the following equation:
Equation 18
$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$
See Section , “After SCK delay (t_{ASC}) for more details.</td></tr>
<tr><td>24–27
DT[0:3]</td><td>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 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 213 lists the scaler values. In continuous serial communications clock operation, the DT value is fixed to one TSCK. The delay after transfer is a multiple of the system clock period and it is computed according to the following equation:
Equation 19
$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$
See Section , “Delay after transfer (tDT) for more details.</td></tr>
</table>

Table 208. DSPI*x*_CTARn field descriptions (continued)

Field	Descriptions
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. *Table 214* lists the baud rate scaler values. The baud rate is computed according to the following equation: **Equation 20** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section , "Baud rate generator* for more details.

Table 209. 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
1	1	10	60/40
1	1	11	57/43

Table 210. DSPI transfer frame size

FMSZ	Frame size	FMSZ	Frame size
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

Table 211. DSPI PCS to SCK delay scaler

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to sck delay 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 212. DSPI after SCK delay scaler

ASC	After SCK delay scaler value	ASC	After SCK delay 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 213. DSPI delay after transfer scaler

DT	Delay after transfer scaler value	DT	Delay after transfer 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 214. 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

DSPI Status Register (DSPI*x*_SR)

The DSPI*x*_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI*x*_SR by writing a 1 to clear it (w1c). Writing a 0 to a flag bit has no effect.

Figure 210. DSPI Status Register (DSPI*x*_SR)

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	0	0	RFOF	0	RFDF	0
W	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	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 215. DSPI*x*_SR field descriptions

Field	Description
0 TCF	Transfer complete flag Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section , "Classic SPI transfer format (CPHA = 0)* for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete. 1 Transfer complete.
1 TXRXS	TX and RX status Reflects the status of the DSPI. Refer to *Section 20.8.2, "Start and stop of DSPI transfers* for information on what clears and sets this bit. 0 TX and RX operations are disabled (DSPI is in STOPPED state). 1 TX and RX operations are enabled (DSPI is in RUNNING state).

Table 215. DSPI*x*_SR field descriptions (continued)

Field	Description
2	Reserved
3 EOQF	End of queue flag Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section , "Classic SPI transfer format (CPHA = 0)* for details. The EOQF bit is 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 executing command. 1 EOQ bit is set in the executing SPI command. EOQF does not function in slave mode.
4 TFUF	Transmit FIFO underflow flag Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules 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 is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred. 1 TX FIFO underflow has occurred.
5	Reserved
6 TFFF	Transmit FIFO fill flag Indicates that the TX FIFO can be filled. 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 an by acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full. 1 TX FIFO is not full.
7–11	Reserved
12 RFOF	Receive FIFO overflow flag 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 is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred. 1 RX FIFO overflow has occurred.
13	Reserved
14 RFDF	Receive FIFO drain flag Indicates that the RX FIFO can be drained. 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 eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty. 1 RX FIFO is not empty. In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.
15	Reserved

Table 215. DSPI*x*_SR field descriptions (continued)

Field	Description
16–19 TXCTR [0:3]	TX FIFO counter 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.
20–23 TXNXTPTR [0:3]	Transmit next pointer 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. Refer to *Section , "Transmit First In First Out (TX FIFO) buffering mechanism* for more details.
24–27 RXCTR [0:3]	RX FIFO counter Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section , "Classic SPI transfer format (CPHA = 0)* for details.
28–31 POPNXTPTR [0:3]	Pop next pointer Contains a pointer to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR is updated when the DSPI*x*_POPR is read. Refer to *Section , "Receive First In First Out (RX FIFO) buffering mechanism* for more details.

DSPI DMA / Interrupt Request Select and Enable Register (DSPI*x*_RSER)

The DSPI*x*_RSER serves two purposes: enables flag bits in the DSPI*x*_SR to generate DMA requests or interrupt requests, and selects the type of request to generate. Refer to the bit descriptions for the type of requests that are supported. Do not write to the DSPI*x*_RSER while the DSPI is running.

Figure 211. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)

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	0	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

Table 216. DSPI*x*_RSER field descriptions

Field	Description
0 TCF_RE	Transmission complete request enable Enables TCF flag in the DSPI*x*_SR to generate an interrupt request. 0 TCF interrupt requests are disabled. 1 TCF interrupt requests are enabled.
1–2	Reserved

Table 216. DSPI*x*_RSER field descriptions (continued)

Field	Description
3 EOQF_RE	DSPI finished request enable Enables the EOQF flag in the DSPI*x*_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled. 1 EOQF interrupt requests are enabled.
4 TFUF_RE	Transmit FIFO underflow request enable The TFUF_RE bit enables the TFUF flag in the DSPI*x*_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled. 1 TFUF interrupt requests are enabled.
5	Reserved
6 TFFF_RE	Transmit FIFO fill request enable Enables the TFFF flag in the DSPI*x*_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.
7 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI*x*_SR is set, and the TFFF_RE bit in the DSPI*x*_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
8–11	Reserved
12 RFOF_RE	Receive FIFO overflow request enable Enables the RFOF flag in the DSPI*x*_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled. 1 RFOF interrupt requests are enabled.
13	Reserved
14 RFDF_RE	Receive FIFO drain request enable Enables the RFDF flag in the DSPI*x*_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.
15 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI*x*_SR is set, and the RFDF_RE bit in the DSPI*x*_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
16–31	Reserved

DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

The DSPI*x*_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. Refer to *Section , "Transmit First In First Out (TX FIFO) buffering mechanism* for more information. Write accesses of 8 or 16 bits to the DSPI*x*_PUSHR transfer 32 bits to the TX FIFO.

Note: TXDATA is used in master and slave modes.

Figure 212. DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

Address Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
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	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 217. DSPI*x*_PUSHR field descriptions

Field	Description
0 CONT	Continuous peripheral chip select enable Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected $\overline{CS}$ signals to remain asserted between transfers. Refer to *Section , "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.
1–3 CTAS [0:2]	Clock and transfer attributes select Selects which of the DSPI*x*_CTARs sets the transfer attributes for the SPI frame. In SPI slave mode, DSPI*x*_CTAR0 is used. The following table shows how the CTAS values map to the DSPI*x*_CTARs. There are eight DSPI*x*_CTARs in the device DSPI implementation. Use in SPI master mode only.

CTAS	Use Clock and Transfer Attributes from
000	DSPI*x*_CTAR0
001	DSPI*x*_CTAR1
010	DSPI*x*_CTAR2
011	DSPI*x*_CTAR3
100	DSPI*x*_CTAR4
101	DSPI*x*_CTAR5
110	DSPI*x*_CTAR6
111	DSPI*x*_CTAR7

Table 217. DSPI*x*_PUSHR field descriptions (continued)

Field	Description
4 EOQ	End of queue 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*x*_SR is set. 0 The SPI data is not the last data to transfer. 1 The SPI data is the last data to transfer. Use in SPI master mode only.
5 CTCNT	Clear SPI_TCNT Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPI*x*_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear SPI_TCNT field in the DSPI*x*_TCR. 1 Clear SPI_TCNT field in the DSPI*x*_TCR. Use in SPI master mode only.
6–7	Reserved
10–15 PCS*x*	Peripheral chip select *x* Selects which $\overline{\text{CS}x}$ signals are asserted for the transfer. 0 Negate the $\overline{\text{CS}x}$ signal. 1 Assert the $\overline{\text{CS}x}$ signal. Use in SPI master mode only.
16–31 TXDATA [0:15]	Transmit data Holds SPI data for transfer according to the associated SPI command. Use TXDATA in master and slave modes.

DSPI POP RX FIFO Register (DSPI*x*_POPR)

The DSPI*x*_POPR allows you to read the RX FIFO. Refer to *Section , "Receive First In First Out (RX FIFO) buffering mechanism* for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI*x*_POPR fetch the RX FIFO data, and update the counter and pointer.

Note: Reading the RX FIFO field fetches data from the RX FIFO. Once the RX FIFO is read, the read data pointer is moved to the next entry in the RX FIFO. Therefore, read DSPIx_POPR only when you need the data. For compatibility, configure the TLB (MMU table) entry for DSPIx_POPR as guarded.

Figure 213. DSPI POP RX FIFO Register (DSPI*x*_POPR)

Address: Base + 0x0038 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	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 218. DSPI*x*_POPR field descriptions

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [0:15]	Received data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNXTPTR).

DSPI Transmit FIFO Registers 0–4 (DSPI*x*_TXFR*n*)

The DSPI*x*_TXFR*n* registers 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*x*_TXFR*n* registers does not alter the state of the TX FIFO. The MCU uses five registers to implement the TX FIFO, that is DSPI*x*_TXFR0–DSPI*x*_TXFR4 are used.

Figure 214. DSPI Transmit FIFO Register 0–4 (DSPI*x*_TXFR*n*)

Address: Base + 0x003C (DSPI*x*_TXFR0)
Base + 0x0040 (DSPI*x*_TXFR1)
Base + 0x0044 (DSPI*x*_TXFR2)
Base + 0x0048 (DSPI*x*_TXFR3)
Base + 0x004C (DSPI*x*_TXFR4)
Access: User read-only

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

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

Table 219. DSPI*x*_TXFR*n* field descriptions

Field	Description
0–15 TXCMD [0:15]	Transmit command Contains the command that sets the transfer attributes for the SPI data. Refer to *Section , "DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the command field.
16–31 TXDATA [0:15]	Transmit data Contains the SPI data to be shifted out.

DSPI Receive FIFO Registers 0–4 (DSPI*x*_RXFR*n*)

The DSPI*x*_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI*x*_RXFR registers are read-only. Reading the DSPI*x*_RXFR*n* registers does not alter the state of the RX FIFO. The device uses five registers to implement the RX FIFO, that is DSPI*x*_RXFR0–DSPI*x*_RXFR4 are used.

Figure 215. DSPI Receive FIFO Registers 0–4 (DSPIx_RXFRn)

Address: Base + 0x007C (DSPI*x*_RXFR0), Base + 0x0080 (DSPI*x*_RXFR1), Base + 0x0084 (DSPI*x*_RXFR2), Base + 0x0088 (DSPI*x*_RXFR3), Base + 0x008C (DSPI*x*_RXFR4) — 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	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 220. DSPI*x*_RXFR*n* field description

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [15:0]	Receive data Contains the received SPI data.

20.8 Functional description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. All communications are through an SPI-like protocol.

The DSPI supports only the serial peripheral interface (SPI) configuration in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPI*x*_MCR register determines the DSPI configuration. Refer to *Table 206* for the DSPI configuration values.

The DSPI*x*_CTAR0–DSPI*x*_CTAR7 registers hold clock and transfer attributes.The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPI*x*_PUSHR.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT_*x* and SIN_*x* signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master's 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*x*_SR is set to indicate a completed transfer. *Figure 216* illustrates how master and slave data is exchanged.

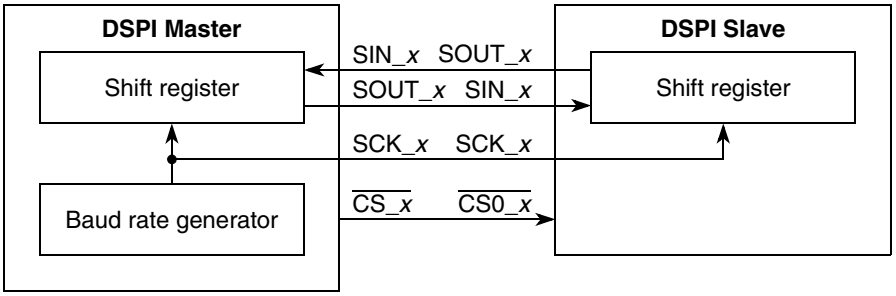


Figure 216. SPI serial protocol overview

Each DSPI has four peripheral chip select ($\overline{CS}x$) signals that select the slaves with which to communicate (DSPI_0 has eight $\overline{CS}x$ signals.)

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in *Section 20.8.5, "Transfer formats.* The transfer rate and delay settings are described in *Section 20.8.4, "DSPI baud rate and clock delay generation.*

Refer to *Section 20.8.8, "Power saving features* for information on the power-saving features of the DSPI.

20.8.1 Modes of operation

The DSPI modules have four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode. All four modes are implemented on this device.

The module-specific modes are determined by bits in the DSPI*x*_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

Master mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI*x*_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI*x*_CTARs set the transfer attributes. Transfer attribute control is on a frame by frame basis.

Refer to *Section 20.8.3, "Serial Peripheral Interface (SPI) configuration* for more details.

Slave mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI*x*_MCR is negated. The DSPI slave is selected by a bus master by having the slave's $\overline{CS0\_x}$ asserted. In slave mode, the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase, and the number of bits to transfer. These must be configured in the DSPI slave for correct communications.

Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

Refer to *Section 20.8.8, "Power saving features* for more details on the module disable mode.

Debug mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

Refer to *Figure 217* for a state diagram.

20.8.2 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. The TXRXS bit in the DSPI*x*_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPI*x*_SR is set in the RUNNING state.

Figure 217 shows a state diagram of the start and stop mechanism.

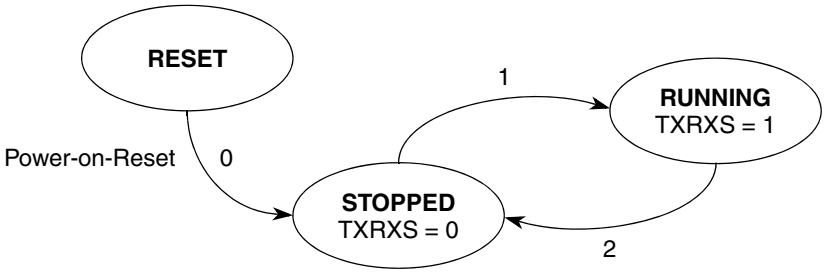


Figure 217. DSPI start and stop state diagram

The transitions are described in *Table 221*.

Table 221. State transitions for start and stop of DSPI transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: – EOQF bit is clear – Debug mode is unselected or the FRZ bit is clear – HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: – EOQF bit is set – Debug mode is selected and the FRZ bit is set – HALT bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

20.8.3 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*x*_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in *Section , "Transmit First In First Out (TX FIFO) buffering mechanism* and *Section , "Receive First In First Out (RX FIFO) buffering mechanism."*

The interrupt and DMA request conditions are described in *Section 20.8.7, "Interrupts/DMA requests."*

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. 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 of the TX FIFO entry is ignored.

SPI master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK_*x*) and the peripheral chip select ($\overline{\text{CS}}x$) signals. The SPI command field in the executing TX FIFO entry determines which CTARs set the transfer attributes and which $\overline{\text{CS}}x$ signals to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT_*x*) 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.

Refer to *Section , "DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the SPI command fields.

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*x*_CTAR0.

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 TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a 1 to the DIS_TXF bit in the DSPI*x*_MCR. The RX FIFO is disabled by writing a 1 to the DIS_RXF bit in the DSPI*x*_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI*x*_PUSHR and received data is read from the DSPI*x*_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if the FIFO must be disabled as a requirement of the application's operating mode. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and can result in incorrect results.

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 five entries, 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*x*_PUSHR). For more information on DSPI*x*_PUSHR refer to *Section , "DSPI PUSH TX FIFO Register (DSPIx_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*x*_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.

Refer to *Section , "DSPI Status Register (DSPIx_SR)* for more information on DSPI*x*_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI*x*_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI*x*_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.

Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI*x*_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPI*x*_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI*x*_PUSHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPI*x*_SR. The TFFF can generate a DMA request or an interrupt request.

Refer to *Section , "Transmit FIFO fill interrupt or DMA request (TFFF)* for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

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 is decremented by one. At the end of a transfer, the TCF bit in the DSPI*x*_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*x*_MCR.

If an external SPI 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*x*_SR is set.

Refer to *Section , "Transmit FIFO underflow interrupt request (TFUF)* for details.

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 five 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 is removed (popped) from the RX FIFO by reading the DSPI*x*_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI*x*_POPR or by flushing the RX FIFO.

Refer to *Section , "DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on the DSPI*x*_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPI*x*_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*x*_SR points to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from

DSPI*x*_RXFR0. For example, POPNXTPTR equal to two means that the DSPI*x*_RXFR2 contains the received SPI data that is returned when DSPI*x*_POPR is read. The POPNXTPTR field is incremented every time the DSPI*x*_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

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*x*_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI*x*_POPR. A read of the DSPI*x*_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

Refer to *Section , "DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on DSPI*x*_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPI*x*_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI*x*_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

20.8.4 DSPI baud rate and clock delay generation

The SCK_*x* 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 of doubling the baud rate.

Figure 218 shows conceptually how the SCK signal is generated.

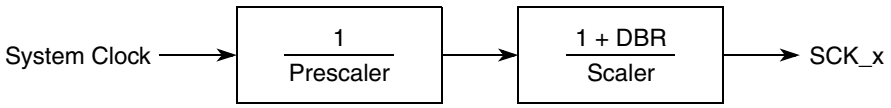


Figure 218. Communications clock prescalers and scalers

Baud rate generator

The baud rate is the frequency of the serial communication clock (SCK_*x*). The system clock is divided by a baud rate prescaler (defined by DSPI*x*_CTAR[PBR]) and baud rate scaler (defined by DSPI*x*_CTAR[BR]) to produce SCK_*x* with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI*x*_CTARs select the frequency of SCK_*x* using the following formula:

Equation 21

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 222 shows an example of a computed baud rate.

Table 222. Baud rate computation example

f_{SYS}	PBR	Prescaler value	BR	Scaler value	DBR value	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

CS to SCK delay (t_{CSC})

The $\overline{CS}\_x$ to SCK_*x* delay is the length of time from assertion of the $\overline{CS}\_x$ signal to the first SCK_*x* edge. Refer to *Figure 220* for an illustration of the $\overline{CS}\_x$ to SCK_*x* delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the $\overline{CS}\_x$ to SCK_*x* delay, and the relationship is expressed by the following formula:

Equation 22

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 223 shows an example of the computed $\overline{CS}$ to SCK_*x* delay.

Table 223. $\overline{CS}$ to SCK delay computation example

PCSSCK	Prescaler value	CSSCK	Scaler value	f_{SYS}	$\overline{CS}$ to SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

After SCK delay (t_{ASC})

The after SCK_*x* delay is the length of time between the last edge of SCK_*x* and the deassertion of $\overline{CS}\_x$. Refer to *Figure 220* and *Figure 221* for illustrations of the after SCK_*x* delay. The PASC and ASC fields in the DSPI*x*_CTAR*n* registers select the after SCK delay. The relationship between these variables is given in the following formula:

Equation 23

$$t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC} \times \text{ASC}$$

Table 224 shows an example of the computed after SCK delay.

Table 224. After SCK delay computation example

PASC	Prescaler value	ASC	Scaler value	f_{SYS}	After SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

Delay after transfer (t_{DT})

The delay after transfer is the length of time between negation of the $\overline{CSx}$ signal for a frame and the assertion of the $\overline{CSx}$ signal for the next frame. The PDT and DT fields in the DSPI*x*_CTAR*n* registers select the delay after transfer.

Refer to *Figure 220* for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

Equation 24

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 225 shows an example of the computed delay after transfer.

Table 225. Delay after transfer computation example

PDT	Prescaler value	DT	Scaler value	f_{SYS}	Delay after transfer
0b01	3	0b1110	32768	100 MHz	0.98 ms

Peripheral Chip Select strobe enable ($\overline{CS5\_x}$)

The $\overline{CS5\_x}$ signal provides a delay to allow the CS*x* signals to settle after transitioning, thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI*x*_MCR, CS5_*x* provides a signal for an external demultiplexer to decode the CS4_*x* signals into as many as 32 glitch-free CS*x* signals.

Figure 219 shows the timing of the CS5_*x* signal relative to CS signals.

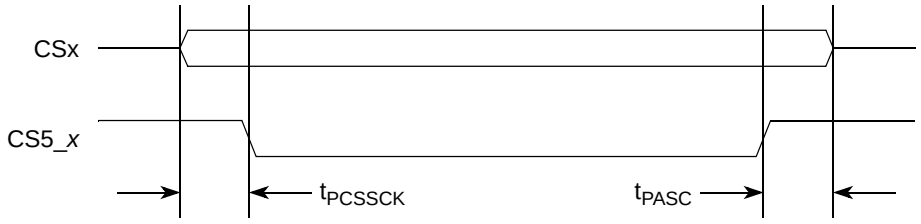


Figure 219. Peripheral Chip Select strobe timing

The delay between the assertion of the CS*x* signals and the assertion of CS5_*x* signal is selected by the PCSSCK field in the DSPI*x*_CTAR based on the following formula:

Equation 25

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5_*x* negation and CS*x* negation is selected by the PASC field in the DSPI*x*_CTAR based on the following formula:

Equation 26

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 226 shows an example of the computed t_{PCSSCK} delay.

Table 226. Peripheral Chip Select strobe assert computation example

PCSSCK	Prescaler	f_{SYS}	Delay before transfer
0b11	7	100 MHz	70.0 ns

Table 227 shows an example of the computed the t_{PASC} delay.

Table 227. Peripheral Chip Select strobe negate computation example

PASC	Prescaler	f_{SYS}	Delay after transfer
0b11	7	100 MHz	70.0 ns

20.8.5 Transfer formats

The SPI serial communication is controlled by the serial communications clock (SCK_*x*) signal and the CS*x* signals. The SCK_*x* signal provided by the master device synchronizes shifting and sampling of the data by the SIN_*x* and SOUT_*x* pins. The CS*x* 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*x*_CTAR*n*) select the polarity and phase of the serial clock, SCK_*x*. The polarity bit selects the idle state of the SCK_*x*. The clock phase bit selects if the data on SOUT_*x* is valid before or on the first SCK_*x* edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI*x*_CTAR0 (SPI slave mode) 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 device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

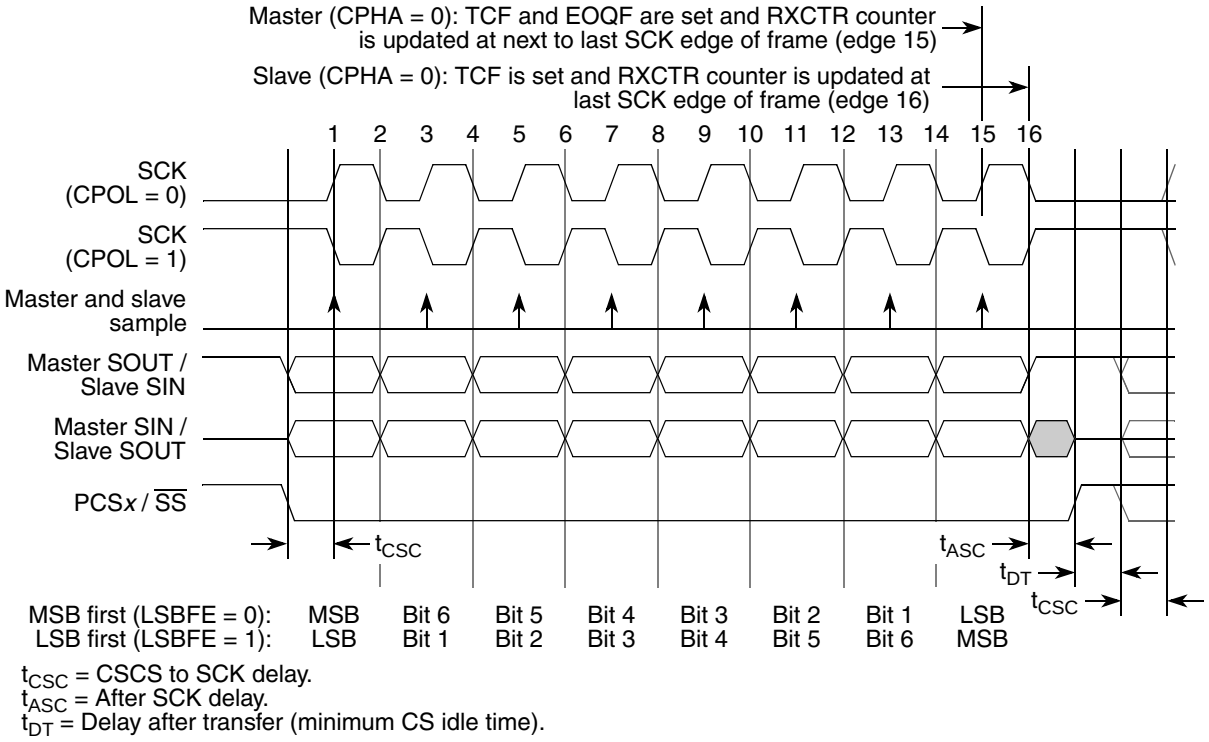
- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

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*x*_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in *Section , "Classic SPI transfer format (CPHA = 0)* and *Section , "Classic SPI transfer format (CPHA = 1)."* The modified transfer formats are described in *Section , "Modified SPI transfer format (MTFE = 1, CPHA = 0)* and *Section , "Modified SPI transfer format (MTFE = 1, CPHA = 1)."*

In the SPI configuration, the DSPI provides the option of keeping the CS signals asserted between frames. Refer to *Section , "Continuous selection format* for details.

Classic SPI transfer format (CPHA = 0)

The transfer format shown in *Figure 220* 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_*x* pins on the odd-numbered SCK_*x* edges and change the data on their SOUT_*x* pins on the even-numbered SCK_*x* edges.



t_{CSC} = CSCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS idle time).

Figure 220. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

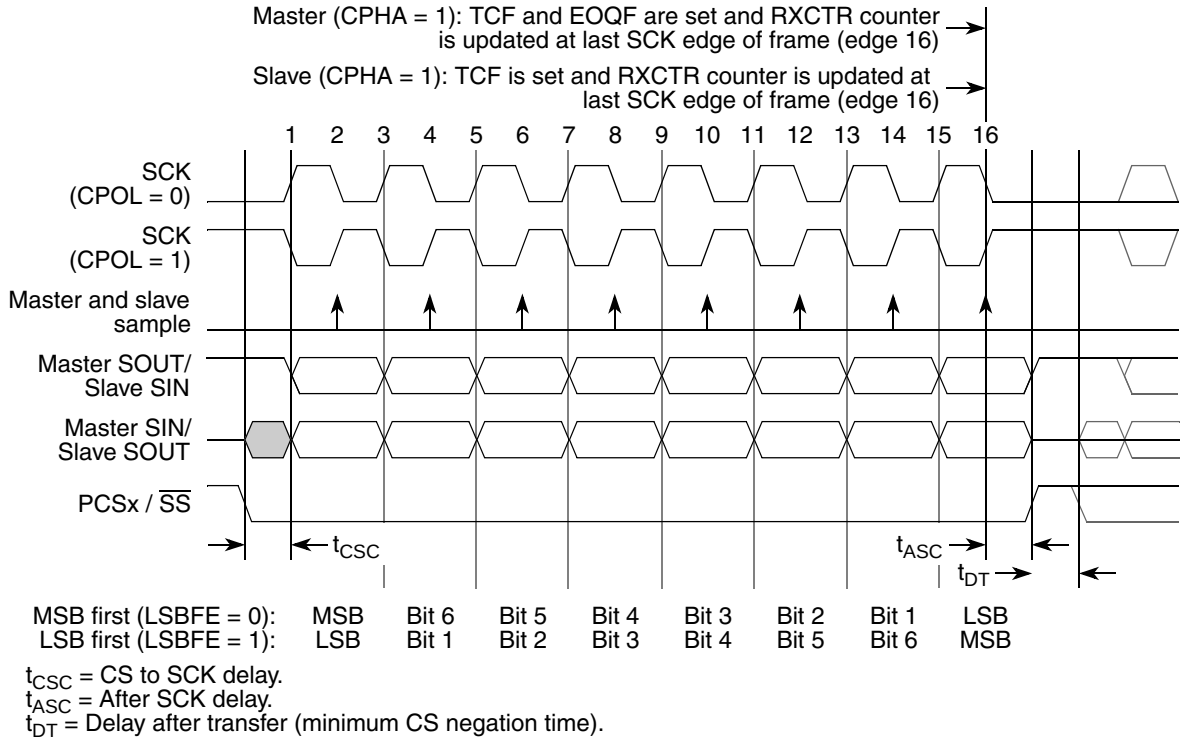
The master initiates the transfer by placing its first data bit on the SOUT_*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT_*x* pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK_*x*. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK_*x* 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_*x* pins on the odd-numbered clock edges and changes the data on their SOUT_*x* 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 CS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of *Figure 220*.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of *Figure 220*.

Classic SPI transfer format (CPHA = 1)

The transfer format shown in *Figure 221* is used to communicate with peripheral SPI slave devices that require the first SCK_*x* edge before the first data bit becomes available on the slave SOUT_*x* pin. In this format the master and slave devices change the data on their SOUT_*x* pins on the odd-numbered SCK_*x* edges and sample the data on their SIN_*x* pins on the even-numbered SCK_*x* edges.



t_{CSC} = CS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 221. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the CS*x* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK_*x* edge and at the same time places valid data on the master SOUT_*x* pin. The slave responds to the first SCK_*x* edge by placing its first data bit on its slave SOUT_*x* pin.

At the second edge of the SCK_*x* the master and slave sample their SIN_*x* pins. For the rest of the frame the master and the slave change the data on their SOUT_*x* pins on the odd-numbered clock edges and sample their SIN_*x* 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 CS*x* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of *Figure 221*. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

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 for 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.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT_*x* pins at the assertion of the CS*x* signal. After the CS*x* to SCK_*x* delay has elapsed the first SCK_*x* edge is generated. The slave samples the master SOUT_*x* signal on every odd numbered SCK_*x* edge. The slave also places new data on the slave SOUT_*x* on every odd numbered clock edge.

The master places its second data bit on the SOUT_*x* line one system clock after odd numbered SCK_*x* edge. The point where the master samples the slave SOUT_*x* is selected by writing to the SMPL_PT field in the DSPI*x*_MCR. *Table 228* lists the number of system clock cycles between the active edge of SCK_*x* and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 228. Delayed master sample point

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK and sampling of SIN
00	0
01	1
10	2
11	Invalid value

Figure 222 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

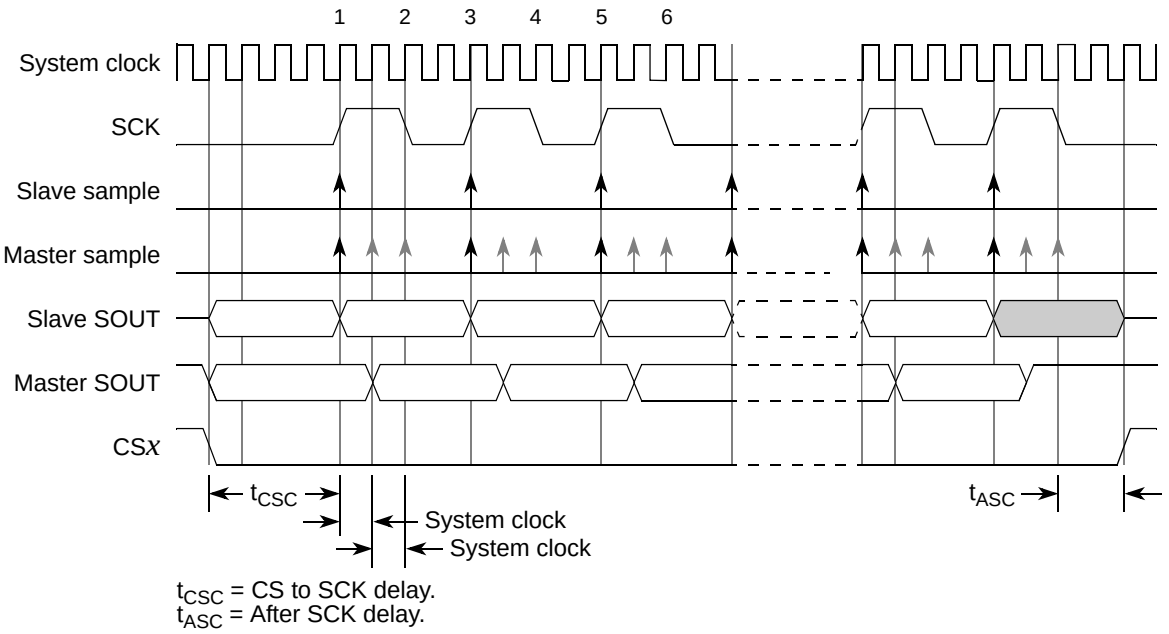


Figure 222. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$)

Modified SPI transfer format (MTFE = 1, CPHA = 1)

At the start of a transfer the DSPI asserts the CS signal to the slave device. After the CS 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 3rd 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. The SCK to CS delay must be programmed to be greater than or equal to half the SCK period.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 223 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is shown.

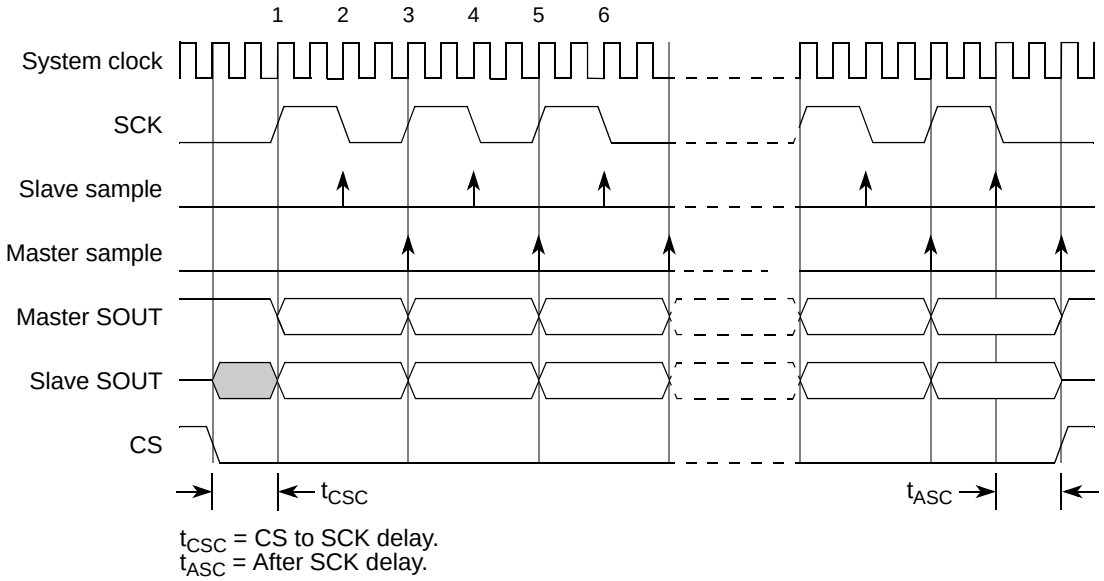


Figure 223. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$)

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 field in the DSPI*x*_MCR.

Figure 224 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

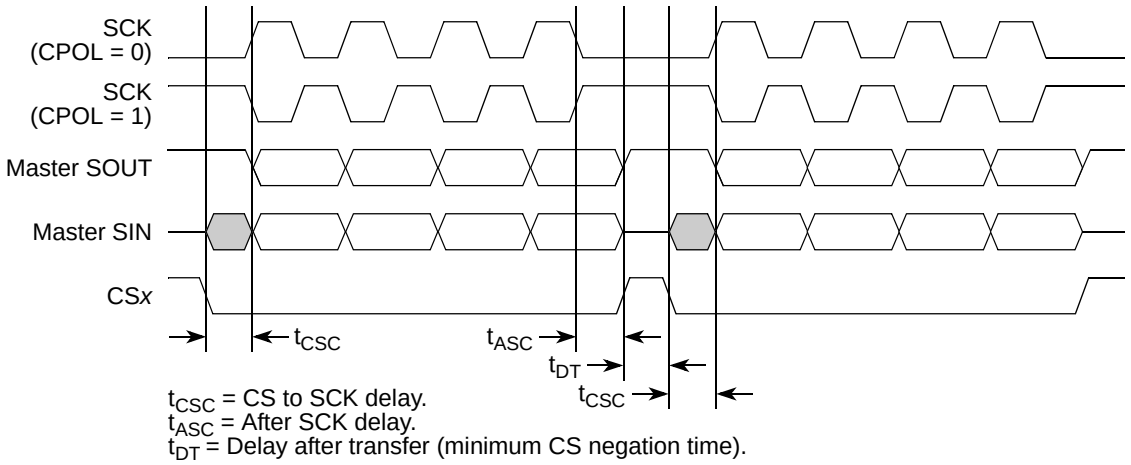


Figure 224. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT = 1 and the CS signal for the next transfer is the same as for the current transfer, the CS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers.

Figure 225 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

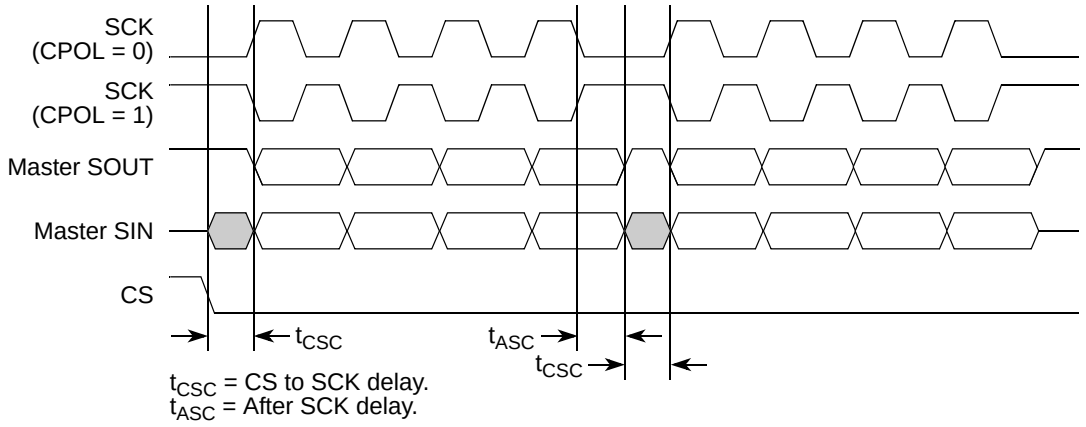


Figure 225. Example of continuous transfer (CPHA = 1, CONT = 1)

In *Figure 225*, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The CS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the CS signals for the next transfer are different from the present transfer, the CS signals behave as if the CONT bit was not set.

Note: It is mandatory to 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. For example; 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).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmited (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be de-selected for any further serial communication; else an underflow error occurs

Clock polarity switching between DSPI transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame.

Refer to *Section , "DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)*.

In *Figure 226*, time 'A' shows the one clock interval. Time 'B' is user programmable from a minimum of two system clocks.

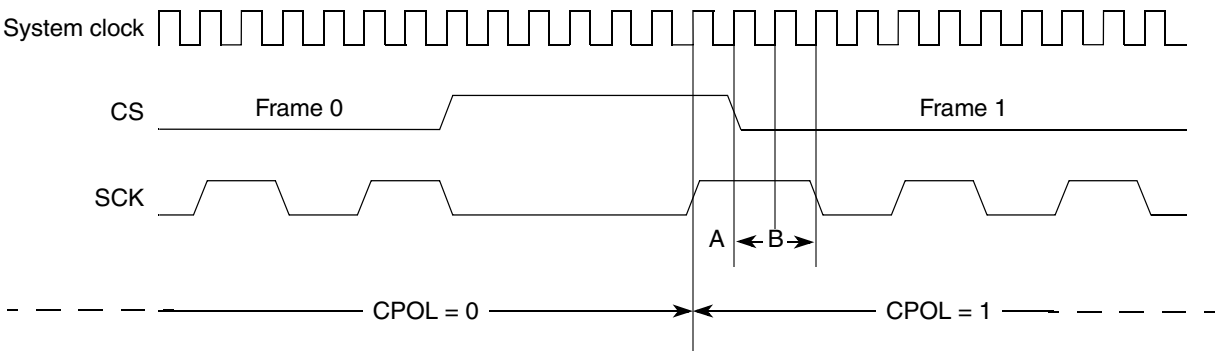


Figure 226. Polarity switching between frames

20.8.6 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*x*_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 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:

- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- In all configurations, 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.

The device is designed to use the same baud rate for all transfers made 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 module disable mode.

Enabling continuous SCK disables the CS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. *Figure 227* shows timing diagram for continuous SCK format with continuous selection disabled.

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.

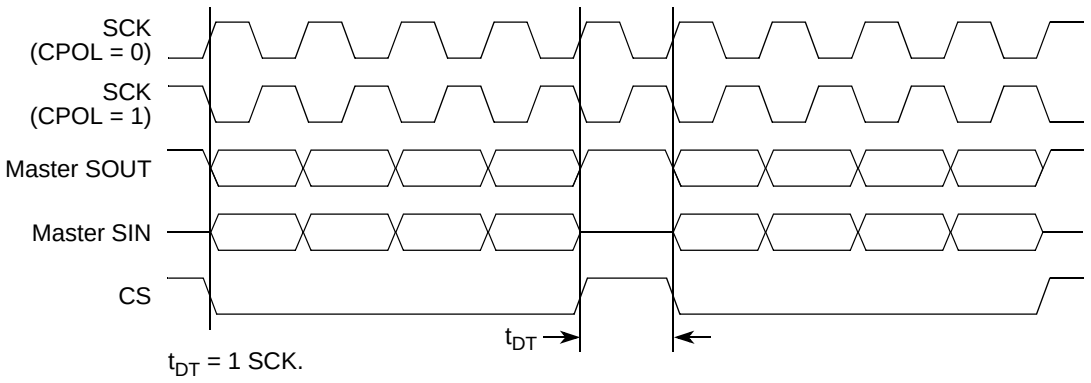


Figure 227. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set, CS remains asserted between the transfers when the CS signal for the next transfer is the same as for the current transfer. *Figure 228* shows timing diagram for continuous SCK format with continuous selection enabled.

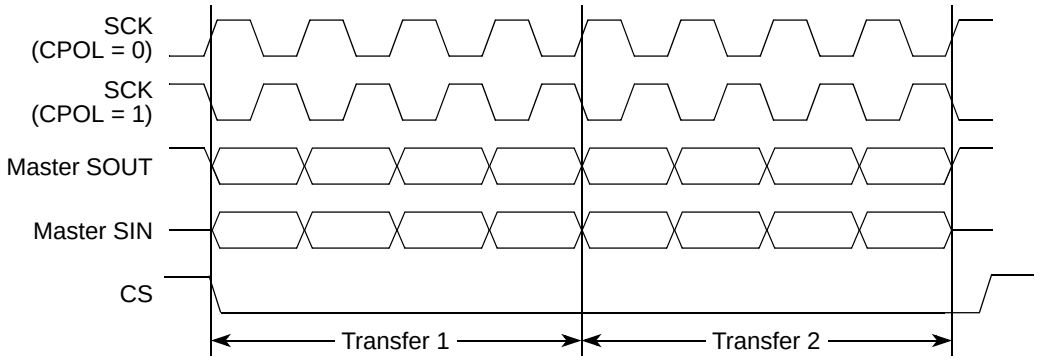


Figure 228. Continuous SCK timing diagram (CONT = 1)

20.8.7 Interrupts/DMA requests

The DSPI has conditions that can generate interrupt requests only, and conditions that can generate interrupts or DMA requests. *Table 229* lists these conditions.

Table 229. Interrupt and DMA request conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	
TX FIFO underflow has occurred	TFUF	X	
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow occurred	RFOF	X	
A FIFO overrun occurred(1)	TFUF ORed with RFOF	X	

1. The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the *Section , "DSPI Status Register (DSPIx_SR)* and the request enable bits are described in the *Section , "DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_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*x*_RSER.

End of queue interrupt request (EOQF)

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 asserted and the EOQF_RE bit in the DSPI*x*_RSER is set. Refer to the EOQ bit description in *Section , "DSPI Status Register (DSPIx_SR)*. Refer to *Figure 220* and *Figure 221* that illustrate when EOQF is set.

Transmit FIFO fill interrupt or DMA request (TFFF)

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 TFFF_RE bit in the DSPI*x*_RSER is set. The TFFF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

Transfer complete interrupt request (TCF)

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 TCF_RE bit is set in the DSPI*x*_RSER. Refer to the TCF bit description in *Section , "DSPI Status Register (DSPIx_SR)*. Refer to *Figure 220* and *Figure 221*, which show when TCF is set.

Transmit FIFO underflow interrupt request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI

operating in slave mode and SPI configuration 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*x*_RSER is set, an interrupt request is generated.

Receive FIFO drain interrupt or DMA request (RFDF)

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 RFDF_RE bit in the DSPI*x*_RSER is set. The RFDF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

Receive FIFO overflow interrupt request (RFOF)

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 RFOF_RE bit in the DSPI*x*_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI*x*_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 negated, the incoming data is ignored.

FIFO overrun request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing together the RX FIFO overflow and TX FIFO underflow signals.

20.8.8 Power saving features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

Module disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPI*x*_MCR.In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect 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 does not have any effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI*x*_MCR does not have any affect 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 affect. Writing to the DSPI*x*_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

20.9 Initialization and application information

20.9.1 Managing queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to manage queues for the DSPI.

1. The last command word from a queue is executed. 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 EOQ flag (EOQF) in the DSPI*x*_SR is set.
3. The setting of the EOQF flag disables both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues 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 eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI*x*_SR or by checking RFDF in the DSPI*x*_SR after each read operation of the DSPI*x*_POPR.
7. Modify DMA descriptor of TX and RX channels for “new” queues.
8. Flush TX FIFO by writing a 1 to the CLR_TXF bit in the DSPI*x*_MCR register and flush the RX FIFO by writing a 1 to the CLR_RXF bit in the DSPI*x*_MCR register.
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*x*_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.

20.9.2 Baud rate settings

Table 230 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*x*_CTARs. The values are calculated at a 100 MHz system frequency.

Table 230. Baud rate values

		Baud Rate divider prescaler values (DSPI_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud rate scaler values (DSPI_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

20.9.3 Delay settings

Table 231 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*x*_CTARs. The values calculated assume a 100 MHz system frequency.

Table 231. Delay values

		Delay prescaler values (DSPI_CTAR[PBR])			
		1	**3**	**5**	**7**
Delay scaler values (DSPI_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

20.9.4 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).

Refer to *Section , "Transmit First In First Out (TX FIFO) buffering mechanism* and *Section , "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 229 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

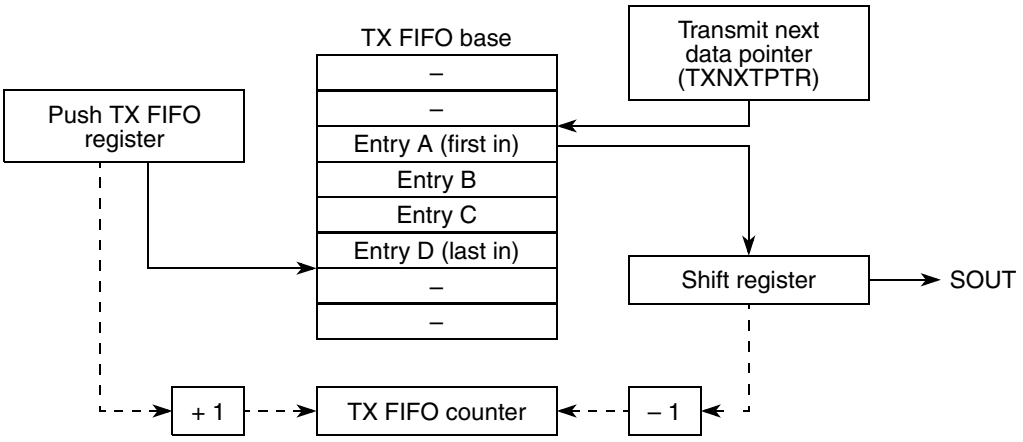


Figure 229. TX FIFO pointers and counter

Address calculation for first-in entry and last-in entry in TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

Equation 27

$$\text{First-in entry address} = \text{TXFIFO base} + 4\ (\text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 28

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXTPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXTPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth (depth is 5)

Address calculation for first-in entry and last-in entry in RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

Equation 29

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 30

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPNXTPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO

RXCTR = receive FIFO counter

POPNXTPTR = pop next pointer

RX FIFO depth = receive FIFO depth (depth is 5)

21 LIN Controller (LINFlex)

21.1 Introduction

The LINFlex (Local Interconnect Network Flexible) 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. LINFlex 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.

21.2 Main features

21.2.1 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 (IAP) purposes
- Wake-up 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 16 MHz fast internal RC oscillator as clock source
- 16 identifier filters for autonomous message handling in Slave mode

21.2.2 UART mode features

- Full duplex communication
- 8- or 9-bit with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

21.2.3 Features common to LIN and UART

- Fractional baud rate generator
- 3 operating modes for power saving and configuration registers lock:
  - Initialization
  - Normal
  - Sleep
- 2 test modes:
  - Loop Back
  - Self Test
- Maskable interrupts

21.3 General description

The increasing number of communication peripherals embedded on microcontrollers, for example CAN, LIN and SPI, requires more and more CPU resources for 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 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as usually the case.

To minimize the CPU load in Master mode, LINFlex handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlex does not request any software intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlex 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, LINFlex 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.

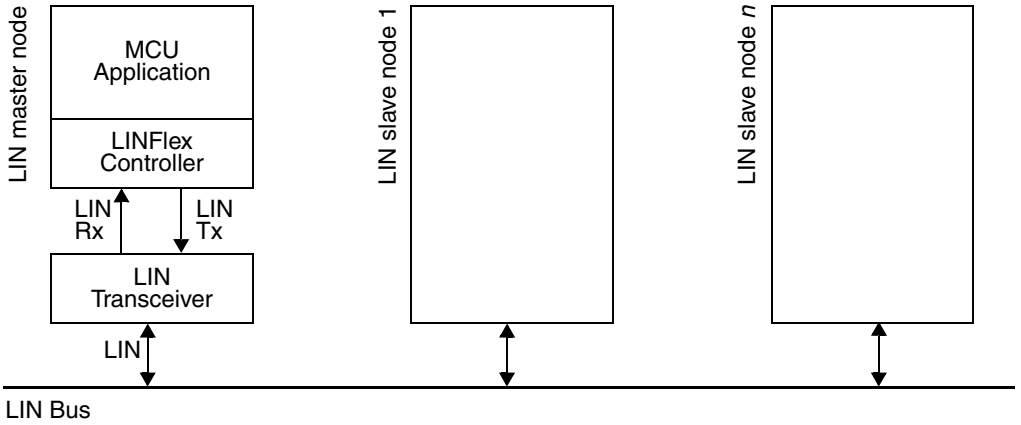
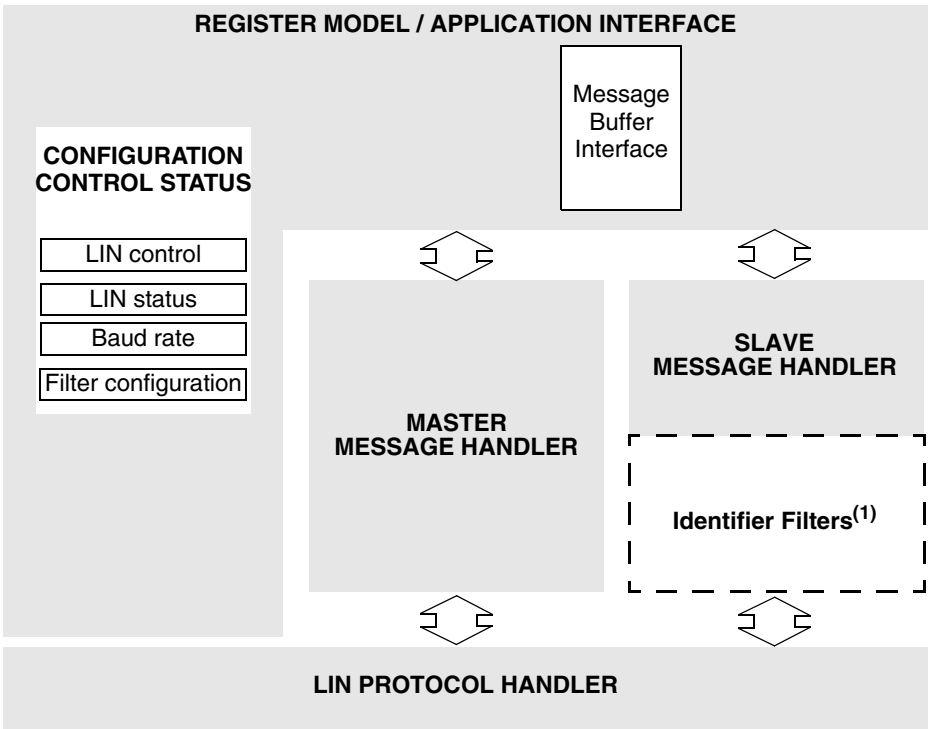


Figure 230. LIN topology network



1. Filter activation optional

Figure 231. LINFlex block diagram

21.4 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBRR) registers.

Equation 31

$$\text{Tx/ Rx baud} = \frac{f_{periph\_set\_1\_clk}}{(16 \times LFDIV)}$$

LFDIV is an unsigned fixed point number. The 12-bit mantissa is coded in the LINIBRR and the fraction is coded in the LINFBRR.

The following examples show how to derive LFDIV from LINIBRR and LINFBRR register values:

Example 11 Deriving LFDIV from LINIBRR and LINFBRR register values

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

Example 12 Programming LFDIV from LINIBRR and LINFBRR register values

To program LFDIV = 25.62d,

LINFBRR = 16 × 0.62 = 9.92, nearest real number 10d = 0xA

LINIBRR = mantissa (25.620d) = 25d = 0x19

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 the LINIBRR.

Note: LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR = 1 and LINFBRR = 8. Therefore, the maximum possible baudrate is fperiph_set_1_clk / 24.

Table 232. Error calculation for programmed baud rates

Baud rate	$f_{periph\_set\_1\_clk}$ = 64 MHz				$f_{periph\_set\_1\_clk}$ = 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	384	0	–0.003	10416.7	96	0	–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
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

21.5 Operating modes

LINFlex has three main operating modes: Initialization, Normal and Sleep. After a hardware reset, LINFlex is in Sleep mode to reduce power consumption. The software instructs LINFlex to enter Initialization mode or Sleep mode by setting the INIT bit or SLEEP bit in the LINCR1.

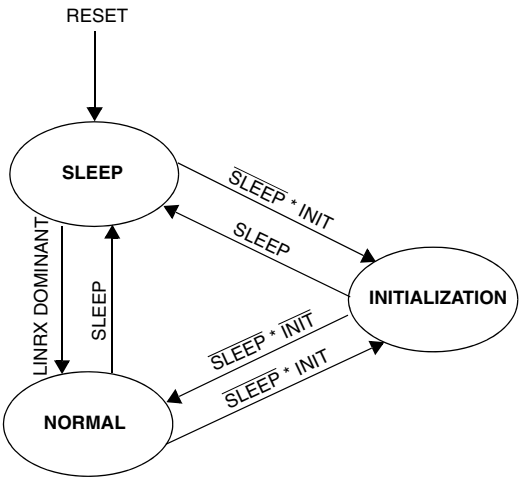


Figure 232. LINFlex operating modes

21.5.1 Initialization mode

The software can be initialized while the hardware is in Initialization mode. To enter this mode the software sets the INIT bit in the LINCR1.

To exit Initialization mode, the software clears the INIT bit.

While in Initialization mode, all message transfers to and from the LIN bus are stopped and the status of the LIN bus output LINTX is recessive (high).

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlex controller, the software selects the mode (LIN Master, LIN Slave or UART), sets up the baud rate register and, if LIN Slave mode with filter activation is selected, initializes the identifier list.

21.5.2 Normal mode

Once initilization is complete, software clears the INIT bit in the LINCR1 to put the hardware into Normal mode.

21.5.3 Low power mode (Sleep)

To reduce power consumption, LINFlex has a low power mode called Sleep mode. To enter Sleep mode, software sets the SLEEP bit in the LINCR1. In this mode, the LINFlex clock is stopped. Consequently, the LINFlex will not update the status bits but software can still access the LINFlex registers.

LINFlex can be awakened (exit Sleep mode) either by software clearing the SLEEP bit or on detection of LIN bus activity if automatic wake-up mode is enabled (AWUM bit is set).

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the LINCR1 is set. To exit from Sleep mode if the AWUM bit is cleared, software clears the SLEEP bit when a wake-up event occurs.

21.6 Test modes

Two test modes are available to the user: Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1. These bits must be configured while LINFlex is in Initialization mode. Once one of the two test modes has been selected, LINFlex must be started in Normal mode.

21.6.1 Loop Back mode

LINFlex can be put in Loop Back mode by setting the LBKM bit in the LINCR. In Loop Back mode, the LINFlex treats its own transmitted messages as received messages.

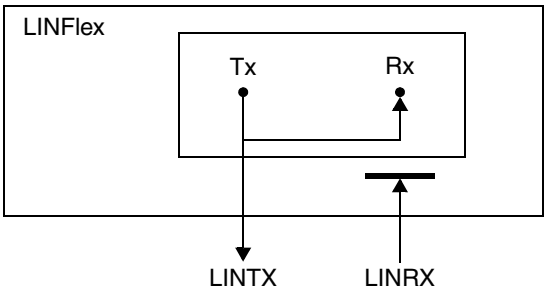


Figure 233. LINFlex in loop back mode

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 LINFlex performs an internal feedback from its Tx output to its Rx input. The actual value of the LINRX input pin is disregarded by the LINFlex. The transmitted messages can be monitored on the LINTX pin.

21.6.2 Self Test mode

LINFlex can be put in Self Test mode by setting the LBKM and SFTM bits in the LINCR. This mode can be used for a “Hot Self Test”, meaning the LINFlex can be tested as in Loop Back 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 LINFlex and the LINTX pin is held recessive.

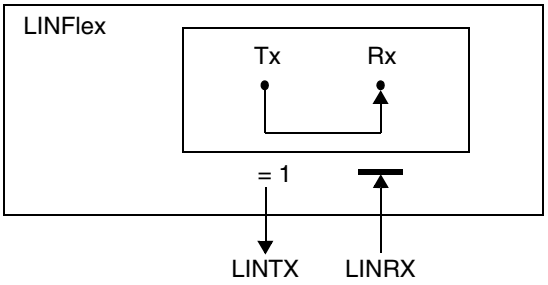


Figure 234. LINFlex in self test mode

21.7 Memory map and registers description

21.7.1 Memory map

See the “Memory map” chapter of this reference manual for the base addresses for the LINFlex modules.

Table 233 shows the LINFlex memory map.

Table 233. LINFlex memory map

Address offset	Register	Location
0x0000	LIN control register 1 (LINCR1)	*on page 21-492*
0x0004	LIN interrupt enable register (LINIER)	*on page 21-495*
0x0008	LIN status register (LINSR)	*on page 21-497*
0x000C	LIN error status register (LINESR)	*on page 21-500*
0x0010	UART mode control register (UARTCR)	*on page 21-501*
0x0014	UART mode status register (UARTSR)	*on page 21-503*
0x0018	LIN timeout control status register (LINTCSR)	*on page 21-505*
0x001C	LIN output compare register (LINOCR)	*on page 21-506*
0x0020	LIN timeout control register (LINTOCR)	*on page 21-506*
0x0024	LIN fractional baud rate register (LINFBRR)	*on page 21-507*
0x0028	LIN integer baud rate register (LINIBRR)	*on page 21-508*
0x002C	LIN checksum field register (LINCFR)	*on page 21-509*
0x0030	LIN control register 2 (LINCR2)	*on page 21-509*
0x0034	Buffer identifier register (BIDR)	*on page 21-511*
0x0038	Buffer data register LSB (BDRL)(1)	*on page 21-512*
0x003C	Buffer data register MSB (BDRM)(2)	*on page 21-512*
0x0040	Identifier filter enable register (IFER)	*on page 21-513*

Table 233. LINFlex memory map (continued)

Address offset	Register	Location
0x0044	Identifier filter match index (IFMI)	*on page 21-514*
0x0048	Identifier filter mode register (IFMR)	*on page 21-514*
0x004C	Identifier filter control register 0 (IFCR0)	*on page 21-516*
0x0050	Identifier filter control register 1 (IFCR1)	*on page 21-517*
0x0054	Identifier filter control register 2 (IFCR2)	*on page 21-517*
0x0058	Identifier filter control register 3 (IFCR3)	*on page 21-517*
0x005C	Identifier filter control register 4 (IFCR4)	*on page 21-517*
0x0060	Identifier filter control register 5 (IFCR5)	*on page 21-517*
0x0064	Identifier filter control register 6 (IFCR6)	*on page 21-517*
0x0068	Identifier filter control register 7 (IFCR7)	*on page 21-517*
0x006C	Identifier filter control register 8 (IFCR8)	*on page 21-517*
0x0070	Identifier filter control register 9 (IFCR9)	*on page 21-517*
0x0074	Identifier filter control register 10 (IFCR10)	*on page 21-517*
0x0078	Identifier filter control register 11 (IFCR11)	*on page 21-517*
0x007C	Identifier filter control register 12 (IFCR12)	*on page 21-517*
0x0080	Identifier filter control register 13 (IFCR13)	*on page 21-517*
0x0084	Identifier filter control register 14 (IFCR14)	*on page 21-517*
0x0088	Identifier filter control register 15 (IFCR15)	*on page 21-517*
0x008C–0x000F	Reserved	

1. LSB: Least significant byte
2. MSB: Most significant byte

LIN control register 1 (LINCR1)

Figure 235. LIN control register 1 (LINCR1)

Offset: 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 W	CCD	CFD	LASE	AWUM	MBL				BF	SFTM	LBKM	MME	SBDT	RBLM	SLEEP	INIT
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Table 234. LINCR1 field descriptions

Field	Description
CCD	Checksum calculation disable This bit disables the checksum calculation (see *Table 235*). 0 Checksum calculation is done by hardware. When this bit is 0, the LINCFR is read-only. 1 Checksum calculation is disabled. When this bit is set the LINCFR is read/write. User can program this register to send a software-calculated CRC (provided CFD is 0). Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
CFD	Checksum field disable This bit disables the checksum field transmission (see *Table 235*). 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. 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 LINFlex hardware during Sleep mode. 0 The Sleep mode is exited on software request by clearing the SLEEP bit of the LINCR. 1 The Sleep mode is exited automatically by hardware on LINRX dominant state detection. The SLEEP bit of the LINCR is cleared by hardware whenever WUF bit in the LINSR is set. Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
MBL	LIN Master Break Length This field indicates the Break length in Master mode (see *Table 236*). Note: This field can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
BF	Bypass filter 0 No interrupt if identifier does not match any filter. 1 An RX interrupt is generated on identifier not matching any filter. – 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 21.6.2, 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.

Table 234. LINCR1 field descriptions (continued)

Field	Description
LBKM	Loop Back Mode This bit controls the Loop Back mode. For more details see *Section 21.6.1, Loop Back mode.* 0 Loop Back mode disable. 1 Loop Back 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.
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 LINFlex to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see *Table 237*).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see *Table 237*).

Table 235. 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 236. LIN master break length selection

MBL	Length
0000	10-bit

Table 236. LIN master break length selection (continued)

MBL	Length
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit
1000	18-bit
1001	19-bit
1010	20-bit
1011	21-bit
1100	22-bit
1101	23-bit
1110	36-bit
1111	50-bit

Table 237. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

LIN interrupt enable register (LINIER)

Figure 236. LIN interrupt enable register (LINIER)

Offset: 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	DBEIE	DRIE	DTIE	HRIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 238. 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 BEF bit in LINESR is set. 1 Interrupt generated when BEF bit in LINESR is set.
CEIE	Checksum Error Interrupt Enable 0 No interrupt on Checksum error. 1 Interrupt generated when checksum error flag (CEF) in LINESR is set.
HEIE	Header Error Interrupt Enable 0 No interrupt on Break Delimiter error, Synch Field error, Identifier field error. 1 Interrupt generated on Break Delimiter error, Synch Field error, Identifier 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 LINS[0:3] in the LINSR.
WUIE	Wake-up 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.
DBEIE	Data Buffer Empty Interrupt Enable 0 No interrupt when buffer data register is empty. 1 Interrupt generated when data buffer register is empty.
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt when data reception is completed. 1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.
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.

Table 238. LINIER field descriptions (continued)

Field	Description
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 is set.

LIN status register (LINSR)

Figure 237. LIN status register (LINSR)

Offset: 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	LINS				0	0	RMB	0	RBSY	RPS	WUF	DBFF	DBEF	DRF	DTF	HRF
W							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

Table 239. LINSR field descriptions

Field	Description
LINS	LIN modes / normal mode states **0000: Sleep mode** LINFlex is in Sleep mode to save power consumption. **0001: Initialization mode** LINFlex is in Initialization mode. Normal mode states **0010: Idle** This state is entered on several events: – SLEEP bit and INIT bit in LINCR1 have been cleared by software, – A falling edge has been received on RX 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. Note: 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. See *Section , LIN control register 1 (LINCR1)* 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: Synch Field** In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field. In Master mode, Synch Field transmission is ongoing. **0110: Identifier Field** In Slave mode, a valid Synch Field has been received. Receiving Identifier 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 the 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

Table 239. LINSR field descriptions (continued)

Field	Description
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 BIDR[DIR] = 0 and reception starts then this bit is set. In this case, user cannot program LINCR2[DTRQ] = 1.
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex 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.

Table 239. LINSR field descriptions (continued)

Field	Description
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 – TX filter match

LIN error status register (LINESR)

Figure 238. LIN error status register (LINESR)

Offset: 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

Table 240. 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. If the dominant state continues, SZF flag is set again after 87-bit time. It is cleared by software.
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. If this bit is set and IOT bit in LINTCSR is set, LINFlex moves to Idle state. If LTOM bit in LINTCSR is set, then OCF is cleared by hardware in Initialization mode. If LTOM bit is cleared, then OCF maintains its status whatever the mode is.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlex 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.

Table 240. LINESR field descriptions (continued)

Field	Description
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 CCD or CFD bit in LINCR1 is set.
SFEF	Synch Field Error Flag This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch 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 a Identifier Parity error occurred. Note: Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex 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 Synch Field or Identifier Field in Slave mode.
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 RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It 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.

UART mode control register (UARTCR)

Figure 239. UART mode control register (UARTCR)

Offset: 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	TDFL		0	RDFL		0	0	0	0	RXEN	TXEN	OP	PCE	WL	UART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 241. UARTCR field descriptions

Field	Description
TDFL	Transmitter Data Field length This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1. 00 Transmit buffer size = 1. 01 Transmit buffer size = 2. 10 Transmit buffer size = 3. 11 Transmit buffer size = 4.
RDFL	Receiver Data Field length This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1. 00 Receive buffer size = 1. 01 Receive buffer size = 2. 10 Receive buffer size = 3. 11 Receive buffer size = 4.
RXEN	Receiver Enable 0 Receiver disable. 1 Receiver enable. This bit can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disable. 1 Transmitter enable. This bit 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.
OP	Odd Parity 0 Sent parity is even. 1 Sent parity is odd. This bit can be programmed in Initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disable. 1 Parity transmit/check enable. This bit can be programmed in Initialization mode only when the UART bit is set.
WL	Word Length in UART mode 0 7-bit data + parity bit. 1 8-bit data (or 9-bit if PCE is set). This bit can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0 LIN mode. 1 UART mode. This bit can be programmed in Initialization mode only.

UART mode status register (UARTSR)

Figure 240. UART mode status register (UARTSR)

Offset: 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	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	0	DRF	DTF	NF
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

Table 242. 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 there is a parity error in the corresponding received byte (Rx3). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE2	Parity Error Flag Rx2 This bit indicates if there is a parity error in the corresponding received byte (Rx2). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE1	Parity Error Flag Rx1 This bit indicates if there is a parity error in the corresponding received byte (Rx1). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE0	Parity Error Flag Rx0 This bit indicates if there is a parity error in the corresponding received byte (Rx0). See *Section , Buffer in UART mode.* No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.

Table 242. UARTSR field descriptions (continued)

Field	Description
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 LINFlex has detected a framing error (invalid stop bit).
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 RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites buffer. it can be cleared by software.
RPS	LIN Receive Pin State This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex 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 i generated if WUIE bit in LINIER is set.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DRIE bit in LINIER is set. In UART mode, this flag is set in case of framing error, parity error or overrun.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed, that is, the number of bytes programmed in TDFL[0:1] have been transmitted. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DTIE bit in LINIER is set.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

LIN timeout control status register (LINTCSR)

Figure 241. LIN timeout control status register (LINTCSR)

Offset: 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																
Reset	0	0	0	0	0	0	0	0	0	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	IOT	TOCE	CNT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Table 243. LINTCSR field descriptions

Field	Description
LTOM	LIN timeout mode 0 LIN timeout mode (header, response and frame timeout detection). 1 Output compare mode. 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. 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. 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 This field indicates the LIN timeout counter value.

LIN output compare register (LINOCR)

Figure 242. LIN output compare register (LINOCR)

Offset: 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	OC2[1]								OC1[1]							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1. If LINTCSR[LTOM] = 0, this field is read-only.

Table 244. LINOCR field descriptions

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.
OC1	Output compare 1 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.

LIN timeout control register (LINTOCR)

Figure 243. LIN timeout control register (LINTOCR)

Offset: 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	0	0	0	0	RTO				0	HTO						
W																
Reset	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0

Table 245. LINTOCR field descriptions

Field	Description
RTO	Response timeout value This field 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 field contains the header timeout duration (in bit time). This value does not include the Break and the Break Delimiter. The reset value is the 0x2C = 44, corresponding to $T_{Header\_Maximum.}$ Programming LINSR[MME] = 1 changes the HTO value to 0x1C = 28. This field can be written only in Slave mode.

LIN fractional baud rate register (LINFBRR)

Figure 244. LIN fractional baud rate register (LINFBRR)

Offset: 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	0	0	0	0	0	0	0	0	0	0	0	0	DIV_F			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 246. LINFBRR field descriptions

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This field can be written in Initialization mode only.

LIN integer baud rate register (LINIBRR)

Figure 245. LIN integer baud rate register (LINIBRR)

Offset: 0x0028 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	DIV_M												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 247. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see *Table 248*). This field can be written in Initialization mode only.

Table 248. Integer baud rate selection

DIV_M[0:12]	Mantissa
0x0000	LIN clock disabled
0x0001	1
...	...
0x1FFE	8190
ox1FFF	8191

LIN checksum field register (LINCFR)

Figure 246. LIN checksum field register (LINCFR)

Offset: 0x002C 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

Table 249. LINCFR field descriptions

Field	Description
CF	Checksum bits When LINCR1[CCD] = 0, this field is read-only. When LINCR1[CCD] = 1, this field is read/write. See *Table 235*.

LIN control register 2 (LINCR2)

Figure 247. LIN control register 2 (LINCR2)

Offset: 0x0030 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	IOPE	0	0	0	0	0	0	0	0	0	0	0	0	0
W				WURQ	DDRQ	DTRQ	ABRQ	HTRQ								
Reset	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 250. 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.
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.
WURQ	Wake-up Generation Request Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up 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 wake-up. Bit error is not checked when transmitting the wake-up 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 LINFlex 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 BIDR[DIR] = 1 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. LINFlex aborts the transmission at the end of the current bit. This bit can also abort a wake-up 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.

Buffer identifier register (BIDR)

Figure 248. Buffer identifier register (BIDR)

Offset: 0x0034 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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 251. BIDR field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1. Normally, LIN uses only DFL[2:0] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[2:0] only. DFL[5:3] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDR registers. 1 LINFlex transmits the data from the BDR 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 earlier. In LIN slave mode (MME bit cleared in LINCR1), this bit must be configured before the header reception. If the slave has to manage frames with 2 types of checksum, filters must be configured.
ID	Identifier Identifier part of the identifier field without the identifier parity.

Buffer data register LSB (BDRL)

Figure 249. Buffer data register LSB (BDRL)

Offset: 0x0038 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA3								DATA2							
Reset	0	0	0	0	0	0	0	0	0	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	DATA1								DATA0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 252. 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.

Buffer data register MSB (BDRM)

Figure 250. Buffer data register MSB (BDRM)

Offset: 0x003C 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

Table 253. 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.

Identifier filter enable register (IFER)

Figure 251. Identifier filter enable register (IFER)

Offset: 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	FACT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 254. IFER field descriptions

Field	Description
FACT	Filter activation The software sets the bit FACT[x] to activate the filters x in identifier list mode. In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n. 0 Filter x is deactivated. 1 Filter x is activated. This field can be set/cleared in Initialization mode only.

Identifier filter match index (IFMI)

Figure 252. Identifier filter match index (IFMI)

Address: Base + 0x0044 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	IFMI[0:4]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 255. IFMI field descriptions

Field	Description
0:26	Reserved
IFMI[0:4] 27:31	Filter match index This register contains the index corresponding to the received identifier. It can be used to directly write or read the data in SRAM (see *Section , Slave mode* for more details). When no filter matches, IFMI[0:4] = 0. When Filter *n* is matching, IFMI[0:4] = *n* + 1.

Identifier filter mode register (IFMR)

Figure 253. Identifier filter mode register (IFMR)

Offset: 0x0048 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	IFM							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 256. IFMR field descriptions

Field	Description
IFM	Filter mode (see *Table 257*). 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*).

Table 257. IFMR[IFM] configuration

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

Identifier filter control register (IFCR2*n*)

Figure 254. Identifier filter control register (IFCR2*n*)

Offsets: 0x004C–0x0084 (8 registers) 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	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register bit can be read in any mode, written only in Initialization mode

Table 258. IFCR2*n* 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 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex 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 earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity.

Identifier filter control register (IFCR2*n* + 1)

Figure 255. Identifier filter control register (IFCR2*n* + 1)

Offsets: 0x0050–0x0088 (8 registers) 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	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Register bit can be read in any mode, written only in Initialization mode

Table 259. IFCR2*n* + 1 field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex 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 field only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity

21.8 Functional description

21.8.1 UART mode

The main features in the UART mode are

- Full duplex communication
- 8- or 9-bit data with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

8-bit data frames: The 8th bit can be a data or a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. 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.

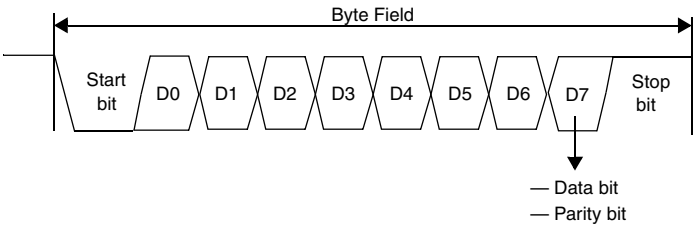


Figure 256. UART mode 8-bit data frame

9-bit frames: The 9th bit is a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. 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.

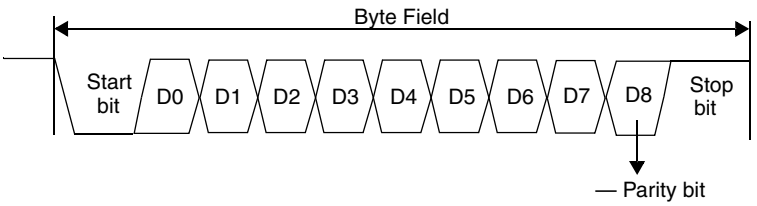


Figure 257. UART mode 9-bit data frame

Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter as shown in *Table 260*.

Table 260. Message buffer

Buffer data register	LIN mode		UART mode	
BDRL[0:31]	Transmit/Receive buffer	DATA0[0:7]	Transmit buffer	Tx0
		DATA1[0:7]		Tx1
		DATA2[0:7]		Tx2
		DATA3[0:7]		Tx3
BDRM[0:31]		DATA4[0:7]	Receive buffer	Rx0
		DATA5[0:7]		Rx1
		DATA6[0:7]		Rx2
		DATA7[0:7]		Rx3

UART transmitter

In order to start transmission in UART mode, you must program the UART bit and the transmitter enable (TXEN) bit in the UARTCR to 1. Transmission starts when DATA0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by UARTCR[TDFL] (see *Table 241*).

The Transmit buffer is 4 bytes, hence a 4-byte maximum transmission can be triggered. Once the programmed number of bytes has been transmitted, the UARTSR[DTF] bit is set. If UARTCR[TXEN] is reset during a transmission then the current transmission is completed and no further transmission can be invoked.

UART receiver

The UART receiver is active as soon as the user exits Initialization mode and programs UARTCR[RXEN] = 1. There is a dedicated 4-byte data buffer for received data bytes. Once the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRF] bit is set. If the RXEN bit is reset during a reception then the current reception is completed and no further reception can be invoked until RXEN is set.

If a parity error occurs during reception of any byte, then the corresponding PE*x* bit in the UARTSR is set. No interrupt is generated in this case. If a framing error occurs in any byte (UARTSR[FE] = 1) then an interrupt is generated if the LINIER[FEIE] bit is set.

If the last received frame has not been read from the buffer (that is, RMB bit is not reset by the user) then upon reception of the next byte an overrun error occurs (UARTSR[BOF] = 1) and one message will be lost. Which message is lost depends on the configuration of LINCR1[RBLM].

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) 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 application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

An interrupt is generated if the LINIER[BOIE] bit is set.

Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In UART mode, the LINFlex controller acknowledges a clock gating request once the data transmission and data reception are completed, that is, once the Transmit buffer is empty and the Receive buffer is full.

21.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode
- Slave mode with identifier filtering
- Slave mode with automatic resynchronization

These submodes are described in the following pages.

Master mode

In Master mode the application uses the message buffer to handle the LIN messages. Master mode is selected when the LINCR1[MME] bit is set.

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 LINFlex the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR before requesting the header transmission by setting LINCR2[HTRQ].

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 application must provide the data to LINFlex before requesting the header transmission. The application stores the data in the message buffer BDR. According to the data field length, LINFlex transmits the data and the checksum. The application uses the BDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, the LINSR[DTF] bit is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (see *Section , Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBEF] bit is set after the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset.

After the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the application sets this bit the response is sent by LINFlex (publisher). Resetting this bit configures the message buffer as subscriber.

Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlex stores the data received from the slave in the message buffer and stores the message status in the LINSR.

If the response has been received successfully, the LINSR[DRF] is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (see *Section , Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBFF] bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

Data discard

To discard data from a slave, the BIDR[DIR] bit must be reset and the LINCR2[DDRQ] bit must be set before starting the header transmission.

Error detection

LINFlex 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.

In Master mode, the following errors are detected:

- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.
- **Response and Frame timeout**: See *Section 21.8.3, 8-bit timeout counter*, for more details.

Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if LINIER[BEIE] = 1.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting

the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer is overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.

Slave mode

In Slave mode the application uses the message buffer to handle the LIN messages. Slave mode is selected when LINCR1[MME] = 0.

Data transmission (transceiver as publisher)

When LINFlex receives the identifier, the LINSR[HRF] is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR, fill the BDR registers, specify the data field length using the BIDR[DFL] and trigger the data transmission by setting the LINCR2[DTRQ] bit.

One or several identifier filters can be configured for transmission by setting the IFCR*x*[DIR] bit and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in transmission and activated, and if the received identifier matches the filter, a specific TX interrupt (instead of an RX interrupt) is generated.

Typically, the application has to copy the data from SRAM locations to the BDR. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex 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 SRAM area and copy this data to the BDR (see *Figure 259*).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BIDR. The software fills the BDR and triggers the data transmission by programming LINCR2[DTRQ] = 1.

If LINFlex cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (see *Section , Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

Data reception (transceiver as subscriber)

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR and specify the data field length using the BIDR[DFL] field before receiving the stop bit of the first byte of data field.

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by programming IFCR*x*[DIR] = 0 and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in reception and activated, and if the received identifier matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the application has to copy the data from the BDR to SRAM locations. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex 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 SRAM area and copy this data from the BDR to the SRAM (see *Figure 259*).

Using a filter avoids the software reading the ID value in the BIDR, and configuring the direction, the data field length and the checksum type in the BIDR.

If LINFlex cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (see *Section , Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

Data discard

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. If the received identifier does not concern the node, you must program LINCR2[DDRQ] = 1. LINFlex returns to idle state after bit DDRQ is set.

Error detection

In Slave mode, the following errors are detected:

- **Header error**: An error occurred during header reception (Break Delimiter error, Inconsistent Synch Field, Header Timeout).
- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.

Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if the BEIE bit in the LINIER is set.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

During header reception, a Break Delimiter error, an Inconsistent Synch Field or a Timeout error leads LINFlex to discard the header. An interrupt is generated if LINIER[HEIE] = 1. LINFlex returns to idle state.

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.

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.

Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer will be overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.
- If buffer is not released (LINSR[RMB] = 1) before reception of next Identifier and if RBLM is set then ID along with the data is discarded.

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. On header reception a slave node 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 LINFlex controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources that would otherwise be needed by software for filtering.

Filter mode

Usually each of the sixteen IFCR registers filters one dedicated identifier, but this limits the number of identifiers LINFlex can handle to the number of IFCR registers implemented in the device. Therefore, in order to be able to handle more identifiers, the filters can be configured in mask mode.

In **identifier list mode** (the default mode), both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register.

In **mask mode**, the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”. For the bit mapping and registers organization, please see *Figure 258*.

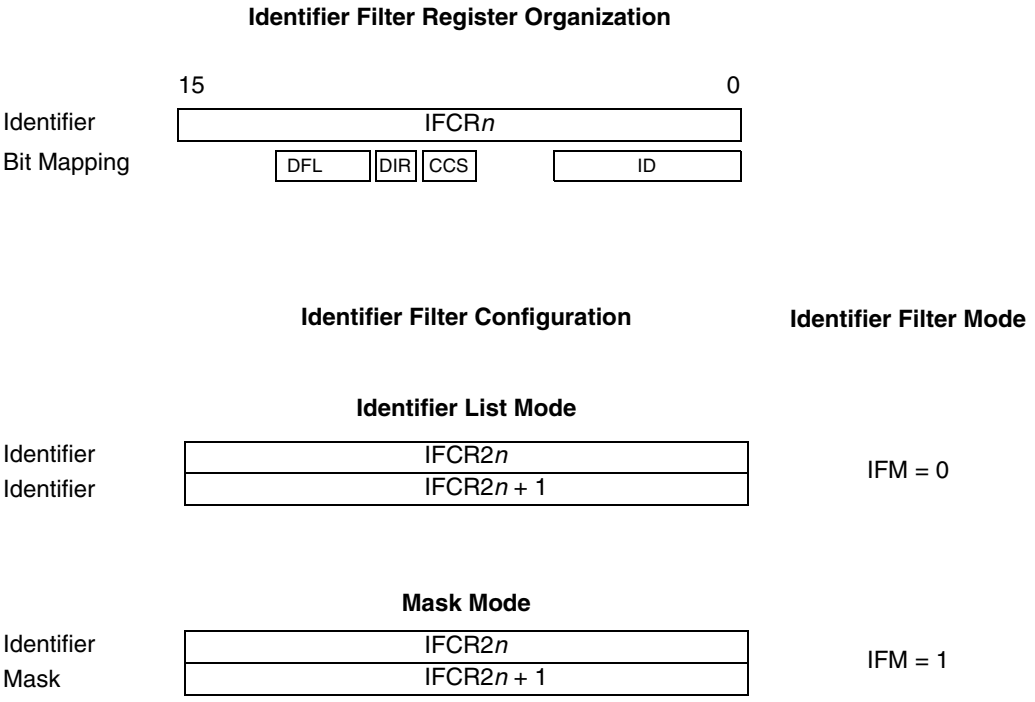


Figure 258. Filter configuration—register organization

Identifier filter mode configuration

The identifier filters are configured in the IFCR*x* registers. To configure an identifier filter the filter must first be activated by programming IFER[FACT] = 1. The **identifier list** or **identifier mask** mode for the corresponding IFCR*x* registers is configured by the IFMR[IFM] bit. For each filter, the IFCR*x* register configures the ID (or the mask), the direction (TX or RX), the data field length, and the checksum type.

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Table 261. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
0	0	0	RX interrupt on all identifiers

Table 261. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
a (a > 0)	a	0	— TX interrupt on identifiers matching the filters, — RX interrupt on all other identifiers if BF bit is set, no RX interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	— TX interrupt on identifiers matching the TX filters, — RX interrupt on identifiers matching the RX filters, — all other identifiers discarded (no interrupt)
b (b > 0)	0	b	— RX interrupt on identifiers matching the filters, — TX interrupt on all other identifiers if BF bit is set, no TX interrupt if BF bit is reset

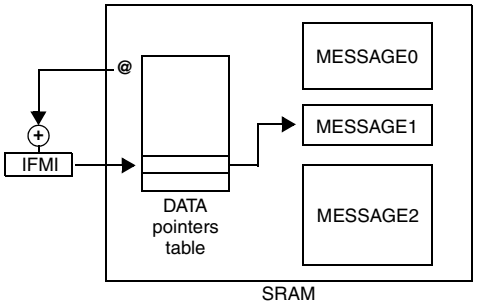


Figure 259. Identifier match index

Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if $f_{periph\_set\_1\_clk}$ tolerance is greater than 1.5%. This feature compensates a $f_{periph\_set\_1\_clk}$ deviation up to 14%, as specified in LIN standard.

This mode is similar to Slave mode as described in *Section , Slave mode*, with the addition of automatic resynchronization enabled by the LASE bit. In this mode LINFlex adjusts the fractional baud rate generator after each Synch Field reception.

Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN Break, the time duration between five falling edges on RDI is sampled on $f_{periph\_set\_1\_clk}$ and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (see *Figure 260*). Then the LFDIV value (and its associated registers LINIBRR and LINFBRR) is automatically updated at the end of the fifth falling edge. During LIN Synch Field

measurement, the LINFlex state machine is stopped and no data is transferred to the data register.

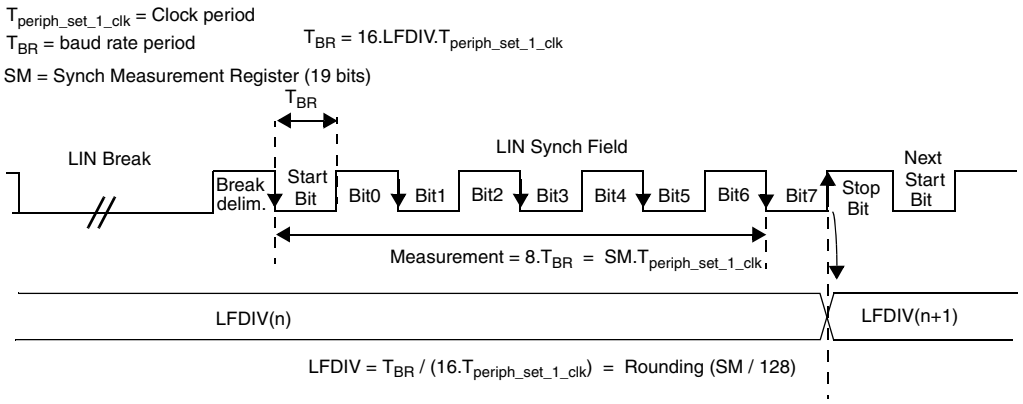


Figure 260. LIN synch field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 12 bits in the LINIBRR and the fraction is coded on 4 bits in the LINFBRR.

If LASE bit = 1 then LFDIV is automatically updated at the end of each LIN Synch Field.

Three internal registers (not user-accessible) manage the auto-update of the LINFlex divider (LFDIV):

- LFDIV_NOM (nominal value written by software at LINIBRR and LINFBRR addresses)
- LFDIV_MEAS (results of the Field Synch 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.

Deviation error on the Synch Field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Synch 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 Synch 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 LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

The second check is based on a measurement of time between each falling edge of the Synch 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 LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

Note that the LINFlex 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 synch byte.

Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In LIN mode, the LINFlex controller acknowledges a clock gating request once the frame transmission or reception is completed.

21.8.3 8-bit timeout counter

LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOCR becomes read-only, and OC1 and OC2 output compare values in the LINOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the LINCR1[MME] bit), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

LIN Master mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO = 28-bit time.

Field OC1 checks T_{Header} and $T_{Response}$ and field OC2 checks T_{Frame} (see *Figure 261*).

When LINFlex moves from Break delimiter state to Synch Field state (see *Section , LIN status register (LINSR)*):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + 28),
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + 28 + RTO × 9 (frame timeout value for an 8-byte frame),
- the TOCE bit is set.

On the start bit of the first response data byte (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)).

On the first response byte is received, 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 response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response (DFL = 7) is always assumed.

LIN Slave mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks T_{Header} and $T_{Response}$ and OC2 checks T_{Frame} (see *Figure 261*).

When LINFlex moves from Break state to Break Delimiter state (see *Section , LIN status register (LINSR)*):

- 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 response data byte (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)).

Once the first response byte is received, 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 reset.

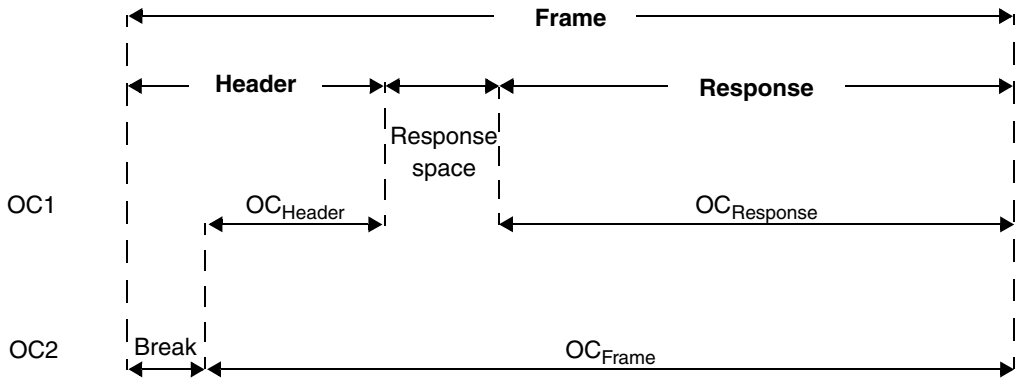


Figure 261. Header and response timeout

Output compare mode

Programming LINTCSR[LTOM] = 1 enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.

21.8.4 Interrupts

Table 262. LINFlex 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

Table 262. LINFlex interrupt control (continued)

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Wake-up 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 TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes

22 FlexCAN

22.1 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in *Figure 262*, which describes the main subblocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. Support for 32 MBs is provided. The functions of the submodules are described in subsequent sections.

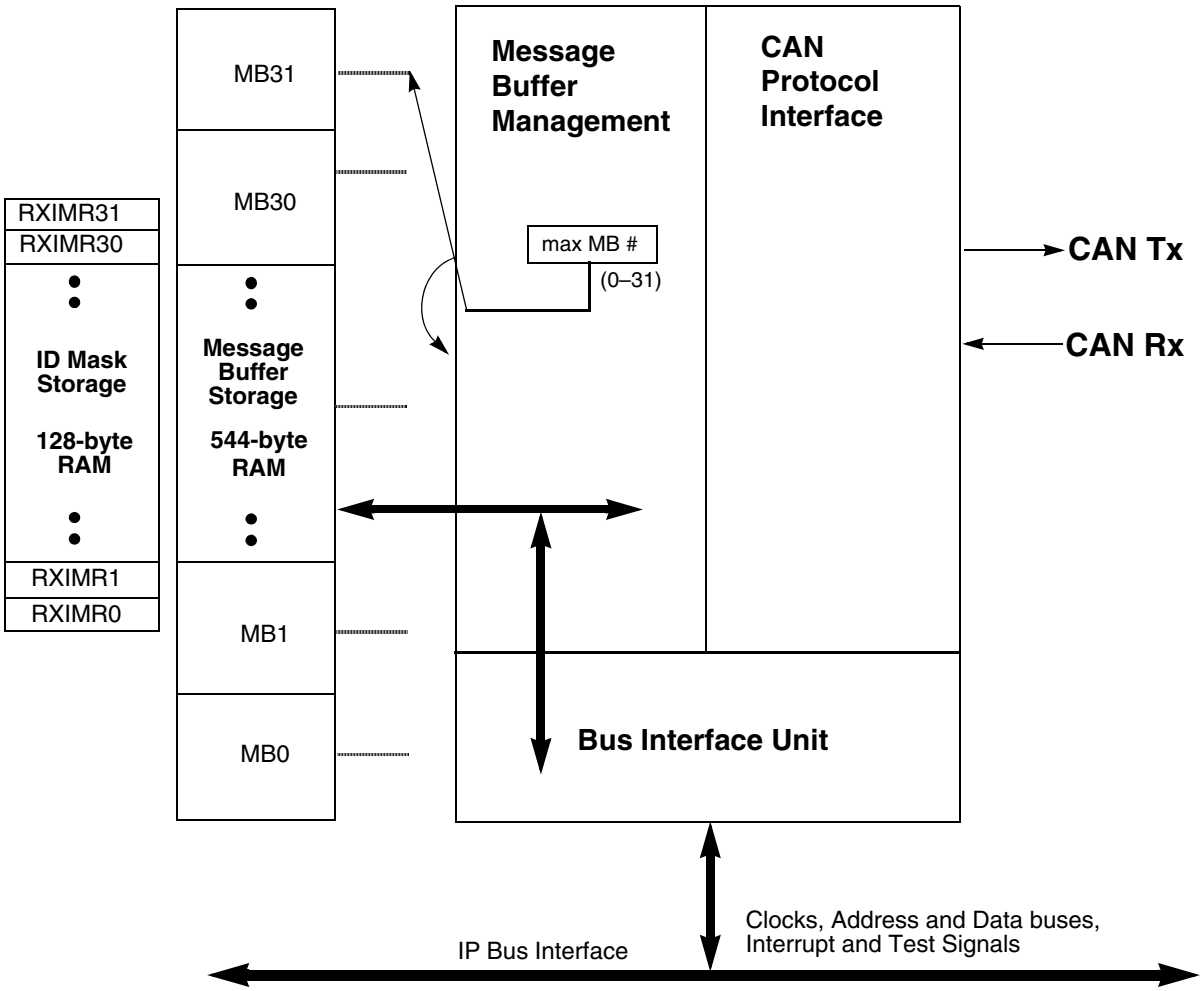


Figure 262. FlexCAN block diagram

22.1.1 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. 32 Message Buffers are supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule 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) submodule handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule 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 Bus Interface Unit.

One FlexCAN module is implemented on the SPC560P40/34 device.

22.1.2 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 as high as 1 Mbit/s
  - Content-related addressing
- 32 Flexible Message Buffers (MBs) of 0 to 8 bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes 544 bytes (32 MBs) of RAM used for MB storage
- Includes 128 bytes (32 MBs) 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 version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen-only mode capability
- Programmable loop-back 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, with programmable wake up

22.1.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only Mode, and Loop-Back mode. There are also these low power modes: Disable mode and Stop mode, and a Test 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

  It is enabled when the FRZ bit in the Module Configuration Register (MCR) is asserted. If enabled, Freeze Mode is entered when the HALT bit in the 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 , "Freeze mode* for more information.

- Listen-Only mode

  The module enters this mode when the LOM bit in the Control Register is asserted. 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 will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

- Loop-Back mode

  The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN 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 is asserted. When disabled, the module shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. Exit from this mode is done by negating the MDIS bit in the MCR. See *Section , "Module disable mode* for more information.

- Stop mode

  This low power mode is entered when Stop mode is requested at device 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. See *Section , "Stop mode* for more information.

22.2 External signal description

22.2.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 263* and described in more detail in the next subsections.

Table 263. FlexCAN signals

Signal name	Direction	Description
RXD	Input	CAN receive pin
TXD	Output	CAN transmit pin

22.2.2 Signal Descriptions

RXD

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

TXD

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

22.3 Memory map and registers 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 device. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN 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.

22.3.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 32 MBs capability is shown in *Table 264*. The access type can be Supervisor (S), Test (T) or Unrestricted (U), also called User access. Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR.

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 BCC bit in the MCR is asserted.

The address ranges 0x0060–0x027F and 0x0880–0x08FF are occupied by two separate embedded memories of RAM, of 544 bytes and 128 bytes, respectively. When the FlexCAN is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the

BCC bit in the MCR is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the address range 0x0880–0x097F is considered reserved space, independent of the value of the BCC bit.

Table 264. FlexCAN module memory map

Offset from FlexCAN_BASE 0xFFFC_0000	Register	Location
0x0000	Module Configuration Register (MCR)	*on page 22-542*
0x0004	Control Register (CTRL)	*on page 22-546*
0x0008	Free Running Timer (TIMER)	*on page 22-549*
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	*on page 22-550*
0x0014	Rx Buffer 14 Mask (RX14MASK)	*on page 22-551*
0x0018	Rx Buffer 15 Mask (RX15MASK)	*on page 22-551*
0x001C	Error Counter Register (ECR)	*on page 22-552*
0x0020	Error and Status Register (ESR)	*on page 22-553*
0x0024	Reserved	
0x0028	Interrupt Masks 1 (IMASK1)	*on page 22-556*
0x002C	Reserved	
0x0030	Interrupt Flags 1 (IFLAG1)	*on page 22-557*
0x0034–0x005F	Reserved	
0x0060–0x007F	Serial Message Buffers (SMB0–SMB1) – Reserved	—
0x0080–0x017F	Message Buffers MB0–MB15	*on page 22-536*
0x0180–0x027F	Message Buffers MB16–MB31	*on page 22-536*
0x0280–0x087F	Reserved	
0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	*on page 22-558*
0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	*on page 22-558*
0x0900–0x3FFF	Reserved	

Table 265. FlexCAN register reset status

Register	Affected by hard reset	Affected by soft reset
Module Configuration Register (MCR)	Yes	Yes
Control Register (CTRL)	Yes	No
Free Running Timer (TIMER)	Yes	Yes

Table 265. FlexCAN register reset status (continued)

Register	Affected by hard reset	Affected by soft reset
Reserved		
Rx Global Mask (RXGMASK)	Yes	No
Rx Buffer 14 Mask (RX14MASK)	Yes	No
Rx Buffer 15 Mask (RX15MASK)	Yes	No
Error Counter Register (ECR)	Yes	Yes
Error and Status Register (ESR)	Yes	Yes
Interrupt Masks 1 (IMASK1)	Yes	Yes
Interrupt Flags 1 (IFLAG1)	Yes	Yes
Serial Message Buffers (SMB0–SMB1) – Reserved	No	No
Message Buffers MB0–MB15	No	No
Message Buffers MB16–MB31	No	No
Rx Individual Mask Registers RXIMR0–RXIMR15	No	No
Rx Individual Mask Registers RXIMR16–RXIMR31	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 266*. The module also implements two additional Message Buffers called SMB0 and SMB1 (Serial Message Buffers), in the address ranges 0x60–0x6F and 0x70–0x7F, which are not accessible to the end user. They are used for temporary storage of frames during the reception and transmission processes. *Table 266* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 266. Message Buffer MB0 memory mapping

Address offset	MB field
0x0080	Control and Status (C/S)
0x0084	Identifier Field
0x0088–0x008F	Data Field 0 – Data Field 7 (1 byte each)

22.3.2 Message buffer structure

The Message Buffer structure used by the FlexCAN module is represented in *Table 263*. 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 263. Message buffer structure

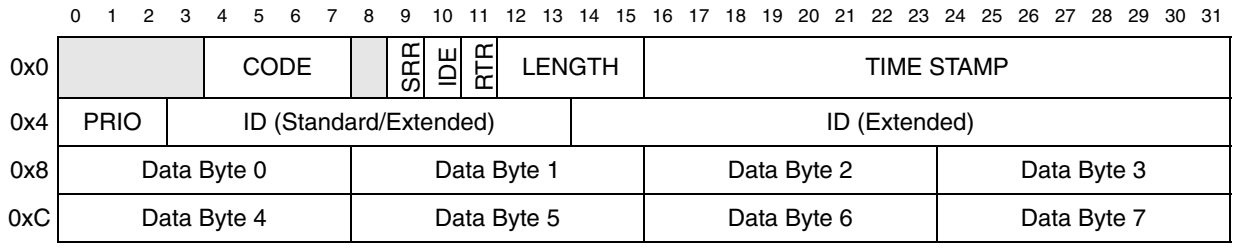


Table 267. Message Buffer structure field description

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 268* and *Table 269*. See *Section 22.4, "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 will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN 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
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If FlexCAN 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 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 Frame 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 *Table 263*). 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.

Table 267. Message Buffer structure field description (continued)

Field	Description
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 22.4.3, "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 268. 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.
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 22.4.5, "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 will be overwritten again, and the code will remain OVERRUN. Refer to *Section 22.4.5, "Matching process* for details about overrun behavior.

Table 268. Message buffer code for Rx buffers (continued)

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
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. Note that for Tx MBs (see *Table 269*), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR.

Table 269. 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 AEN bit in MCR is asserted. 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 restart the process again.
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 will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

Table 270. 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

Table 270. MB0–MB31 addresses (continued)

Address	Register	Address	Register
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

22.3.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFF (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Figure 264* shows the Rx FIFO data structure. The region 0x00–0x0C contains an MB structure that is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x10–0xDF is reserved for internal use of the FIFO engine. The region 0xE0–0xFF contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. *Table 271* 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 22.4.7, "Rx FIFO* for more information.

Figure 264. Rx FIFO structure

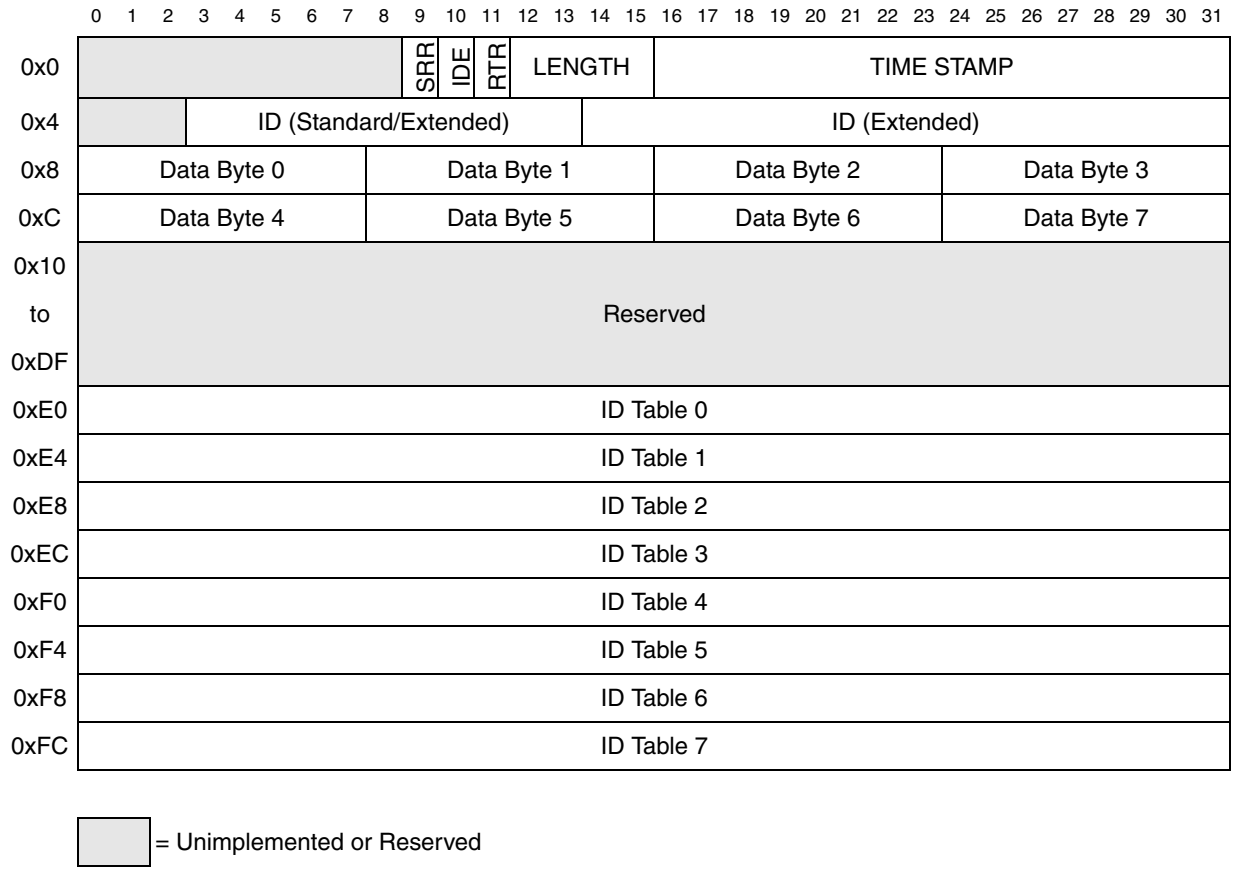


Table 271. ID Table 0 - 7

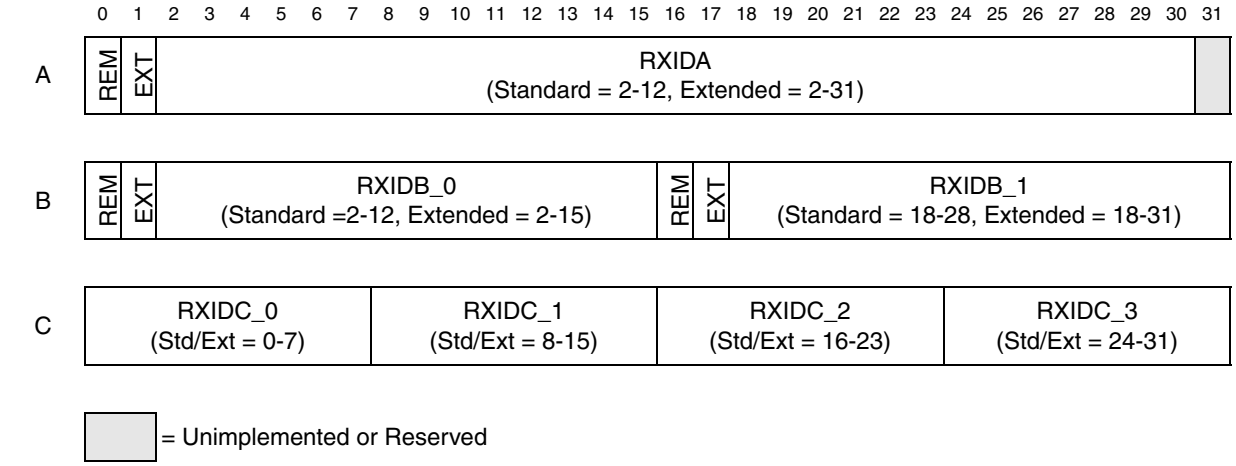


Table 272. Rx FIFO Structure field description

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 (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0 RXIDB_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) (3 to 13) 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 RXIDC_1 RXIDC_2 RXIDC_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.

22.3.4 Registers description

The FlexCAN registers are described in this section in ascending address order.

Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Figure 265. Module Configuration Register (MCR)

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	0	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

Table 273. MCR field descriptions

Field	Description
0 MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in MCR not affected by soft reset. See *Section , "Module disable mode* for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
1 FRZ	Freeze Enable The FRZ 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 asserted, FlexCAN is able to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 0 Not able to enter Freeze Mode. 1 Able to enter Freeze Mode.
2 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 22.3.3, "Rx FIFO structure* and *Section 22.4.7, "Rx FIFO* for more information. 0 FIFO disabled. 1 FIFO enabled.
3 HALT	Halt FlexCAN Assertion of 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 FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See *Section , "Freeze mode* for more information. 0 No Freeze Mode request. 1 Enters Freeze Mode if the FRZ bit is asserted.
4 NOT_RDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode. 1 FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode.

Table 273. MCR field descriptions (continued)

Field	Description
5 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.
6 SOFT_RST	Soft Reset When this bit is asserted, FlexCAN 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 asserted directly by the CPU when it writes to the MCR, but it is also asserted 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 asserted while reset is pending, and is automatically negated 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 264*.
7 FRZ_ACK	Freeze Mode Acknowledge This read-only bit indicates that FlexCAN 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 FlexCAN has actually entered Freeze Mode. If Freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See *Section , “Freeze mode* for more information. 0 FlexCAN not in Freeze mode, prescaler running. 1 FlexCAN in Freeze mode, prescaler stopped.
8 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 264*. 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.

Table 273. MCR field descriptions (continued)

Field	Description
10 WRN_EN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be 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 transition from <96 to ≥ 96.
11 LPM_ACK	Low Power Mode Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See *Section , "Module disable mode* and *Section , "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.
14 SRX_DIS	Self Reception Disable This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be 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 will be generated due to the frame reception. 0 Self reception enabled 1 Self reception disabled
15 BCC	Backwards Compatibility Configuration This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: – For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN 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, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to '0110' (overrun). Upon reset this bit is negated, 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.
18 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
19 AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 0 Abort disabled 1 Abort enabled

Table 273. MCR field descriptions (continued)

Field	Description
22–23 IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 274*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 22.3.3, "Rx FIFO structure.*
26–31 MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will 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 FlexCAN will not transmit or receive frames.

Table 274. IDAM coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back 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. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Figure 266. Control Register (CTRL)

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

Table 275. CTRL field descriptions

Field	Description
0–7 PRESDIV	Prescaler Division Factor This 8-bit 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 , "Protocol timing.* **Sclock frequency = CPI clock frequency / (PRESDIV + 1).**
8–9 RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. **Resync Jump Width = RJW + 1.**
10–12 PSEG1	PSEG1 — Phase Segment 1 This 3-bit 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.**
13–15 PSEG2	PSEG2 — Phase Segment 2 This 3-bit 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.**
16 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.
17 ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled. 1 Error interrupt enabled.
18 CLK_SRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator 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 , "Protocol timing* for more information. 0 The CAN engine clock source is the oscillator clock. 1 The CAN engine clock source is the bus clock. Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.

Table 275. CTRL field descriptions (continued)

Field	Description
19 TWRN_MSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Tx Warning Interrupt disabled. 1 Tx Warning Interrupt enabled.
20 RWRN_MSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Rx Warning Interrupt disabled. 1 Rx Warning Interrupt enabled.
21 LPB	Loop Back This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN 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 Loop Back disabled. 1 Loop Back enabled.
24 SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx 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.
25 BOFF_REC	Bus Off Recovery Mode This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated 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 asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting 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.
26 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 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

Table 275. CTRL field descriptions (continued)

Field	Description
27 LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for Message Buffer transmission. When asserted, 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.
28 LOM	Listen-Only Mode This bit configures FlexCAN 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 will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 Listen Only Mode is deactivated. 1 FlexCAN module operates in Listen Only Mode.
29–31 PROPSEG	Propagation Segment This 3-bit 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.

1. One time quantum is equal to the Sclock period.

Free Running Timer (TIMER)

This 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 will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Figure 267. Free Running Timer (TIMER)

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

Table 276. TIMER field descriptions

Field	Description
TIMER	Holds the value for this timer.

Rx Global Mask register (RXGMASK)

This 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 MB, setting the BCC bit in the MCR causes the RXGMASK register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as an acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in the 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.

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.

Figure 268. Rx Global Mask register (RXGMASK)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
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	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 277. RXGMASK field description

Field	Description
0–31 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.

Rx 14 Mask (RX14MASK)

This 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 MB, setting the BCC bit in the MCR causes the RX14MASK register to have no effect on the module operation.

RX14MASK is used as an acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in the MCR is set (FIFO enabled), the RX14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Figure 269. Rx Buffer 14 Mask register (RX14MASK)

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

Table 278. RX14MASK field description

Field	Description
0–31 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.

Rx 15 Mask (RX15MASK)

This 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 MB, setting the BCC bit in the MCR causes the RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in the MCR is set (FIFO enabled), the RX15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Figure 270. Rx Buffer 15 Mask register (RX15MASK)

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

Table 279. RX15MASK field description

Field	Description
0–31 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.

Error Counter Register (ECR)

This register has two 8-bit fields that reflect the value of two FlexCAN error counters:

- Transmit Error Counter (Tx_Err_Counter field)
- 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 Test Mode or Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register 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 will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g., transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any

influence on the bus when in ‘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 FLT_CONF field in the Error and Status Register 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 FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to 0.
- If FlexCAN 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 0 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 FLT_CONF field in the Error and Status Register 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 ACK_ERR bit in the Error and Status Register). 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 to ‘Error Active’ state.

Figure 271. Error Counter Register (ECR)

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

Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four 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 16–21. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, and ERR_INT, that are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect). See *Section 22.4.10, "Interrupts* for more details.

Figure 272. Error and Status Register (ESR)

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																
Reset	0	0	0	0	0	0	0	0	0	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 280. Error and Status Register (ESR) field description

Field	Description
14 TWRN_INT	Tx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control 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.
15 RWRN_INT	Rx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, 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 Control 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.
16 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.
17 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.

Table 280. Error and Status Register (ESR) field description (continued)

Field	Description
18 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
19 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.
20 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.
21 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.
22 TX_WRN	TX Error Counter This bit indicates when repetitive errors are occurring during message transmission. 0 No such occurrence. 1 TX_Err_Counter ≥ 96.
23 RX_WRN	Rx Error Counter This bit indicates when repetitive errors are occurring during message reception. 0 No such occurrence. 1 Rx_Err_Counter ≥ 96.
24 IDLE	CAN bus IDLE state This bit indicates when the CAN bus is in IDLE state. 0 No such occurrence. 1 CAN bus is now IDLE.
25 TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE = 0). 1 FlexCAN is transmitting a message (IDLE = 0).
26–27 FLT_CONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 281*. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.

Table 280. Error and Status Register (ESR) field description (continued)

Field	Description
29 BOFF_INT	Bus Off Interrupt This bit is set when FlexCAN enters 'Bus Off' state. If the corresponding mask bit in the Control 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.
30 ERR_INT	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 Control 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 Error and Status Register.
31	Wake-Up Interrupt When FlexCAN 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.

Table 281. Fault confinement state

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number 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 (that is, when the corresponding IFLAG1 bit is set).

Figure 273. Interrupt Masks 1 Register (IMASK1)

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

Table 282. IMASK1 field descriptions

Field	Description
0–31 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. 1The 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.

Interrupt Flags 1 Register (IFLAG1)

This 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 will be generated. The Interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the MCR is set (FIFO enabled), the function of the eight 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.

Figure 274. Interrupt Flags 1 Register (IFLAG1)

Address: Base + 0x0030 Access: User read/write

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

Table 283. IFLAG1 field descriptions

Field	Description
0–23 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 MB has successfully completed transmission or reception.
24 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.

Table 283. IFLAG1 field descriptions (continued)

Field	Description
25 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.
26 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.
27–31 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.

Rx Individual Mask Registers (RXIMR0–RXIMR31)

These registers are used as acceptance masks for ID filtering in Rx MBs 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 (FEN bit in MCR 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 MBs, 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 will return “all zeros”. Furthermore, if the BCC bit in the MCR is negated, any read or write operation to these registers results in access error.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK and RX15MASK registers are available, regardless of the value of the BCC bit.

Figure 275. Rx Individual Mask Registers (RXIMR0–RXIMR31)

Address: See *Table 285* 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

Table 284. RXIMR0–RXIMR31 field descriptions

Field	Description
0–31 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.

Table 285. 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
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

22.4 Functional description

22.4.1 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 by a set of as many as 32 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 22.3.2, "Message buffer structure*). The memory corresponding to the first eight 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 (as many as 8 extended IDs or 16 standard IDs or 32 eight-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 MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB 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 MB with a 0000 code is inactive (refer to *Table 268*). Similarly, a Tx MB with a 1000 or 1001 code is also inactive (refer to *Table 269*). An MB not programmed with 0000, 1000, or 1001 will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see *Section , "Message Buffer deactivation*).

22.4.2 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 MB 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 , "Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write '1000' to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see *Section , "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 MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer 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 MB in step 4 (see *Table 268* and *Table 269* in *Section 22.3.2, "Message buffer structure*). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to

update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

22.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[a] or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the Control 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 MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB 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 MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, 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 MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in the MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN 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. FlexCAN transmits as many as 8 data bytes, even if the DLC (Data Length Code) value is bigger.

22.4.4 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

1. If the MB 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

a. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

(see *Section , "Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write '1000' to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section , "Message Buffer deactivation*). If the MB already programmed as a receiver, just write '0000' to the Code field of the Control and Status word to keep the MB inactive.

2. Write the ID word
3. Write '0100' to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see *Table 268* and *Table 269* in *Section 22.3.2, "Message buffer structure*)
- 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 MB 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 Free Running Timer (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 MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 22.4.6, "Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 268*. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: **never perform polling by directly reading the C/S word of the MBs. Instead, read the IFLAG registers**.

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be 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 22.4.7, "Rx FIFO*). Upon receiving the frames

available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

22.4.5 Matching process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs 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 MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB 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 MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB 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 MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see *Section , “Message Buffer lock mechanism*)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to *Table 268* and *Table 269*). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see *Section , “Message Buffer lock mechanism*).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB 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 MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs 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 BCC bit in the MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to *Section , "Rx Individual Mask Registers (RXIMR0–RXIMR31).* During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is "don't care". Please 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 asserted and while the module is in Freeze Mode.

FlexCAN 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 BCC bit in the MCR is negated.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.

22.4.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Transmit process and *Section 22.4.4, "Receive process.* Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

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 asserting the AEN bit in the MCR.

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 MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB 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. In 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 MB 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 MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

1. CPU writes 1001 into the code field of the C/S word.
2. CPU reads the CODE field and compares it to the value that was written.
3. 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 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 MB was aborted (CODE = 1001) or it was transmitted (CODE = 1000).

Message Buffer deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB 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 , “Transmission abort mechanism* should be used.

Message Buffer lock mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

Note: The locking mechanism only applies to Rx MBs that have a code different than INACTIVE (‘0000’) or EMPTY[b] (‘0100’). Also, Tx MBs can not be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. 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 MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB 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 and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

22.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. 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 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see *Section 22.3.3, “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 as many as six 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 an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80

b. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. In current FlexCAN versions, this behavior is maintained when the BCC bit is negated.

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 4frames 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 22.3.3, "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 8 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 RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR 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.

22.4.8 CAN protocol related features

Remote frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to 1. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, 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 MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit 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 an MB, this message buffer immediately enters the internal arbitration process, but is considered as 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), FlexCAN will 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 will be 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).

Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on 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

Time stamp

The value of the Free Running Timer 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 Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in *Section , “Control Register (CTRL).*

Protocol timing

Figure 276 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 output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

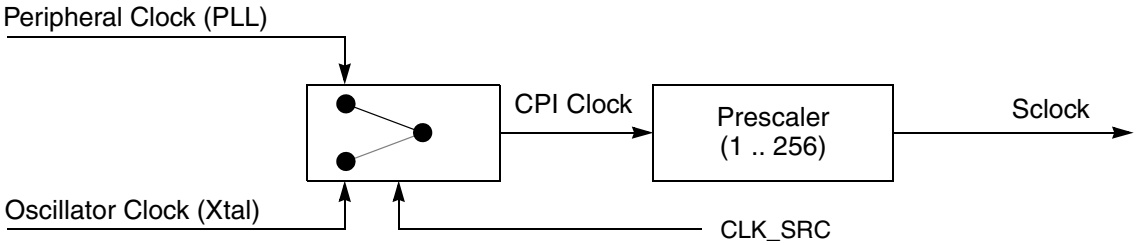


Figure 276. CAN engine clocking scheme

The crystal oscillator clock should be selected whenever a tight tolerance (to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See *Section , “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.

Equation 32

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ Value)}$$

A bit time is subdivided into three segments[c] (reference *Figure 277* and *Table 286*):

- 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.

Equation 33

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

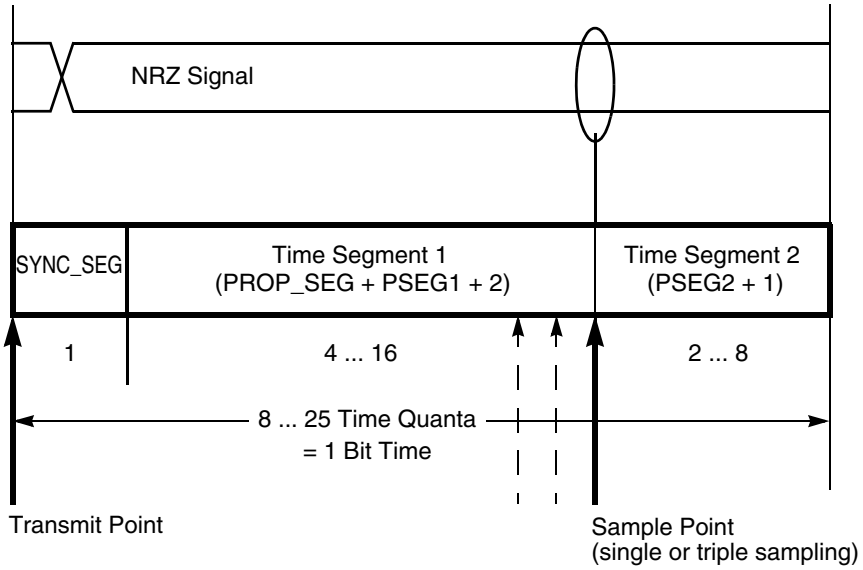


Figure 277. Segments within the bit time

c. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 286. 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.

Table 287 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 287. CAN standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Resynchronization 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.

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 278*.

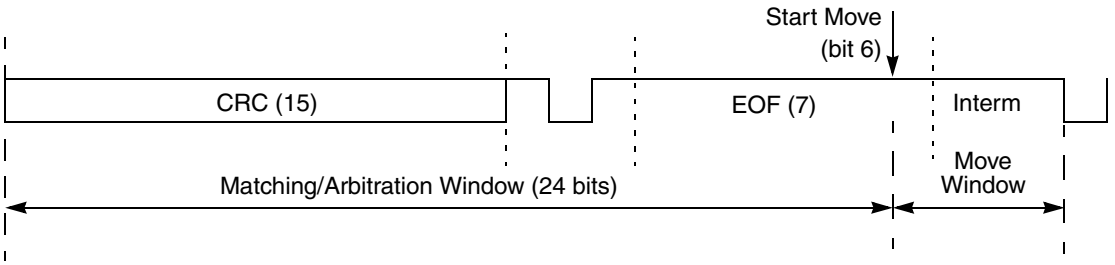


Figure 278. Arbitration, match, and move time windows

When doing matching and arbitration, FlexCAN 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:

- A valid CAN bit timing must be programmed, as indicated in *Table 287*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, that is, the PLL can not 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 288*

Table 288. 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 288* 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, PSEG2). As an example, taking the case of 32 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to 1 and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

22.4.9 Modes of operation details

Freeze mode

This mode is entered by asserting the HALT bit in the MCR or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR and the module is not in any of the low power modes. When Freeze Mode is requested during transmission or reception, FlexCAN 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 Rx input pin and drives the Tx 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 FRZ_ACK bit to be asserted in the MCR before executing any other action, otherwise FlexCAN 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 negates the FRZ bit in the MCR
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

Module disable mode

This low power mode is entered when the MDIS bit in the MCR is asserted. 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 negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN 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 Rx input pin and drives its Tx 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 Free Running Timer, the Error Counter Register, and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

Stop mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the LPM_ACK bit, negates 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, FlexCAN 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 Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Stop Mode is exited by the CPU resuming the clocks and removing the Stop Mode request.

22.4.10 Interrupts

The module can generate as many as 38 interrupt sources (32 interrupts due to message buffers and 6 interrupts due to ORed interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning, and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

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 assigned a flag bit 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 (bit FEN on MCR 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 , "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 MB caused the interrupt.

The other five interrupt sources (Bus Off, Error, Tx Warning, Rx Warning, and Wakeup) 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.

22.4.11 Bus interface

The CPU access to FlexCAN registers is subject to the following rules:

- All reads and writes to test registers must be qualified with ips_test_access signal. Read and write access to test mode registers in non test mode results in access error.
- 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 unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in the MCR is negated 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 FlexCAN is configured with 32 MBs and MAXMB is programmed with 0. The maximum number of MBs in this case becomes 1. 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 MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

22.5 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

22.5.1 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 264* to see which 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 asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not 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 negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is unsynchronized to the CAN bus, the HALT and FRZ bits in the MCR are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR are set. The Tx pin is in recessive state and FlexCAN 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 FlexCAN is put into Freeze Mode (see *Section , "Freeze mode*). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts) and in MCR for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

23 Analog-to-Digital Converter (ADC)

23.1 Overview

23.1.1 Device-specific features

- 1 ADC unit
- 10-bit resolution
- 16 input channels
  - 15 channels on 100-pin LQFP; 11 channels on 64-pin LQFP
  - Channel 15 dedicated to the internal 1.2 V rail
- Conversion time < 1 µs including sampling time at full precision (conversion time target of 700 ns for the analog section)
- Cross triggering unit (CTU)
- 4 analog watchdogs with interrupt capability for continuous hardware monitoring of as many as 4 analog input channels
- Clock stretching (with CTU pulse)
- Sampling and conversion time register CTR0 (internal precision channels)
- Left-aligned result format
- Right-aligned result format
- One Shot/Scan Modes
- Chain Injection Mode
- Power-down mode
- 2 different Abort functions allow aborting either single-channel conversion or chain conversion
- As many as 16 data registers for storing converted data. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- Auto-clock-off
- 2 modes of operation, each with DMA compatible interface
  - Normal Mode
  - CTU Control Mode

These features are absent on the device:

- Support of external channels
- Presampling
- Alternate analog thresholds
- Offset Cancellation and Offset Refresh Control
- External start and triggering

23.1.2 Device-specific pin configuration features

- For *Section 23.3.3, ADC sampling and conversion timing,"* f_{ck} = (1/2) MC_PLL_CLK is true where the bit ADCLKSEL would be always 0 (default value), meaning that AD_clk is half of MC_PLL_CLK. A clock prescaler (1 or 2) can be configured. The AD_clk has

the same frequency of MC_PLL_CLK or is half of MC_PLL_CLK, depending on the value of the bit ADCLKSEL.

- CTUEN field in the MCR: Enables or disables CTU control mode
- Registers CDR[16..95] not used
- ADC channel 15 is internally connected to the core voltage

23.1.3 Device-specific implementation

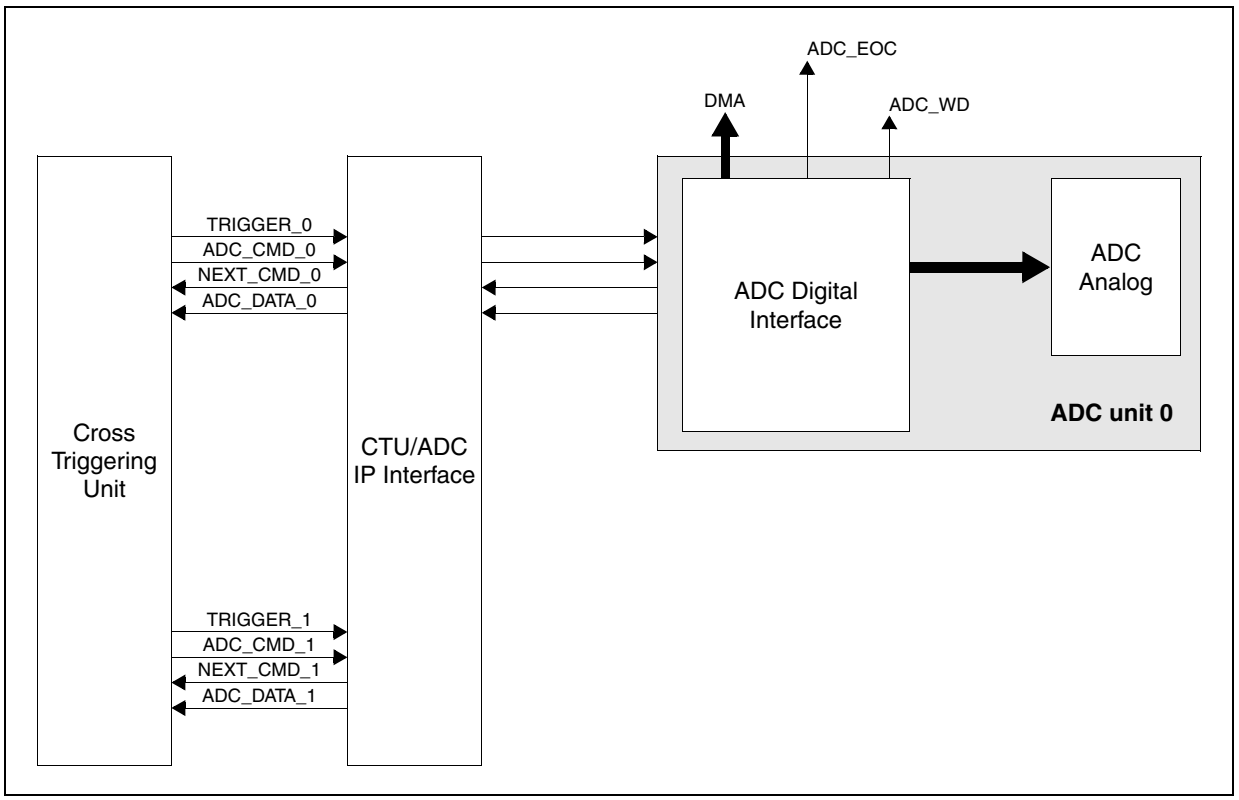


Figure 279. ADC implementation diagram

23.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications.

The ADC contains advanced features for normal or injected conversion. It provides support for eDMA (direct memory access) mode operation. A conversion can be triggered by software or hardware (Cross Triggering Unit or PIT).

The mask registers present within the ADC can be programmed to configure the channel to be converted.

Analog watchdogs allow continuous hardware monitoring.

23.3 Functional description

23.3.1 Analog channel conversion

Two conversion modes are available within the ADC:

- Normal conversion
- Injected conversion

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 field of NCMR registers. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (NSTART bit in the Main Status Register (MSR) is reset).

Start of normal conversion

By programming the configuration bits in the Main Configuration Register (MCR), the normal conversion can be started in two ways:

- By software — The conversion chain starts when the MCR[NSTART] bit is set.
- By trigger — An on-chip internal signal triggers an ADC conversion. The settings in the MCR select how conversions are triggered based on these internal signals:
  - A rising/falling edge detected in the signal sets the MSR[NSTART] bit and starts the programmed conversion.
  - The conversion is started if and only if the MCR[NSTART] bit is set and the programmed level on the trigger signal is detected.

Table 289. Configurations for starting normal conversion

Type of conversion start	NSTART (in MCR)	NSTART (in MSR)	Result
Software	1	1	Conversion chain starts
Trigger	—	1	A falling or rising edge detected in a trigger signal sets the NSTART bit in the MSR and starts the programmed conversion.
	1	1	The conversion is started if the programmed level on the trigger signal is detected: the start of conversion is enabled if the external pin is low or high.

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.

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 interrupt ECH (see interrupt controller chapter for further details) is immediately issued after the start of conversion.

Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, it is necessary to program the MCR[MODE] bit. 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 280*.

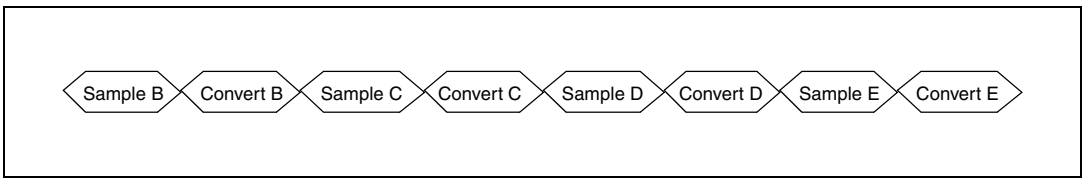


Figure 280. Normal conversion flow

In **One Shot Mode** (MODE = 0) a sequential conversion specified in the NCMR registers is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

Example 13 One Shot Mode (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. 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 NSTART status bit in the MSR 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** (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, the MCR[NSTART] bit 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 MSR[NSTART] bit.

Example 14 Scan Mode (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. 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 till the MCR[NSTART] bit is cleared 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 in the IMR register).

Injected channel conversion

A conversion chain can be injected into the ongoing normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As with normal conversion, each channel can be selected individually. This injected conversion (which can occur only in One Shot mode) interrupts the normal conversion (which can be in One Shot or Scan mode). When an injected conversion is inserted, ongoing normal 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 aborted, as shown in *Figure 281*.

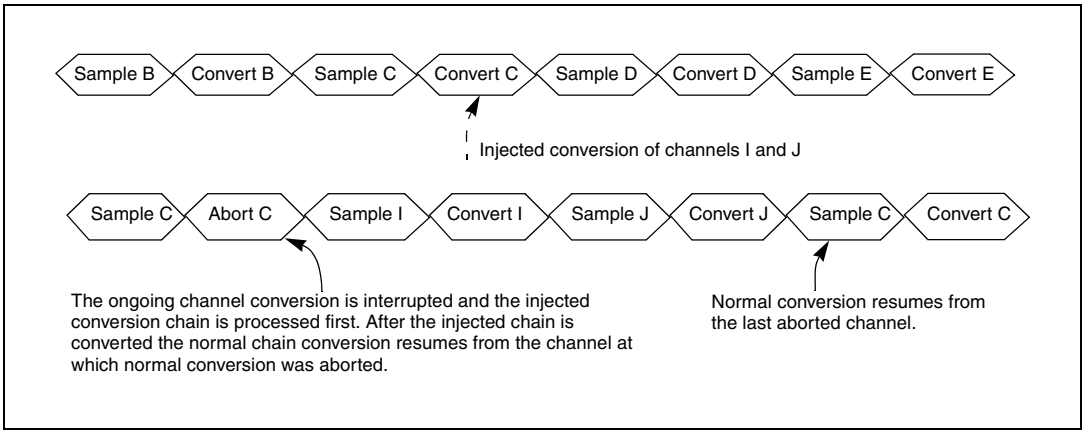


Figure 281. Injected sample/conversion sequence

The injected conversion can be started using two options:

- By software setting the MCR[JSTART]; the current conversion is suspended and the injected chain is converted. At the end of the chain, the JSTART bit in the MSR is reset and the normal chain conversion is resumed.
- By an internal trigger signal from the PIT when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from PIT or CTU starts the injected conversion by setting the MSR[JSTART]. At the end of the chain, the MSR[JSTART] is cleared and the normal conversion chain is resumed.

The MSR[JSTART] is automatically set when the Injected conversion starts. At the same time the MCR[JSTART] 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 IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the IMR[MSKJEOC]).

If the content of all the injected conversion mask registers (JCMR) is zero (that is, no channel is selected) the JECH interrupt 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 , Abort conversion*).

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. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion of the next channel starts. The EOC interrupt 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 MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the scan mode is enabled, a new chain conversion is started. The EOC interrupt 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 (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 NSTART and JSTART bits are also reset).

23.3.2 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL]. When this bit is set to ‘1’ the ADC clock has the same frequency as the MC_PLL_CLK. Otherwise, the ADC clock is half of the MC_PLL_CLK frequency. The 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 basically 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.

Depending on the position of the rising edge of the signal internal trigger signal coming from the CTU, the ADC clock could also be stretched as illustrated in *Figure 282*.

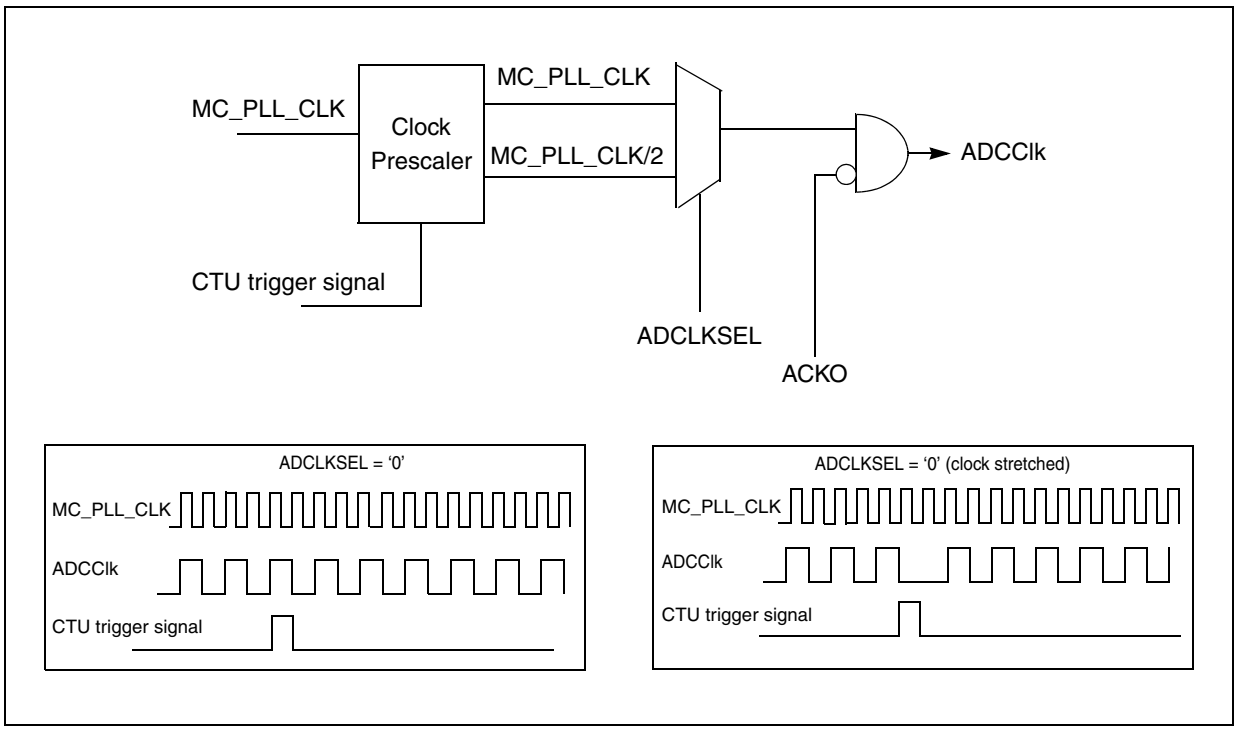


Figure 282. Prescaler simplified block diagram

The clock stretching is implemented if and only if ADCLKSEL = 0 (and clock is half of the MC_PLL_CLK).

23.3.3 ADC sampling and conversion timing

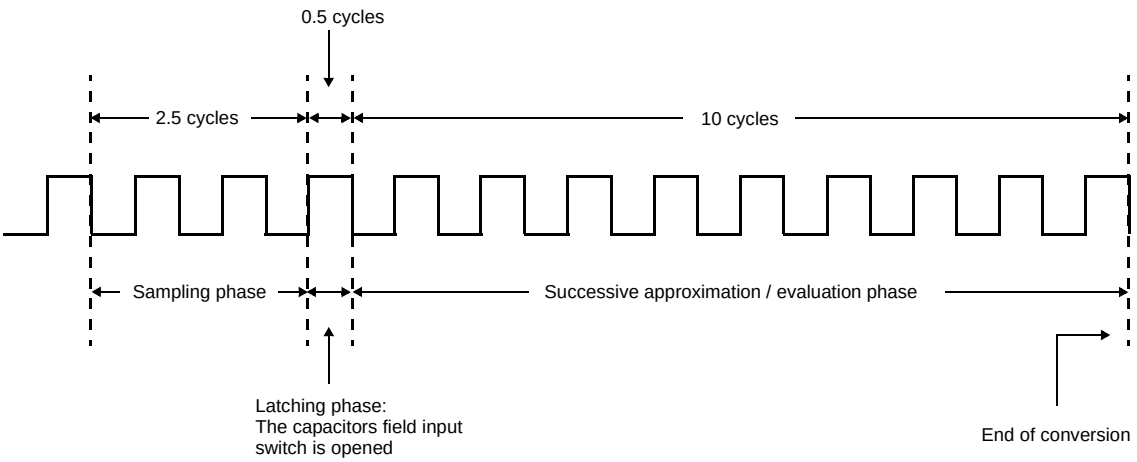
In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

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 bit fields of the CTR. Bit fields INPLATCH, INPCMP, and INPSAMP 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}).

ADC_0

Figure 283 represents the sampling and conversion sequence.



Note: Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and Fadc clk = 20 MHz

Figure 283. Sampling and conversion timings

The sampling phase duration is:

$$T_{sample} = (INPSAMP - ndelay) \bullet T_{ck}$$

$$INPSAMP \geq 3$$

where ndelay is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 3 (hardware requirement).

The total evaluation phase duration is:

$$T_{eval} = 10 \bullet T_{biteval} = 10 \bullet (INPCMP \bullet T_{ck})$$

$$(INPCMP \geq 1) \quad \text{and} \quad (INPLATCH < INPCMP)$$

INPCMP must be greater than or equal to 1 and INPLATCH must be less than INCMP (hardware requirements).

The total conversion duration is (not including external multiplexing):

$$T_{conv} = T_{sample} + T_{eval} + (ndelay \bullet T_{ck})$$

The timings refer to the unit T_{ck}, where f_{ck} = (1/2 x ADC peripheral set clock).

Table 290. ADC sampling and conversion timing at 5 V / 3.3 V for ADC0

Clock (MHz)	T_{ck} (µs)	INPSAMPLE(1)	Ndelay (2)	T_{sample} (3)	T_{sample}/T_{ck}	INPCMP	T_{eval} (µs)	INPLATCH	T_{conv} (µs)	T_{conv}/T_{ck}
6	0.167	4	0.5	0.583	3.500	1	1.667	0	2.333	14.000
7	0.143	4	0.5	0.500	3.500	1	1.429	0	2.000	14.000
8	0.125	5	0.5	0.563	4.500	1	1.250	0	1.875	15.000

Table 290. ADC sampling and conversion timing at 5 V / 3.3 V for ADC0 (continued)

Clock (MHz)	T_{ck} (µs)	INPSAMPLE(1)	Ndelay (2)	T_{sample} (3)	T_{sample}/T_{ck}	INPCMP	T_{eval} (µs)	INPLATCH	T_{conv} (µs)	T_{conv}/T_{ck}
16	0.063	9	1	0.500	8.000	1	0.625	0	1.188	19.000
32	0.031	17	1	0.500	16.000	2	0.625	1	1.156	37.000

1. Where: INPSAMPLE ≥ 3
2. Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1
3. Where: $T_{sample} = (INPSAMP-N)T_{ck}$; Must be ≥ 500 ns

Table 291. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V for ADC0

INPCMP	INPLATCH	Max f_{ADC_clk}	Min f_{ADC_clk}
00/01	0	20+4%	6
	1	—	—
10	0	—	—
	1	32+4%	6
11	0	—	—
	1	32+4%	9

23.3.4 ADC CTU (Cross Triggering Unit)

Overview

The ADC cross triggering unit (CTU) is added to enhance the injected conversion capability of the ADC. The CTU contains multiple event inputs that can be used to select the channels to be converted from the appropriate event configuration register. The CTU generates a trigger output pulse of one clock cycle and outputs onto an internal data bus the channel to be converted. A single channel is converted for each request. After performing the conversion, the ADC returns the result on internal bus.

The conversion result is also saved in the corresponding data register and it is compared with watchdog thresholds if requested.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the MC_PLL_CLK in both cases.

CTU in control mode

In CTU control mode, the CPU is able to write in the ADC registers but it cannot start any conversion. Conversion requests can be generated only by the CTU trigger pulse. If a normal or injected conversion is requested, it is automatically discarded.

When a CTU trigger pulse is received with the injected channel number, the conversion starts. The CTUSTART bit is set automatically at this point and it is also automatically reset when CTU Control mode is disabled (CTUEN = ‘0’).

23.3.5 Programmable analog watchdog

Introduction

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in *Figure 284*) specified by an upper and a lower threshold value named THRH and THRL respectively.

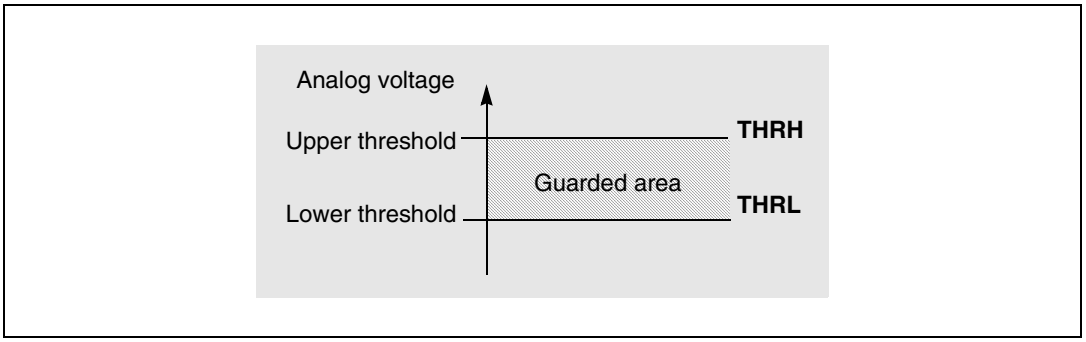


Figure 284. Guarded area

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 WTISR[WDGxH] and WTISR[WDGxL] as explained in *Table 292*. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 292. Values of WDGxH and WDGxL fields

WDGxH	WDGxL	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRL <= converted data <= THRH

The TRC[THRCH] field specifies the channel on which the analog watchdog is applied. The analog watchdog is enabled by setting the corresponding TRC[THREN].

The lower and higher threshold values for the analog watchdog are programmed using the THRHLR registers.

For example, if channel number 3 is to be monitored with threshold values in THRHLR1, then the TRC[THRCH] field is programmed to select channel number 3.

A set of threshold registers (THRHLRx and TRCx) can be linked only to a single channel for a particular THRCH value. If another channel is to be monitored with same threshold values, then the TRCx[THRCH] must be programmed again.

Note: If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is less than the lower threshold, then the WDGxL interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt

WDGxH for high threshold violation is set. Thus, the user should avoid that situation as it could lead to misinterpretation of the watchdog interrupts.

Analog watchdog functionality

For each input channel the result of the comparison is reflected in the THROP bit in TRC register based on the converted analog values received by the analog watchdogs:

- If the converted data value is lower than the lower threshold then the THROP bit in TRC register will be set to 1.
- If the converted voltage is higher than the higher threshold then the THROP bit in TRC register will be set to 0.
- If the converted voltage lies between the upper and the lower threshold guard window then THROP bit in TRC register will keep its logic value.

The logic level of the THROP bit can be programmed by software. In fact, the user can decide to keep the behavior described or to invert the output logic level by setting the THRINV bit in the TRC register.

An example of the operation is shown in *Table 293*.

Table 293. Example for Analog watchdog operation

Converted data watchdog[x]	Upper threshold watchdog[x]	Lower threshold watchdog[x]	THRINV watchdog[x]	THROP[x]
155h	055h	000h	0	0
055h	1ffh	088h	0	1
155h	055h	000h	1	1
055h	1ffh	088h	1	0

23.3.6 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set, the DMA request is cleared the register enabled for DMA transfer has been read.

23.3.7 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
- WDGxL and WDGxH (watchdog threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in register description for ISR and IMR.

The analog watchdog interrupts are handled by two 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 INTC module. The Watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

Note: In order to reduce the number of interrupt lines, EOC, ECH, JEOC, JECH and EOCTU are combined (OR-ed) on the same line ADC_EOC, and WDG0L, WDG0H, WDG1L, WDG1H, WDG2L, WDG2H, WDG3L and WDG3H are combined (OR-ed) on the same line ADC_WD. According to this, the total number of interrupt lines is 2. If the ADC is in CTU Control Mode, only the sources EOCTU, WDG0L, WDG0H, WDG1L, WDG1H, WDG2L, WDG2H, WDG3L and WDG3H can generate an interrupt request.

23.3.8 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN]. 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 resetting the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN]. If a conversion is ongoing, the ADC must complete the conversion before entering 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 ABORTCHAIN bit.

MSR[ADCSTATUS] bit is set only when ADC enters power-down mode.

After the power-down phase is completed the process ongoing before the power-down phase must be restarted manually by setting the appropriate MCR[START] bit.

Resetting MCR[PWDN] bit and setting MCR[NSTART] or MCR[JSTART] bit during the same cycle is forbidden.

If a CTU trigger pulse is received during power-down, it is discarded.

If the CTU is enabled and the MSR[CTUSTART] bit is ‘1’, then the MCR[PWDN] bit cannot be set.

23.3.9 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.

Note: The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when MCR.ADCCLKSEL = 1, the analog clock will not shut down in IDLE mode.

23.4 Register descriptions

23.4.1 Introduction

Table 294 lists ADC registers with their address offsets and reset values.

Table 294. ADC digital registers

Offset from base address 0xFFE0_0000	Register name	Location
0x0000	Main Configuration Register (MCR)	on page 23-588
0x0004	Main Status Register (MSR)	on page 23-590
0x0008–0x000F	Reserved	
0x0010	Interrupt Status Register (ISR)	on page 23-591
0x0014–0x001F	Reserved	
0x0020	Interrupt Mask Register (IMR)	on page 23-592
0x0024–0x002F	Reserved	
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	on page 23-593
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	on page 23-594
0x0038–0x003F	Reserved	
0x0040	DMA Enable Register (DMAE)	on page 23-595
0x0044	DMA Channel Select Register 0 (DMAR0)	on page 23-596
0x0048–0x004F	Reserved	
0x0050	Threshold Control Register 0 (TRC0)	on page 23-597
0x0054	Threshold Control Register 1 (TRC1)	on page 23-597
0x0058	Threshold Control Register 2 (TRC2)	on page 23-597
0x005C	Threshold Control Register 3 (TRC3)	on page 23-597
0x0060	Threshold Register 0 (THRHLR0)	on page 23-598
0x0064	Threshold Register 1 (THRHLR1)	on page 23-598
0x0068	Threshold Register 2 (THRHLR2)	on page 23-598
0x006C	Threshold Register 3 (THRHLR3)	on page 23-598
0x0070–0x0093	Reserved	
0x0094	Conversion Timing Register 0 (CTR0)	on page 23-599
0x0098–0x00A3	Reserved	
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	on page 23-599
0x00A8–0x00B3	Reserved	
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	on page 23-600
0x00B8–00C7	Reserved	
0x00C8	Power-down Exit Delay Register (PDEDR)	on page 23-601
0x00CC–0x00FF	Reserved	
0x0100	Channel 0 Data Register (CDR0)	on page 23-601
0x0104	Channel 1 Data Register (CDR1)	on page 23-601
0x0108	Channel 2 Data Register (CDR2)	on page 23-601

Table 294. ADC digital registers

Offset from base address 0xFFE0_0000	Register name	Location
0x010C	Channel 3 Data Register (CDR3)	on page 23-601
0x0110	Channel 4 Data Register (CDR4)	on page 23-601
0x0114	Channel 5 Data Register (CDR5)	on page 23-601
0x0118	Channel 6 Data Register (CDR6)	on page 23-601
0x011C	Channel 7 Data Register (CDR7)	on page 23-601
0x0120	Channel 8 Data Register (CDR8)	on page 23-601
0x0124	Channel 9 Data Register (CDR9)	on page 23-601
0x0128	Channel 10 Data Register (CDR10)	on page 23-601
0x012C	Channel 11 Data Register (CDR11)	on page 23-601
0x0130	Channel 12 Data Register (CDR12)	on page 23-601
0x0134	Channel 13 Data Register (CDR13)	on page 23-601
0x0138	Channel 14 Data Register (CDR14)	on page 23-601
0x013C	Channel 15 Data Register (CDR15)	on page 23-601

23.4.2 Control logic registers

Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Figure 285. Main Configuration Register (MCR)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	0	0	0	0	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	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	ADCLK SEL	ABORT CHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 295. MCR field descriptions

Field	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 15 to (15 – resolution + 1)). The WLSIDE bit affects all the CDR registers simultaneously. See *Figure 300* and *Figure 300*.
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. This bit stays high while the conversion is ongoing (or pending during injection mode). 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 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 will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.
CTUEN	Cross trigger unit conversion enable 0 CTU triggered conversion disabled 1 CTU triggered conversion enabled
ADCLKSEL	Analog clock select This bit can only be written when ADC in Power-Down mode 0 ADC clock frequency is half Peripheral Set Clock frequency 1 ADC clock frequency is equal to Peripheral Set Clock frequency
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

Table 295. MCR field descriptions (continued)

Field	Description
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. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned. 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 disabled 1 Auto clock off 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

Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Figure 286. Main Status Register (MSR)

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	NSTART	JABORT	0	0	JSTART	0	0	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	ACK0	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 296. MSR field descriptions

Field	Description
NSTART	This status bit is used to signal that a Normal conversion is ongoing.
JABORT	This status bit is used to signal that an Injected conversion has been aborted. This bit is reset when a new injected conversion starts.
JSTART	This status bit is used to signal that an Injected conversion is ongoing.

Table 296. MSR field descriptions (continued)

Field	Description
CTUSTART	This status bit is used to signal that a CTU conversion is ongoing.
CHADDR	Current conversion channel address This status field indicates current conversion channel address.
ACKO	Auto-clock-off enable This status bit is used to signal if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 IDLE — The ADC is powered up but idle. 001 Power-down — The ADC is powered down. 010 Wait state — The ADC is waiting for an external multiplexer. This occurs only when the DSDR register is non-zero. 011 Reserved 100 Sample — The ADC is sampling the analog signal. 101 Reserved 110 Conversion — The ADC is converting the sampled signal. 111 Reserved

Note: MSR[JSTART] is automatically set when the injected conversion starts. At the same time MCR[JSTART] is reset, allowing the software to program a new start of conversion.

The JCMR registers do not change their values.

23.4.3 Interrupt registers

Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Figure 287. Interrupt Status Register (ISR)

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	EOCTU	JEOC	JECH	EOC	ECH
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 297. ISR field descriptions

Field	Description
EOCTU	End of CTU Conversion interrupt flag When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt flag When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt flag When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt flag When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt flag When this bit is set, an ECH interrupt has occurred.

Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Figure 288. Interrupt Mask Register (IMR)

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	0	0	0	0	0	0	0	0	0	0	0	MSKE OCTU	MSK JEOC	MSK JECH	MSK EOC	MSK ECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 298. IMR field descriptions

Field	Description
MSKEOCTU	Mask for end of CTU conversion (EOCTU) interrupt When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask for end of injected channel conversion (JEOC) interrupt When set, the JEOC interrupt is enabled.
MSKJECH	Mask for end of injected chain conversion (JECH) interrupt When set, the JECH interrupt is enabled.
MSKEOC	Mask for end of channel conversion (EOC) interrupt When set, the EOC interrupt is enabled.
MSKECH	Mask for end of chain conversion (ECH) interrupt When set, the ECH interrupt is enabled.

Watchdog Threshold Interrupt Status Register (WTISR)

Figure 289. 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	CIM 15	CIM 14	CIM 13	CIM 12	CIM 11	CIM 10	CIM 9	CIM 8	CIM 7	CIM 6	CIM 5	CIM 4	CIM 3	CIM 2	CIM 1	CIM 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 290. 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	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	WDG3H	WDG2H	WDG1H	WDG0H	WDG3L	WDG2L	WDG1L	WDG0L
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

Table 299. WTISR field descriptions

Field	Description
WDGxH	This corresponds to the status flag generated on the converted value being higher than the programmed higher threshold (for [x = 0..3]).
WDGxL	This corresponds to the status flag generated on the converted value being lower than the programmed lower threshold (for [x = 0..3]).

Watchdog Threshold Interrupt Mask Register (WTIMR)

Figure 291. 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	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	MSKWDG3H	MSKWDG2H	MSKWDG1H	MSKWDG0H	MSKWDG3L	MSKWDG2L	MSKWDG1L	MSKWDG0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 300. WTIMR field descriptions

Field	Description
MSKWDGxH	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0..3]). When set the interrupt is enabled.
MSKWDGxL	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0..3]). When set the interrupt is enabled.

23.4.4 DMA registers

DMA Enable (DMAE) register

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Figure 292. DMA Enable (DMAE) register

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

Table 301. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller 1 DMA request cleared on read of data registers
DMAEN	DMA global enable 0 DMA feature disabled 1 DMA feature enabled

DMA Channel Select Register (DMAR[0])

DMAR0 = Enable bits for channel 0 to 15 (precision channels)

Figure 293. DMA Channel Select Register 0 (DMAR0)

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

Table 302. DMARx field descriptions

Field	Description
DMAn	DMA enable When set (DMAn = 1), channel n is enabled to transfer data in DMA mode.

23.4.5 Threshold registers

Introduction

The Threshold registers store the user programmable lower and upper thresholds' values. The inverter bit and the mask bit for mask the interrupt are stored in the TRC registers.

Threshold Control Register (TRC*x*, *x* = [0..3])

Figure 294. Threshold Control Register (TRC*x*, *x* = [0..3])

Address: Base + 0x0050 (TRC0)
Base + 0x0054 (TRC1)
Base + 0x0058 (TRC2)
Base + 0x005C (TRC3)

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	THR EN	THR INV	THR OP	0	0	0	0	0	0	THRCH						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 303. TRC*x* field descriptions

Field	Description
THREN	Threshold enable When set, this bit enables the threshold detection feature for the selected channel.
THRINV	Invert the output pin Setting this bit inverts the behavior of the threshold output pin.
THROP	This bit reflects the output pin status. See *Section , Analog watchdog functionality."* It reflects the output pin status.
THRCH	Choose the channel for threshold comparison.

Threshold Register (THRHLR[0:3])

The four THRHLR*n* registers store the user-programmable thresholds' 10-bit values.

Figure 295. Threshold Register (THRHLR[0:3])

Address: Base + 0x0060 (THRHLR0)
Base + 0x0064 (THRHLR1)
Base + 0x0068 (THRHLR2)
Base + 0x006C (THRHLR3)

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	THRH									
W																
Reset	0	0	0	0	0	0	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	0	0	THRL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 304. THRHLR*x* field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

23.4.6 Conversion Timing Registers CTR[0]

CTR0 = associated to internal precision channels (from 0 to 15)

Figure 296. Conversion Timing Registers CTR[0]

Address: Base + 0x0094 (CTR0) 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	1	0	1

Table 305. CTR field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration
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 00h and 001h is reached when the A_{VIN} is equal to 0 11 Not used Available only on CTR0
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

23.4.7 Mask registers

Introduction

The Mask registers are used to program the 16 input channels that are converted during Normal and Injected conversion.

Normal Conversion Mask Registers (NCMR[0])

NCMR0 = Enable bits of normal sampling for channel 0 to 15 (precision channels)

Figure 297. Normal Conversion Mask Register 0 (NCMR0)

Address: Base + 0x00A4 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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 306. NCMR field descriptions

Field	Description
CHn	Sampling enable When set Sampling is enabled for channel n.

Injected Conversion Mask Registers (JCMR[0])

JCMR0 = Enable bits of injected sampling for channel 0 to 15 (precision channels)

Figure 298. Injected Conversion Mask Register 0 (JCMR0)

Address: Base + 0x00B4 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	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 307. JCMR field descriptions

Field	Description
CHn	Sampling enable When set, sampling is enabled for channel n.

23.4.8 Delay registers

Power-Down Exit Delay Register (PDEDR)

Figure 299. 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

Table 308. PDEDR field descriptions

Field	Description
PDED	Delay between the power-down bit reset and the start of conversion. The delay is to allow time for the ADC power supply to settle before commencing conversions. The power down delay is calculated as: PDED x 1/frequency of ADC clock.

23.4.9 Data registers

Introduction

ADC conversion results are stored in data registers. There is one register per channel.

Channel Data Registers (CDR[0..15])

CDR[0..15] = precision channels

Each data register also gives information regarding the corresponding result as described below.

Figure 300. Channel Data Registers (CDR[0..26])

Address: See *Table 294* 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	0	0	CDATA[0:9] (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[0:9] (MCR[WLSIDE] = 1)										0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 309. CDR 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 bit 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 0-15 converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this field can be changed as shown in *Figure 300*.

24 Cross Triggering Unit (CTU)

24.1 Introduction

In 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-cos 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 cases the pre-setting of the acquisition times 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. Additionally, four FIFOs inside the CTU are available to store the ADC results.

24.2 CTU overview

The CTU receives various incoming signals from different sources (PWM, timers, position decoder and/or external pins). These signals are then processed to generate as many as eight trigger events. An input can be a rising edge, a falling edge or both, edges of each incoming signal. The output can be a pulse or a command (or a stream of consecutive commands for over-sampling support) or both, to one or more peripherals (for example, ADC or timers).

The CTU interfaces to the following peripherals:

- PWM—13 inputs
- eTimer—1 input
- GPIO—1 external input signal

The 16 input signals are digital signals and the CTU must be able to detect a rising and/or a falling edge for each of them.

The CTU comprises the following:

- Input signals interface
- User interface (such as configuration registers)
- ADC interface
- Timers interface

The block diagram of the CTU is shown in *Figure 301*.

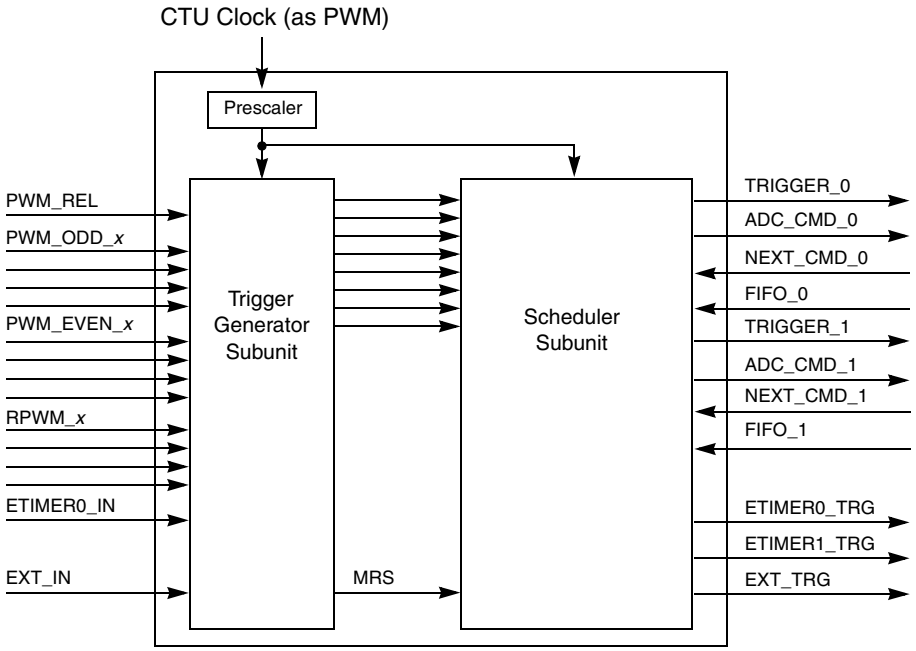


Figure 301. Cross triggering unit diagram

The CTU consists of two subunits:

- Trigger generator
- Scheduler

The trigger generator subunit handles incoming signals, selecting for each signal, the active edges to generate the Master Reload signal, and generates as many as eight trigger events (signals). The scheduler subunit generates the trigger event output according to the occurred trigger event (signal).

24.3 Functional description

The following describes the functionality of the CTU.

24.3.1 Trigger events features

The TGS has the capability to generate as many as eight trigger events. Each trigger event has the following characteristics:

- Generation of the trigger event is sequential in time
- The triggers list uses eight 16-bit double-buffered registers
- On each Master Reload Signal (MRS), the new triggers list is loaded
- The triggers list is reloaded on a MRS occurrence, only if the reload enable bit is set

24.3.2 Trigger generator subunit (TGS)

The trigger generator subunit has the following two modes:

- Triggered mode—Each event source for the incoming signals can generate as many as eight trigger event outputs. For the ADC, a commands list is entered by the CPU, and each event source can generate as many as 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 is selected using the TGS_M bit in the TGS Control Register.

24.3.3 TGS in triggered mode

The structure of the TGS in Triggered mode is shown in *Figure 302*.

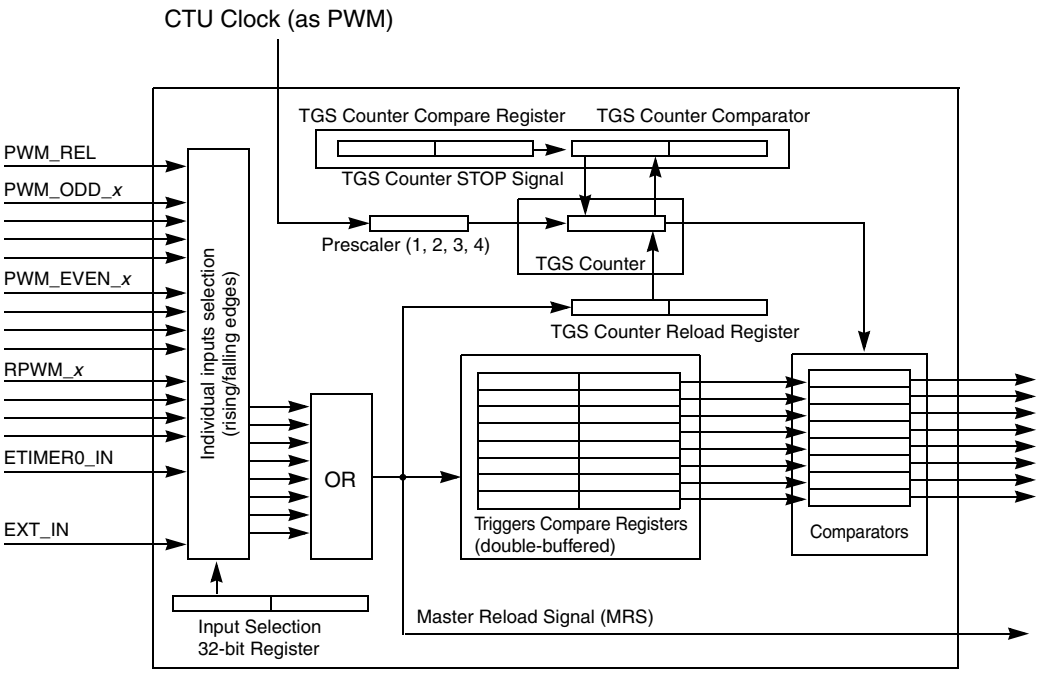


Figure 302. TGS in triggered mode

The TGS has 16 input signals, each of which is selected from 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 ORed in order to generate the MRS. The MRS, at the beginning of the control cycle *n* (defined by the MRS occurrence), preloads the TGS counter register, using the preload value written into the

double-buffered register (TGSCRR), during the control cycle *n* – 1 and reloads all the double-buffered registers (such as Trigger Compare registers, TGSCR, TGSCRR itself).

The triggers list registers 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 MRS_SG bit in the CTU control register, if set to 1, generates equivalent software MRS (that is, resets/reloads 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 303*. 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.

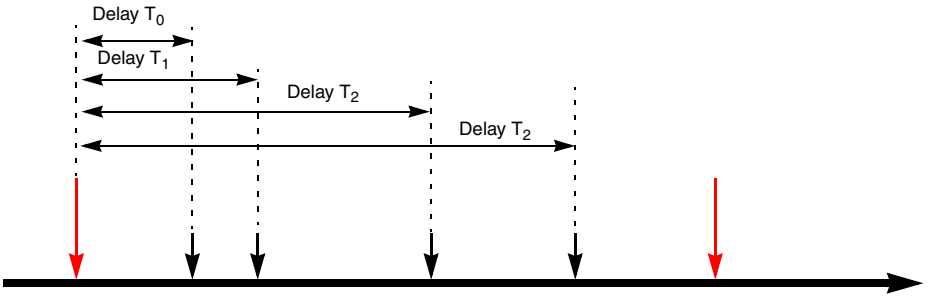


Figure 303. Example timing for TGS in triggered mode

24.3.4 TGS in sequential mode

The structure of the TGS in sequential mode is shown in *Figure 304*.

The 32 input events (16 signals with two edges for signal), which can be individually enabled, are ORed in order to generate the event signal (ES). The ES enables the reload of the TGS counter register and pilots the 3-bit counter in order to select the next active trigger. One of the 32 input events can be selected, through the MRS_SM (master reload selection sequential mode) bits in the TGS control register, to be the MRS, that enables the reload of the triggers list and resets the 3-bit counter (incoming events counter), that is, the MRS is the signal linked with the control cycle defined as the time window between two consecutive MRSs. 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 that enables, according to the number of event signals occurred, only one of the eight trigger signals to the scheduler subunit. Sequences of as many as eight trigger events can be supported within this control cycle.

For the other features see the previous paragraphs.

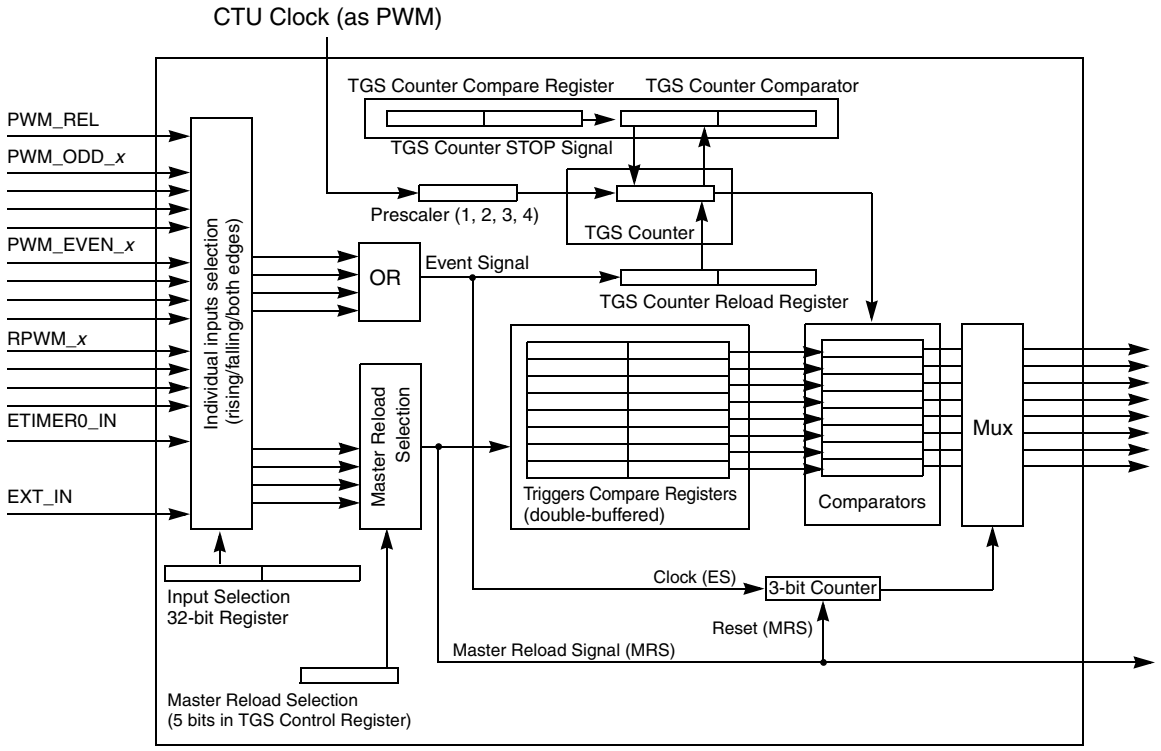


Figure 304. TGS in sequential mode

An example timing diagram for TGS in sequential mode is shown in *Figure 305*. 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.

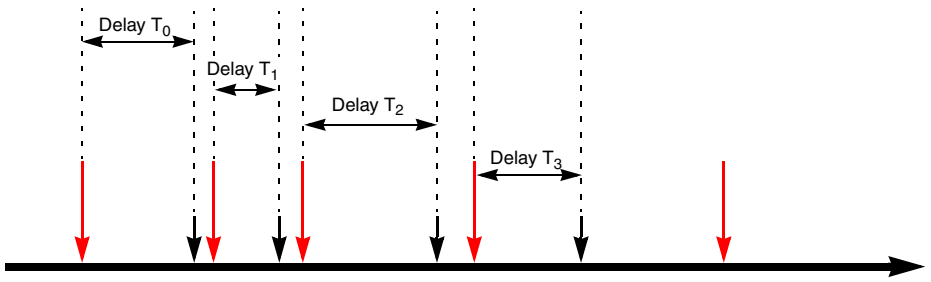


Figure 305. Example timing for TGS in sequential mode

24.3.5 TGS counter

The TGS counter is able to count from negative to positive, that is, from 0x8000 to 0x7FFF. *Figure 306* 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.

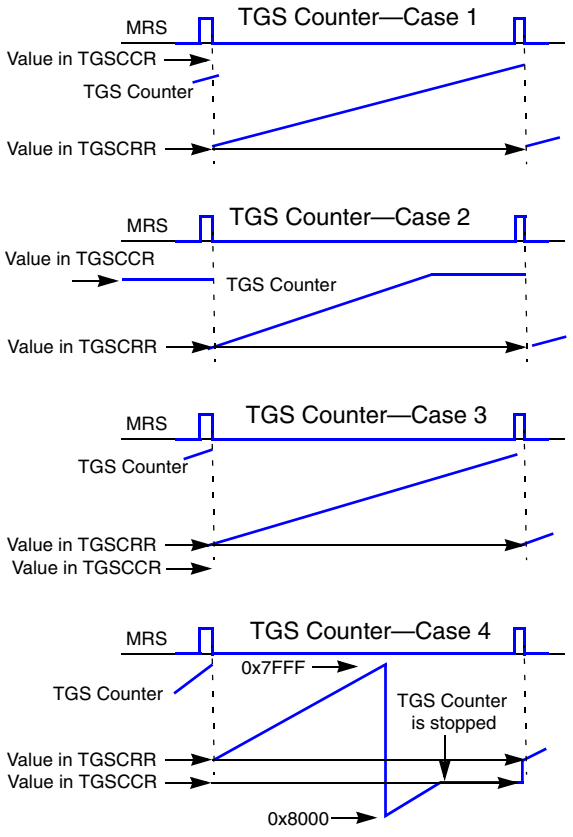


Figure 306. TGS counter cases

24.4 Scheduler subunit (SU)

The structure of the SU is shown in *Figure 307*.

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 four SU outputs:

- ADC command or ADC stream of commands
- eTimer1 pulse (ETIMER0_TRG internally connected to eTimer_0 AUX0)
- eTimer2 pulse (ETIMER1_TRG internally connected to eTimer_0 AUX1)
- 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 will be generated only if the ON time + Guard Time has past. The ON-Time and the Guard Time are only used for external Triggers.

External signals can be asynchronous with motor control clock. For this reason a programmable digital filter is available. The external signal is considered at 1 if it is latched N time at 1, and is considered at 0 if it is latched N time at 0, where N is a value in the digital filter control register.

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 CTU Control Register is used to generate an equivalent software trigger event. Each of these bits is cleared by a respective hardware or software trigger event.

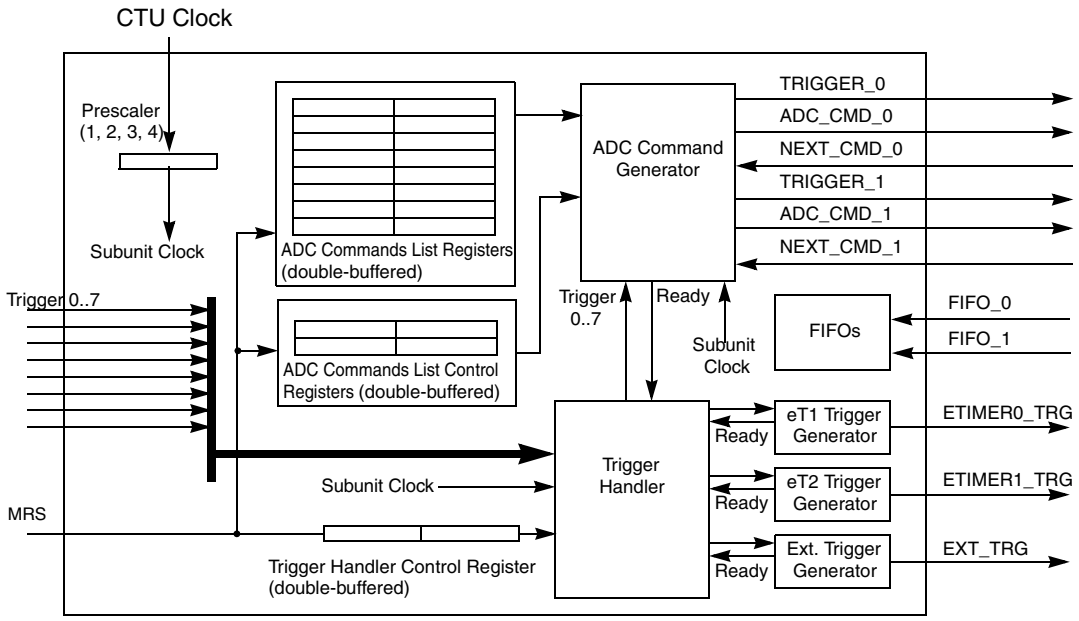


Figure 307. Scheduler subunit

24.4.1 ADC commands list

The ADC can be controlled by the CPU (CPU Control Mode) and 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 can not 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 twenty-four 16-bit commands (see *Section 24.4.2, "ADC commands list format*) and it is double-buffered, that is, the commands list can be updated at any time between two consecutive MRS, but the changes become workable 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 CLCRx (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.

24.4.2 ADC commands list format

The CTU can be interfaced with two ADCs, supporting the Single Conversion Mode and the Dual Conversion Mode.

In Single Conversion Mode only one ADC starts a conversion at a time. In Dual Conversion Mode both ADCs start a conversion at the same time; in particular both the ADC

conversions are performed at the same time while the storage of the results is performed in series. In Dual Conversion Mode, 4 bits select each channel number, and the conversion mode selection bit selects 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 24.8.10, "Commands list register x (x = 1,...,24) (CLRx)*.

The result of each conversion is stored in one of the four available FIFOs.

The interrupt request bit is used as an interrupt request when 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 (FC) bit is checked to see if it is 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 0: number of ADC channel to sample from ADC unit 0 (4 bits)
- Channel 1: number of ADC channel to sample from ADC unit 1 (4 bits)
- FIFO selection bits for the ADC unit 0/1 (2 bits)
- Conversion Mode selection bit
- First command bit (only for CTU internal use)
- Interrupt request bit (only for CTU internal use)

On this device, only ADC_0 is implemented so a new CTU/ADC interface is implemented in order to interface the CTU and the only ADC_0. This new CTU/ADC interface is a logic machine between the CTU outputs and the ADC_0 inputs and it has no configuration registers. It is implemented to ensure software compatibility between SPC560P40/34 and the 512 Kbyte memory family device, in fact it is able to virtualize ADC_1 on SPC560P40/34, so, for example, the user can write on SPC560P40/34 a command to start a conversion on ADC_1 channels 0 and the CTU/ADC interface will translate this command into a command for ADC_0 channel 6. Moreover CTU/ADC interface will manage software programming mistakes for SPC560P40/34 as not allowed Dual Conversion Mode or wrong ADC_0/1 channel number selection to ensure that the CTU does not go in a blocking status.

Table 310. ADC commands translation

Input command	Output command
Single sampling ADC_0 channel 0	Single sampling ADC_0 channel 0
Single sampling ADC_0 channel 1	Single sampling ADC_0 channel 1
Single sampling ADC_0 channel 2	Single sampling ADC_0 channel 2
Single sampling ADC_0 channel 3	Single sampling ADC_0 channel 3
Single sampling ADC_0 channel 4	Single sampling ADC_0 channel 4
Single sampling ADC_0 channel 5	Single sampling ADC_0 channel 5
Single sampling ADC_0 channel 6	Not valid - force EOC to CTU
Single sampling ADC_0 channel 7	Not valid - force EOC to CTU
Single sampling ADC_0 channel 8	Not valid - force EOC to CTU
Single sampling ADC_0 channel 9	Not valid - force EOC to CTU
Single sampling ADC_0 channel 10	Not valid - force EOC to CTU

Table 310. ADC commands translation (continued)

Input command	Output command
Single sampling ADC_0 channel 11	Single sampling ADC_0 channel 11
Single sampling ADC_0 channel 12	Single sampling ADC_0 channel 12
Single sampling ADC_0 channel 13	Single sampling ADC_0 channel 13
Single sampling ADC_0 channel 14	Single sampling ADC_0 channel 14
Single sampling ADC_0 channel 15	Single sampling ADC_0 channel 15
Single sampling ADC_1 Channel 0	Single sampling ADC_0 Channel 6
Single sampling ADC_1 Channel 1	Single sampling ADC_0 Channel 7
Single sampling ADC_1 Channel 2	Single sampling ADC_0 Channel 8
Single sampling ADC_1 Channel 3	Single sampling ADC_0 Channel 9
Single sampling ADC_1 Channel 4	Single sampling ADC_0 Channel 10
Single sampling ADC_1 Channel 5	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 6	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 7	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 8	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 9	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 10	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 11	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 12	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 13	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 14	Not valid - force EOC to CTU
Single sampling ADC_1 Channel 15	Not valid - force EOC to CTU
Dual sampling ADC_0 channel x / ADC_1 channel y	Not valid - force EOC to CTU

24.4.3 ADC results

ADC results can be stored in the channel relevant standard result register and/or in one of the four FIFOs: the different FIFOs allow to dispatch ADC results according to their type of acquisition (for example phase currents, rotor position or ground-noise). 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, that is, 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:

- FIFO1 and FIFO2—16 entries (sized to avoid overflow during a full PWM period for current acquisitions)
- FIFO3 and FIFO4—4 entries (low acquisition rate FIFOs)

Results in each FIFO can be read by a 16-bit read transaction (only the result is read in order to minimize the CPU load before computing on results) or by a 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 [9:0] are used for 10-bit resolution and bits [15:10] always return zero when read.

- Signed left-justified

  Conversion result is signed left-justified data. Bit [15] is reserved for sign and is always read as zero for this ADC, bits [14:5] are used for 10-bit resolution, and bits [4:0] always return zero when read.

24.5 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, eg. 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, 1 bit enables the reload operation, that is, 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 guarantee the coherency, the reload of all double-buffered registers is enabled by setting GRE (General Reload Enable) bit in the CTU Control Register. The user must ensure that all intended double-buffered registers are updated before a new MRS occurrence. If an MRS occurs before a 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 General Reload Enable (GRE) bit to enable the reload when the MRS occurs. The GRE bit is R/S (Read/Set) and if this bit is 1, the reload can be performed, while if this bit is 0, the reload is not performed. A correct reload resets the GRE bit. None of the double-buffered registers can be written while the GRE bit remains set. The GRE bit can be reset by the occurrence of the next MRS (that is, a correct reload) or by software setting the CGRE bit.

The CGRE is reset by hardware after that GRE bit is reset. If the user sets the CGRE bit and at the same time a MRS occurs, CGRE has the priority so GRE is reset and the reload is not performed. In the same way, the GRE has the priority when compared with the MRS occurrence, and the CGRE has the priority compared with the 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, FGRE (Flag GRE bit in the CTU Control Register) bit is used. When one of the double-buffered registers is written, this flag is set to 1 and it is reset by a correct reload. When the MRS occurs while FGRE is 1 and GRE is 1, a correct reload is performed (because all intended registers have been updated before the MRS occurs). If FGRE is 1 and 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 FGRE is 0 it is not necessary to perform a reload because all the double-buffered registers are unchanged (see *Figure 308*).

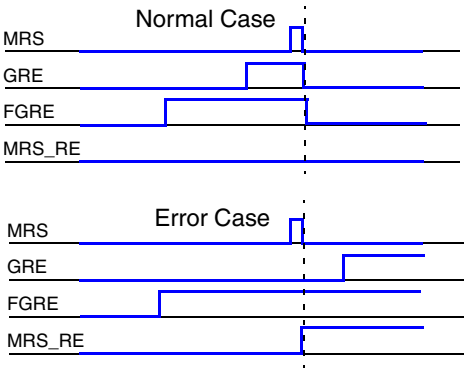


Figure 308. Reload error scenario

24.6 Power safety mode

To reduce power consumption two mechanisms are implemented:

- MDIS bit in the CTUPCR
- STOP mode

24.6.1 MDIS bit

The MDIS bit in the CTUPCR is used for stopping the clock to all non memory mapped registers.

24.6.2 STOP mode

To reduce consumption, it is also possible to enable a stop request from the Mode Entry module. 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. 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 after a MDIS bit setting the FIFO have to be empty. In order to avoid the problems linked to a wrong trigger, the CTU output can be disabled by the CTU_ODIS bit and the ADC interface state machine can be reset by the CRU_ADC_R (see *Section 24.8.21, “Cross triggering unit control register (CTUCR)*).

24.7 Interrupts and DMA requests

24.7.1 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, that is, only if the 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.

24.7.2 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 MRS_RE bit is set.
- Receiving more than eight EVs before that the next MRS occurs in TGS sequential mode and the SM_TO bit is set.
- A trigger event occurs during the time when the actions of the previous trigger event are not completed (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).

  There are four overrun flags (one for each type of output). The general mechanism shall be as in *Figure 307*.

  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 shall be 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 the user to pre-set a trigger to the eTimer0 in the middle of an ADC conversion, that is, 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 following bits is set: ADC_OE, T0_OE, T1_OE, or ET_OE.
- Invalid (unrecognized) ADC command and the 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 than the trigger event selected by the previous EV occurs. The incoming EV sets an internal busy flag.

The outgoing trigger event (all line are ORed) resets this flag to 0. TGS Overrun in the sequential mode shall be 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 CTU error flag register and in the CTU interrupt flag register can be cleared by writing a 1 while writing a 0 has no effect. The CTU does not support a write-protection mechanism.

24.7.3 CTU interrupt/DMA requests

The CTU can perform the following interrupt/DMA requests (15 interrupt lines):

- Error interrupt request (see *Section 24.7.2, "CTU faults and errors*) (1 interrupt line)
- ADC command interrupt request (1 interrupt line)
- Interrupt request on MRS occurrence (1 interrupt line)
- Interrupt request on each trigger event occurrence (1 interrupt line for each trigger event)
- FIFOs interrupt requests and/or DMA transfer request (1 interrupt line for each FIFO)
- DMA transfer request on the MRS occurrence if GRE bit is set

The interrupt flags are shown in *Table 311*.

Table 311. CTU interrupts

Category	Interrupt	Interrupt function
Managed individually	MRS_I	MRS Interrupt flag (IRQ193)
	T0_I	Trigger 0 interrupt flag (IRQ194)
	T1_I	Trigger 1 interrupt flag (IRQ195)
	T2_I	Trigger 2 interrupt flag (IRQ196)
	T3_I	Trigger 3 interrupt flag (IRQ197)
	T4_I	Trigger 4 interrupt flag (IRQ198)
	T5_I	Trigger 5 interrupt flag (IRQ199)
	T6_I	Trigger 6 interrupt flag (IRQ200)
	T7_I	Trigger 7 interrupt flag (IRQ201)
	ADC_I	ADC command interrupt flag (IRQ206)
ORed onto FIFO1_I (IRQ202)	FIFO_FULL0	This bit is set to 1 if the FIFO 0 is full.
	FIFO_EMPTY0	This bit is set to 1 if the FIFO 0 is empty.
	FIFO_OVERFLOW 0	This bit is set to 1 if the number of words exceeds the value set in the threshold 0.
	FIFO_OVERRUN0	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL0 flag is set.

Table 311. CTU interrupts (continued)

Category	Interrupt	Interrupt function
ORed onto FIFO2_I (IRQ203)	FIFO_FULL1	This bit is set to 1 if the FIFO 1 is full.
	FIFO_EMPTY1	This bit is set to 1 if the FIFO 1 is empty.
	FIFO_OVERFLOW 1	This bit is set to 1 if the number of words exceeds the value set in the threshold 1.
	FIFO_OVERRUN1	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL1 flag is set.
ORed onto FIFO3_I (IRQ204)	FIFO_FULL2	This bit is set to 1 if the FIFO 2 is full.
	FIFO_EMPTY2	This bit is set to 1 if the FIFO 2 is empty.
	FIFO_OVERFLOW 2	This bit is set to 1 if the number of words exceeds the value set in the threshold 2.
	FIFO_OVERRUN2	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL2 flag is set.
ORed onto FIFO4_I (IRQ205)	FIFO_FULL3	This bit is set to 1 if the FIFO 3 is full.
	FIFO_EMPTY3	This bit is set to 1 if the FIFO 3 is empty.
	FIFO_OVERFLOW 3	This bit is set to 1 if the number of words exceeds the value set in the threshold 3.
	FIFO_OVERRUN3	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL3 flag is set.
ORed onto ERR_I (IRQ207)	MRS_RE	Master Reload Signal Reload Error
	SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode
	ICE	Invalid Command Error
	MRS_O	Master Reload Signal Overrun
	TGS_OSM	TGS Overrun in Sequential Mode
	ADC_OE	ADC command generation Overrun Error
	T0_OE	Timer 0 trigger generation Overrun Error
	T1_OE	Timer 1 trigger generation Overrun Error
	ET_OE	External Trigger generation Overrun Error

24.8 Memory map

Table 312. CTU memory map

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	*on page 24-621*
0x0004	TGSCRR — Trigger Generator Subunit Control Register	*on page 24-624*
0x0006	T0CR — Trigger 0 Compare Register	*on page 24-624*

Table 312. CTU memory map (continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0008	T1CR — Trigger 1 Compare Register	on page 24-624
0x000A	T2CR — Trigger 2 Compare Register	on page 24-624
0x000C	T3CR — Trigger 3 Compare Register	on page 24-624
0x000E	T4CR — Trigger 4 Compare Register	on page 24-624
0x0010	T5CR — Trigger 5 Compare Register	on page 24-624
0x0012	T6CR — Trigger 6 Compare Register	on page 24-624
0x0014	T7CR — Trigger 7 Compare Register	on page 24-624
0x0016	TGSCCR — TGS Counter Compare Register	on page 24-625
0x0018	TGSCRR — TGS Counter Reload Register	on page 24-625
0x001A	Reserved	
0x001C	CLCR1 — Commands List Control Register 1	on page 24-626
0x0020	CLCR2 — Commands List Control Register 2	on page 24-626
0x0024	THCR1 — Trigger Handler Control Register 1	on page 24-627
0x0028	THCR2 — Trigger Handler Control Register 2	on page 24-629
0x002C	CLR1—Commands List Register 1	on page 24-631
0x002E	CLR2—Commands List Register 2	on page 24-631
0x0030	CLR3—Commands List Register 3	on page 24-631
0x0032	CLR4—Commands List Register 4	on page 24-631
0x0034	CLR5—Commands List Register 5	on page 24-631
0x0036	CLR6—Commands List Register 6	on page 24-631
0x0038	CLR7—Commands List Register 7	on page 24-631
0x003A	CLR8—Commands List Register 8	on page 24-631
0x003C	CLR9—Commands List Register 9	on page 24-631
0x003E	CLR10—Commands List Register 10	on page 24-631
0x0040	CLR11—Commands List Register 11	on page 24-631
0x0042	CLR12—Commands List Register 12	on page 24-631
0x0044	CLR13—Commands List Register 13	on page 24-631
0x0046	CLR14—Commands List Register 14	on page 24-631
0x0048	CLR15—Commands List Register 15	on page 24-631
0x004A	CLR16—Commands List Register 16	on page 24-631
0x004C	CLR17—Commands List Register 17	on page 24-631
0x004E	CLR18—Commands List Register 18	on page 24-631
0x0050	CLR19—Commands List Register 19	on page 24-631

Table 312. CTU memory map (continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0052	CLR20—Commands List Register 20	on page 24-631
0x0054	CLR21—Commands List Register 21	on page 24-631
0x0056	CLR22—Commands List Register 22	on page 24-631
0x0058	CLR23—Commands List Register 23	on page 24-631
0x005A	CLR24—Commands List Register 24	on page 24-631
0x005C—0x006B	Reserved	
0x006C	FDCR — FIFO DMA Control Register	on page 24-632
0x0070	FCR — FIFO Control Register	on page 24-633
0x0074	FTH — FIFO Threshold Register	on page 24-634
0x0078–0x007B	Reserved	
0x007C	FST — FIFO Status Register	on page 24-635
0x0080	FR0 — FIFO Right aligned data register 0	on page 24-636
0x0084	FR1 — FIFO Right aligned data register 1	on page 24-636
0x0088	FR2 — FIFO Right aligned data register 2	on page 24-636
0x008C	FR3 — FIFO Right aligned data register 3	on page 24-636
0x0080–0x009F	Reserved	
0x00A0	FL0 — FIFO Left aligned data register 0	on page 24-637
0x00A4	FL1 — FIFO Left aligned data register 1	on page 24-637
0x00A8	FL2 — FIFO Left aligned data register 2	on page 24-637
0x00AC	FL3 — FIFO Left aligned data register 3	on page 24-637
0x00B0–0x00BF	Reserved	
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	on page 24-637
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	on page 24-638
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	on page 24-639
0x00C6	COTR — Control ON-Time Register	on page 24-640
0x00C8	CTUCR — Cross triggering unit control register	on page 24-641
0x00CA	CTUDF — Cross Triggering Unit Digital Filter register	on page 24-642
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	on page 24-642
0x00CE–0x3FFF	Reserved	

Table 313. TGS registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	Yes	TGSISR_RE	0x0000_0000
0x0004	TGSCRR — Trigger Generator Subunit Control Register	Yes	MRS	0x0000
0x0006	T0CR — Trigger 0 Compare Register	Yes	MRS	0x0000
0x0008	T1CR — Trigger 1 Compare Register	Yes	MRS	0x0000
0x000A	T2CR — Trigger 2 Compare Register	Yes	MRS	0x0000
0x000C	T3CR — Trigger 3 Compare Register	Yes	MRS	0x0000
0x000E	T4CR — Trigger 4 Compare Register	Yes	MRS	0x0000
0x0010	T5CR — Trigger 5 Compare Register	Yes	MRS	0x0000
0x0012	T6CR — Trigger 6 Compare Register	Yes	MRS	0x0000
0x0014	T7CR — Trigger 7 Compare Register	Yes	MRS	0x0000
0x0016	TGSCCR — TGS Counter Compare Register	Yes	MRS	0x0000
0x0018	TGSCRR — TGS Counter Reload Register	Yes	MRS	0x0000

Table 314. SU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x001C	CLCR1 — Commands List Control Register 1	Yes	MRS	0x0000_0000
0x0020	CLCR2 — Commands List Control Register 2	Yes	MRS	0x0000_0000
0x0024	THCR1 — Trigger Handler Control Register 1	Yes	MRS	0x0000_0000
0x0028	THCR2 — Trigger Handler Control Register 2	Yes	MRS	0x0000_0000
0x002C – 0x005A	CLRx — Commands List Register x (x = 1,...,24)	Yes	MRS	0x0000

Table 315. CTU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	No	—	0x0000
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	No	—	0x0000
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	No	—	0x0000
0x00C6	COTR — Control ON-Time Register	Yes	MRS	0x0000
0x00C8	CTUCR — Cross triggering unit control register	No	—	0x0000

Table 315. CTU registers (continued)

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x00CA	CTUDF — Cross Triggering Unit Digital Filter	Yes	DFE	0x0000
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	No	—	0x0000

Table 316. FIFO registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x006C	FDCR — FIFO DMA Control Register	No	—	0x0000
0x0070	FCR — FIFO Control Register	No	—	0x0000_0000
0x0074	FTH — FIFO Threshold Register	No	—	0x0000_0000
0x007C	FST — FIFO Status Register	No	—	0x0000_0000
0x0080	FR0 — FIFO Right aligned data 0	No	—	0x0000_0000
0x0084	FR1 — FIFO Right aligned data 1	No	—	0x0000_0000
0x0088	FR2 — FIFO Right aligned data 2	No	—	0x0000_0000
0x008C	FR3 — FIFO Right aligned data 3	No	—	0x0000_0000
0x00A0	FL0 — FIFO Left aligned data 0	No	—	0x0000_0000
0x00A4	FL1 — FIFO Left aligned data 1	No	—	0x0000_0000
0x00A8	FL2 — FIFO Left aligned data 2	No	—	0x0000_0000
0x00AC	FL3 — FIFO Left aligned data 3	No	—	0x0000_0000

24.8.1 Trigger Generator Sub-unit Input Selection Register (TGSISR)

Figure 309. Trigger Generator Sub-unit Input Selection Register (TGSISR)

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

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	I15_FE	I15_RE	0	0	I13_FE	I13_RE	I12_FE	I12_RE	I11_FE	I11_RE	I10_FE	I10_RE	I9_FE	I9_RE	I8_FE	I8_RE
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	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
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 317. TGSISR field descriptions

Field	Description
I15_FE	Input 15 — ext_signals Falling edge Enable 0 Disabled 1 Enabled
I15_RE	Input 15 — ext_signals Rising edge Enable 0 Disabled 1 Enabled
I13_FE	Input 13 — eTimer0 [ETC2] Falling edge Enable 0 Disabled 1 Enabled
I13_RE	Input 13 — eTimer0 [ETC2] Rising edge Enable 0 Disabled 1 Enabled
I12_FE	Input 12 —PWM X[3] Falling edge Enable 0 Disabled 1 Enabled
I12_RE	Input 12 — PWM X[3] Rising edge Enable 0Disabled 1 Enabled
I11_FE	Input 11 — PWM X[2] Falling edge Enable 0 Disabled 1 Enabled
I11_RE	Input 11 — PWM X[2] Rising edge Enable 0 Disabled 1 Enabled
I10_FE	Input 10 — PWM X[1] Falling edge Enable 0Disabled 1 Enabled
I10_RE	Input 10 — PWM X[1] Rising edge Enable 0 Disabled 1 Enabled
I9_FE	Input 9 — PWM X[0] Falling edge Enable 0Disabled 1 Enabled
I9_RE	Input 9 — PWM X[0] Rising edge Enable 0 Disabled 1 Enabled
I8_FE	Input 8 — PWM OUT_TRIG 1 [3] Falling edge Enable 0 Disabled 1 Enabled
I8_RE	Input 8 — PWM OUT_TRIG 1 [3] Rising edge Enable 0 Disabled 1 Enabled

Table 317. TGSISR field descriptions (continued)

Field	Description
I7_FE	Input 7 — PWM OUT_TRIG 1 [2] Falling edge Enable 0 Disabled 1 Enabled
I7_RE	Input 7 — PWM OUT_TRIG 1 [2] Rising edge Enable 0 Disabled 1 Enabled
I6_FE	Input 6 — PWM OUT_TRIG 1 [1] Falling edge Enable 0 Disabled 1 Enabled
I6_RE	Input 6 — PWM OUT_TRIG 1 [1] Rising edge Enable 0 Disabled 1 Enabled
I5_FE	Input 5 — PWM OUT_TRIG 1 [0] Falling edge Enable 0 Disabled 1 Enabled
I5_RE	Input 5 — PWM OUT_TRIG 1 [0] Rising edge Enable 0 Disabled 1 Enabled
I4_FE	Input 4 — PWM OUT_TRIG 0 [3] Falling edge Enable 0 Disabled 1 Enabled
I4_RE	Input 4 — PWM OUT_TRIG 0 [3] Rising edge Enable 0 Disabled 1 Enabled
I3_FE	Input 3 — PWM OUT_TRIG 0 [2] Falling edge Enable 0 Disabled 1 Enabled
I3_RE	Input 3 — PWM OUT_TRIG 0 [2] Rising edge Enable 0 Disabled 1 Enabled
I2_FE	Input 2 — PWM OUT_TRIG 0 [1] Falling edge Enable 0 Disabled 1 Enabled
I2_RE	Input 2 — PWM OUT_TRIG 0 [1] Rising edge Enable 0 Disabled 1 Enabled h
I1_FE	Input 1 — PWM OUT_TRIG 0 [0] Falling edge Enable 0 Disabled 1 Enabled
I1_RE	Input 1 — PWM OUT_TRIG 0 [0] Rising edge Enable 0 Disabled 1 Enabled

Table 317. TGSISR field descriptions (continued)

Field	Description
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

24.8.2 Trigger Generator Sub-unit Control Register (TGSCR)

Figure 310. Trigger Generator Sub-unit Control Register (TGSCR)

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

Table 318. TGSCR field descriptions

Field	Description
ET_TM	This bit enables toggle mode for external triggers.
PRES	TGS and SU prescaler selection bits 00 1 01 2 10 3 11 4
MRS_SM	Master Reload Selection in Sequential Mode (5 bits to select one of 32 inputs)
TGS_M	Trigger Generator Subunit Mode 0 Triggered Mode 1 Sequential Mode

24.8.3 Trigger x Compare Register (TxCR, *x* = 0...7)

Figure 311. Trigger x Compare Register (TxCR, *x* = 0...7)

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	TxCRV															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 319. TxCR field descriptions

Field	Description
TxCRV	Trigger x Compare Register Value

24.8.4 TGS Counter Compare Register (TGSCCR)

Figure 312. TGS Counter Compare Register (TGSCCR)

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

Table 320. TGSCCR field format

Field	Description
TGSCCV	TGS Counter Compare Value

24.8.5 TGS Counter Reload Register (TGSCRR)

Figure 313. TGS Counter Reload Register (TGSCRR)

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

Table 321. TGSCRR field descriptions

Field	Description
TGSCRV	TGS Counter Reload Value

24.8.6 Commands list control register 1 (CLCR1)

Figure 314. Commands list control register 1 (CLCR1)

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

Table 322. 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

24.8.7 Commands list control register 2 (CLCR2)

Figure 315. Commands list control register 2 (CLCR2)

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

Table 323. 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

24.8.8 Trigger handler control register 1 (THCR1)

Figure 316. Trigger handler control register 1 (THCR1)

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	T3_E	T3_ ETE	T3_ T1E	T3_ T0E	T3_ ADCE	0	0	0	T2_E	T2_ ETE	T2_ T1E	T2_ T0E	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	0	0	T1_E	T1_ ETE	T1_ T1E	T1_ T0E	T1_ ADCE	0	0	0	T0_E	T0_ ETE	T0_ T1E	T0_ T0E	T0_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 324. THCR1 field descriptions

Field	Description
T3_E	Trigger 3 enable 0 Disabled 1 Enabled
T3_ETE	Trigger 3 External Trigger output enable 0 Disabled 1 Enabled
T3_T1E	Trigger 3 Timer 1 output enable 0 Disabled 1 Enabled
T3_T0E	Trigger 3 Timer 0 output enable 0 Disabled 1 Enabled
T3_ADCE	Trigger 3 ADC command output enable 0 Disabled 1 Enabled
T2_E	Trigger 2 enable 0 Disabled 1 Enabled
T2_ETE	Trigger 2 External Trigger output enable 0 Disabled 1 Enabled
T2_T1E	Trigger 2 Timer 1 output enable 0 Disabled 1 Enabled
T2_T0E	Trigger 2 Timer 0 output enable 0 Disabled 1 Enabled

Table 324. THCR1 field descriptions (continued)

Field	Description
T2_ADCE	Trigger 2 ADC command output enable 0 Disabled 1 Enabled
T1_E	Trigger 1 enable 0 Disabled 1 Enabled
T1_ETE	Trigger 1 External Trigger output enable 0 Disabled 1 Enabled
T1_T1E	Trigger 1 Timer 1 output enable 0 Disabled 1 Enabled
T1_T0E	Trigger 1 Timer 0 output enable 0 Disabled 1 Enabled
T1_ADCE	Trigger 1 ADC command output enable 0 Disabled 1 Enabled
T0_E	Trigger 0 enable 0 Disabled 1 Enabled
T0_ETE	Trigger 0 External Trigger output enable 0 Disabled 1 Enabled
T0_T1E	Trigger 0 Timer 1 output enable 0 Disabled 1 Enabled
T0_T0E	Trigger 0 Timer 0 output enable 0 Disabled 1 Enabled
T0_ADCE	Trigger 0 ADC command output enable 0 Disabled 1 Enabled

24.8.9 Trigger handler control register 2 (THCR2)

Figure 317. Trigger handler control register 2 (THCR2)

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	0	0	T7_E	T7_ ETE	T7_ T1E	T7_ T0E	T7_ ADCE	0	0	0	T6_E	T6_ ETE	T6_ T1E	T6_ T0E	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	0	0	T5_E	T5_ ETE	T5_ T5E	T5_ T0E	T5_ ADCE	0	0	0	T4_E	T4_ ETE	T4_ T1E	T4_ T0E	T4_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 325. 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_T1E	Trigger 7 Timer 1 output enable 0 Disabled 1 Enabled
T7_T0E	Trigger 7 Timer 0 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_T1E	Trigger 6 Timer 1 output enable 0 Disabled 1 Enabled
T6_T0E	Trigger 6 Timer 0 output enable 0 Disabled 1 Enabled

Table 325. THCR2 field descriptions (continued)

Field	Description
T6_ADCE	Trigger 6 ADC command output enable 0 Disabled 1 Enabled
T5_E	Trigger 5 enable 0 Disabled 1 Enabled
T5_ETE	Trigger 5 External Trigger output enable 0 Disabled 1 Enabled
T5_T1E	Trigger 5 Timer 1 output enable 0 Disabled 1 Enabled
T5_T0E	Trigger 5 Timer 0 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_T1E	Trigger 4 Timer 1 output enable 0 Disabled 1 Enabled
T4_T0E	Trigger 4 Timer 0 output enable 0 Disabled 1 Enabled
T4_ADCE	Trigger 4 ADC command output enable 0 Disabled 1 Enabled

24.8.10 Commands list register x (x = 1,...,24) (CLR*x*)

Figure 318 and *Table 326* show the register configured for ADC command format in single conversion mode (CMS = 0) (CLR*x*).

Figure 318. Commands list register *x* (*x* = 1,...,24) (CMS = 0)

Address: Base + 0x002C,...,0x005A (See *Table 312*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	0	0	0	SU	0	CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 326. CLR*x* (CMS = 0) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection bit 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit 0/1 00 FIFO 0 selected 01 FIFO 1 selected 10 FIFO 2 selected 11 FIFO 3 selected
SU	Selection Unit bit 0 ADC unit 0 selected 1 ADC unit 1 selected
CH	ADC unit channel number

Figure 319 and *Table 327* show the register configured for ADC command format in dual conversion mode (CMS = 1).

Figure 319. Commands list register x (x = 1,...,24) (CMS = 1)

Address: Base + 0x002C ... 0x005A (See *Table 312*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	CH_B				0	CH_A			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 327. CLRx (CMS = 1) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit A/B
CH_B	ADC unit B channel number
CH_A	ADC unit A channel number

24.8.11 FIFO DMA control register (FDCR)

Figure 320. FIFO DMA control register (FDCR)

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	0	0	0	0	0	0	0	0	DE3	DE2	DE1	DE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 328. FDCR field descriptions

Name	Description
DEx	This bit enables DMA for the FIFOx 0 Disabled 1 Enabled

24.8.12 FIFO control register (FCR)

Figure 321. FIFO control register (FCR)

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

Table 329. FCR field descriptions

Field	Description
OR_EN3	FIFO 3 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN3	FIFO 3 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN3	FIFO 3 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN3	FIFO 3 Full interrupt enable 0 Disabled 1 Enabled
OR_EN2	FIFO 2 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN2	FIFO 2 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN2	FIFO 2 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN2	FIFO 2 Full interrupt enable 0 Disabled 1 Enabled
OR_EN1	FIFO 1 Overrun interrupt enable 0 Disabled 1 Enabled

Table 329. FCR field descriptions (continued)

Field	Description
OF_EN1	FIFO 1 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN1	FIFO 1 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN1	FIFO 1 Full interrupt enable 0 Disabled 1 Enabled
OR_EN0	FIFO 0 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN0	FIFO 0 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN0	FIFO 0 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN0	FIFO 0 Full interrupt enable 0 Disabled 1 Enabled

24.8.13 FIFO threshold register (FTH)

Figure 322. FIFO threshold register (FTH)

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

Table 330. FTH field descriptions

Field	Description
TH3	FIFO 3 Threshold
TH2	FIFO 2 Threshold

Table 330. FTH field descriptions (continued)

Field	Description
TH1	FIFO 1 Threshold
TH0	FIFO 0 Threshold

24.8.14 FIFO status register (FST)

Figure 323. FIFO status register (FST)

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	r1c				r1c				r1c				r1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 331. FST field descriptions

Field	Description
OR3	FIFO 3 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF3	FIFO 3 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP3	FIFO 3 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL3	FIFO 3 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR2	FIFO 2 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF2	FIFO 2 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP2	FIFO 2 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 331. FST field descriptions (continued)

Field	Description
FULL2	FIFO 2 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR1	FIFO 1 Overrun interrupt flag A read of this bit clears it. 0Interrupt has not occurred. 1 Interrupt has occurred.
OF1	FIFO 1 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP1	FIFO 1 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL1	FIFO 1 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR0	FIFO 0 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF0	FIFO 0 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP0	FIFO 0 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL0	FIFO 0 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

24.8.15 FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Figure 324. FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Address: Base + 0x0080,...,0x008C 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	N_CH[4:0]				
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	DATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 332. FR*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

24.8.16 FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Figure 325. FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Address: Base + 0x00A0,...,0x00AC 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	N_CH[4:0]				
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	DATA										0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 333. FL*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

24.8.17 Cross triggering unit error flag register (CTUEFR)

Figure 326. Cross triggering unit error flag register (CTUEFR)

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	0	0	0	0	ET_OE	T1_OE	T0_OE	ADC_OE	TGS_OSM	MRS_O	ICE	SM_TO	MRS_RE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 334. CTUEFR field descriptions

Field	Description
ET_OE	External Trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
T1_OE	Timer 1 trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
T0_OE	Timer 0 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 Overrun 0 Error has not occurred. 1 Error has occurred.
ICE	Invalid Command Error 0 Error has not occurred. 1 Error has occurred.
SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode 0 Error has not occurred. 1 Error has occurred.
MRS_RE	Master Reload Signal Reload Error 0 Error has not occurred. 1 Error has occurred.

24.8.18 Cross triggering unit interrupt flag register (CTUIFR)

Figure 327. Cross triggering unit interrupt flag register (CTUIFR)

Address: Base + 0x00C2 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	ADC_I	T7_I	T6_I	T5_I	T4_I	T3_I	T2_I	T1_I	T0_I	MRS_I
W							r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 335. CTUIFR field descriptions

Field	Description
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.
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	MRS Interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

24.8.19 Cross triggering unit interrupt/DMA register (CTUIR)

Figure 328. Cross triggering unit interrupt/DMA register (CTUIR)

Address: Base + 0x00C4 Access: User read-only

	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	0	0	0	MRS_DMAE	MRS_IE	IEE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 336. CTUIR field descriptions

Field	Description
T7_IE	Trigger 7 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T6_IE	Trigger 6 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T5_IE	Trigger 5 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T4_IE	Trigger 4 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T3_IE	Trigger 3 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T2_IE	Trigger 2 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T1_IE	Trigger 1 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T0_IE	Trigger 0 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
MRS_DMAE	DMA transfer Enable on MRS occurrence if GRE bit is set 0 Interrupt disabled 1 Interrupt enabled
MRS_IE	MRS Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
IEE	Interrupt Error Enable 0 Interrupt disabled 1 Interrupt enabled

24.8.20 Control ON time register (COTR)

Figure 329. Control ON time register (COTR)

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

Table 337. COTR field descriptions

Field	Description
COTR	Control ON-Time and Guard Time for external trigger

24.8.21 Cross triggering unit control register (CTUCR)

Figure 330. Cross triggering unit control register (CTUCR)

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	CRU_ADC_R	CTU_ODIS	FE	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

Table 338. CTUCR field descriptions

Field	Description
T7_SG	Trigger 7 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T6_SG	Trigger 6 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T5_SG	Trigger 5 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T4_SG	Trigger 4 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T3_SG	Trigger 3 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T2_SG	Trigger 2 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T1_SG	Trigger 1 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T0_SG	Trigger 0 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
CRU_ADC_R	CTU/ADC state machine Reset
CTU_ODIS	CTU Output Disable

Table 338. CTUCR field descriptions (continued)

Field	Description
DFE	Digital Filter Enable
CGRE	Clear GRE
FGRE	Flag GRE
MRS_SG	MRS Software Generated
GRE	General Reload Enable
TGSISR_RE	TGS Input Selection Register Reload Enable

24.8.22 Cross triggering unit digital filter (CTUDF)

Figure 331. Cross triggering unit digital filter (CTUDF)

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	N							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 339. CTUDF field descriptions

Field	Description
0-7	Reserved
8-15 N	Digital Filter value (the external signal is considered at 1 if it is latched N time at 1 and is considered at 0 if it is latched N time at 0)

24.8.23 Cross triggering unit power control register (CTUPCR)

Figure 332. Cross triggering unit power control register (CTUPCR)

Address: Base + 0x00CC 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	MDIS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 340. CTUPCR field descriptions

Field	Description
0-14	Reserved
15 MDIS	Module Disable

25 FlexPWM

25.1 Overview

The pulse width modulator module (PWM) contains four PWM submodules, each of which capable of controlling a single half-bridge power stage and two fault input 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.

25.2 Features

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- PWM outputs can operate as complimentary 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
- 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.

25.3 Modes of operation

Care must be exercised when using this module in certain device operating modes. Some motors (such 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/HALT and debug modes. PWM outputs will be reactivated (assuming they were active to begin with) when these modes are exited.

Table 341. Modes when PWM operation is restricted

Mode	Description
STOP	Peripheral and CPU clocks are stopped. PWM outputs are driven inactive.
WAIT/HALT	CPU clocks are stopped while peripheral clocks continue to run. PWM outputs are driven inactive as a function of the WAITEN bit.(1)
DEBUG	CPU and peripheral clocks continue to run, but CPU maybe stalled for periods of time. PWM outputs are driven inactive as a function of the DBGEN bit.

1. WAIT mode may be called HALT mode at the SoC level.

25.4 Block diagrams

25.4.1 Module level

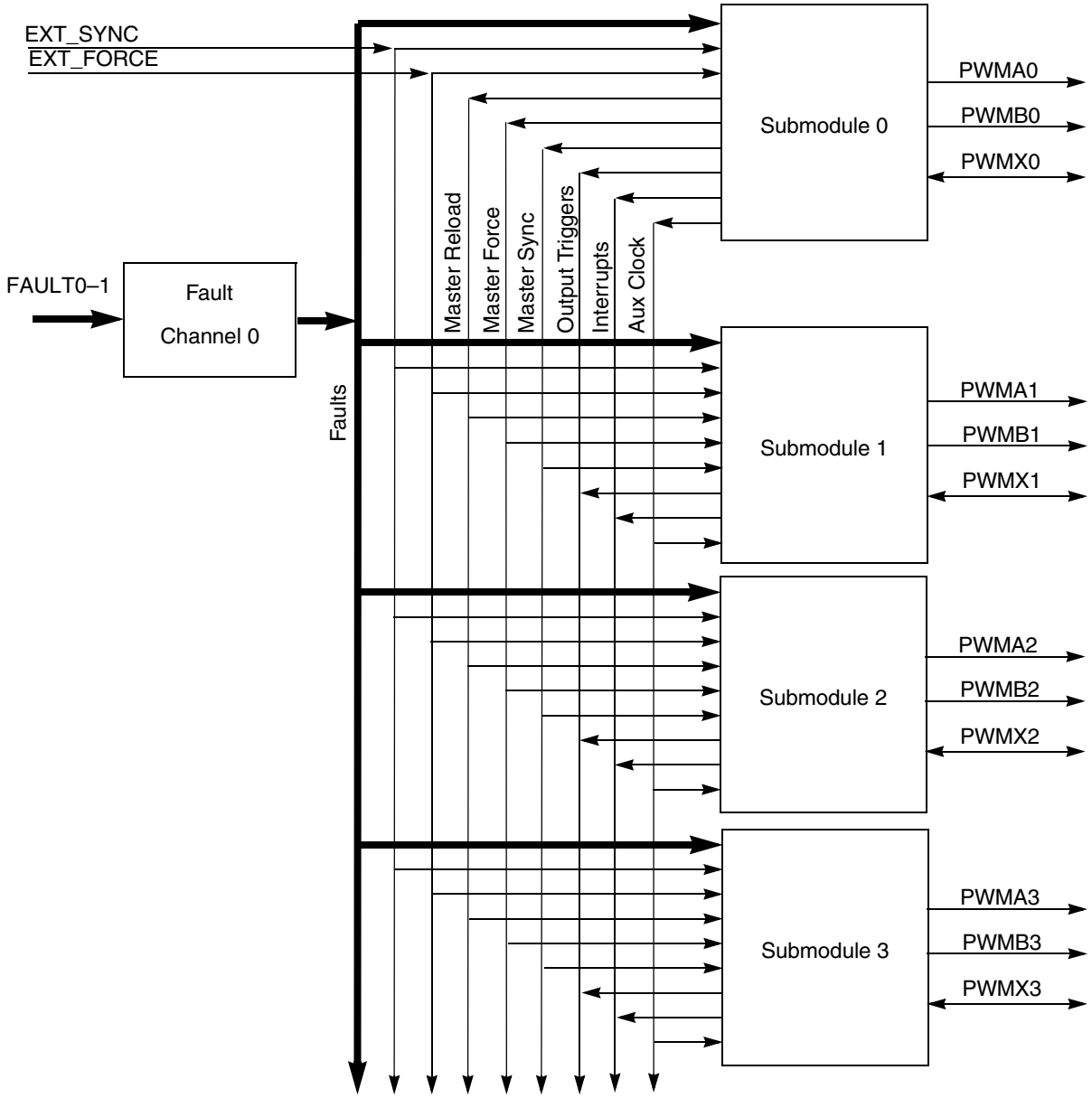


Figure 333. PWM block diagram

25.4.2 PWM submodule

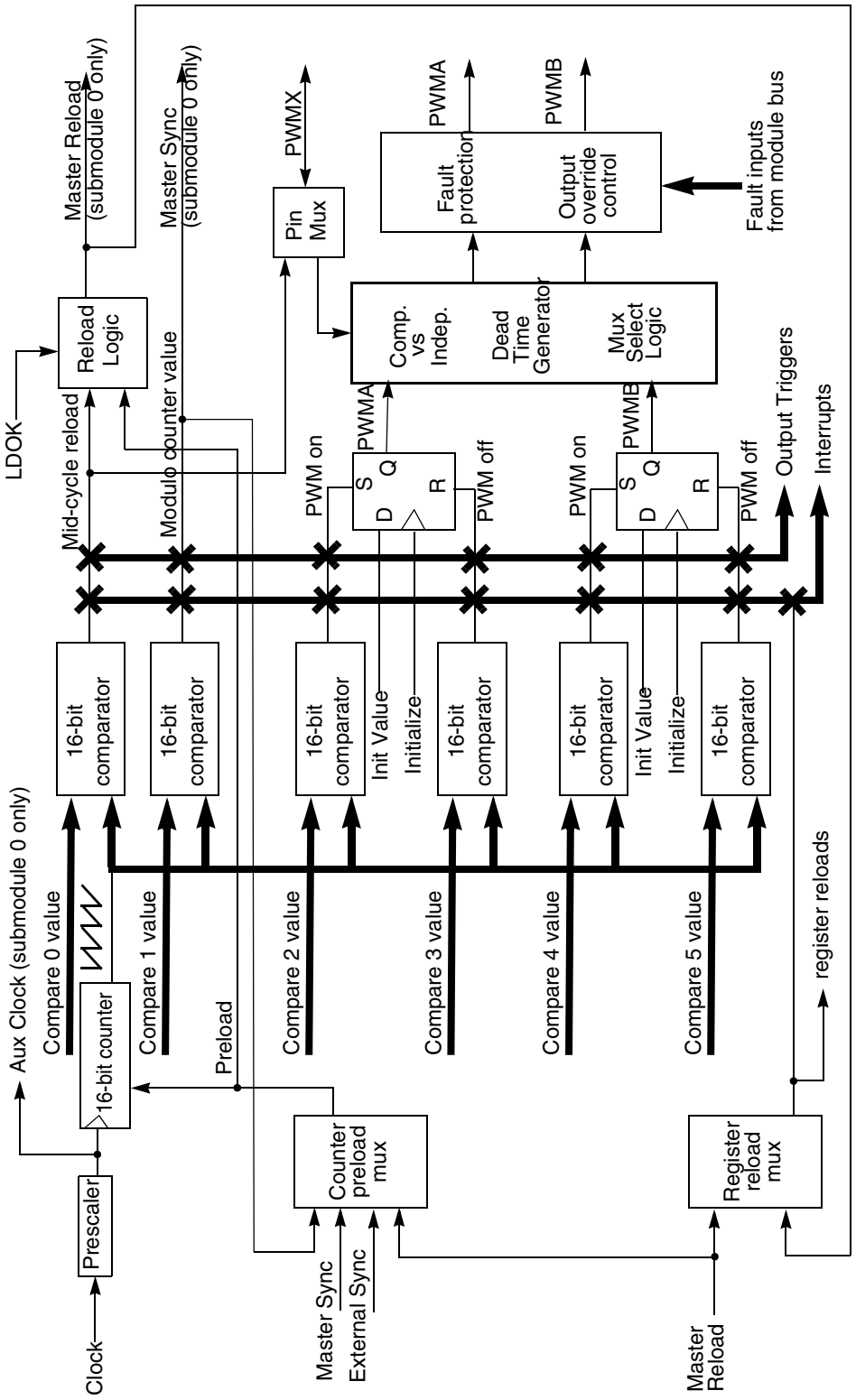


Figure 334. PWM submodule block diagram

25.5 External signal descriptions

The PWM module has external pins named PWMA[*n*], PWMB[*n*], PWMX[*n*], FAULT[*n*], EXT_SYNC. The PWM module also has on-chip inputs called EXT_CLK, EXT_FORCE and on-chip outputs called OUT_TRIG[*n*].

25.5.1 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.

25.5.2 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 to generate the IPOL bit during deadtime correction.

25.5.3 FAULT[*n*] — fault inputs

These are input pins for disabling selected PWM outputs.

25.5.4 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.

25.5.5 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 source of EXT_FORCE signal is eTimer_0 OFLAG.

25.5.6 OUT_TRIG0[n] and OUT_TRIG1[n] — output triggers

These outputs allow the PWM submodules to control timing of the ADC conversions. See *Section , "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.

25.5.7 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.

25.6 Memory map and registers

25.6.1 FlexPWM module memory map

Table 342. FlexPWM memory map

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Access	Reset value	Location
0x0000	CNT—Counter Register (Submodule 0)	R	0x0000	on page 25-651
0x0002	INIT—Initial Count Register (Submodule 0)	R/W	0x0000	on page 25-651
0x0004	CTRL2—Control 2 Register (Submodule 0)	R/W	0x0000	on page 25-652
0x0006	CTRL1—Control 1 Register (Submodule 0)	R/W	0x0000	on page 25-654
0x0008	VAL0—Value Register 0 (Submodule 0)	R/W	0x0000	on page 25-656
0x000A	VAL1—Value Register 1 (Submodule 0)	R/W	0x0000	on page 25-657
0x000C	VAL2—Value Register 2 (Submodule 0)	R/W	0x0000	on page 25-657
0x000E	VAL3—Value Register 3 (Submodule 0)	R/W	0x0000	on page 25-658
0x0010	VAL4—Value Register 4 (Submodule 0)	R/W	0x0000	on page 25-658
0x0012	VAL5—Value Register 5 (Submodule 0)	R/W	0x0000	on page 25-659
0x0014–0x0017	Reserved			
0x0018	OCTRL—Output Control Register (Submodule 0)	R/W	0x0000	on page 25-659
0x001A	STS—Status Register (Submodule 0)	R/W	0x0000	on page 25-660
0x001C	INTEN—Interrupt Enable Register (Submodule 0)	R/W	0x0000	on page 25-661
0x001E	DMAEN—DMA Enable Register (Submodule 0)	R/W	0x0000	on page 25-662
0x0020	TCTRL—Output Trigger Control Register (Submodule 0)	R/W	0x0000	on page 25-663
0x0022	DISMAP—Fault Disable Mapping Register (Submodule 0)	R/W	0xFFFF	on page 25-664
0x0024	DTCNT0—Deadtime Count Register 0 (Submodule 0)	R/W	0x07FF	on page 25-664
0x0026	DTCNT1—Deadtime Count Register 1 (Submodule 0)	R/W	0x07FF	on page 25-664
0x0028–0x004F	Reserved			
0x0050	CNT—Counter Register (Submodule 1)	R	0x0000	on page 25-651
0x0052	INIT—Initial Count Register (Submodule 1)	R/W	0x0000	on page 25-651
0x0054	CTRL2—Control 2 Register (Submodule 1)	R/W	0x0000	on page 25-652
0x0056	CTRL1—Control 1 Register (Submodule 1)	R/W	0x0000	on page 25-654
0x0058	VAL0—Value Register 0 (Submodule 1)	R/W	0x0000	on page 25-656
0x005A	VAL1—Value Register 1 (Submodule 1)	R/W	0x0000	on page 25-657
0x005C	VAL2—Value Register 2 (Submodule 1)	R/W	0x0000	on page 25-657
0x005E	VAL3—Value Register 3 (Submodule 1)	R/W	0x0000	on page 25-658
0x0060	VAL4—Value Register 4 (Submodule 1)	R/W	0x0000	on page 25-658

Table 342. FlexPWM memory map (continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Access	Reset value	Location
0x0062	VAL5—Value Register 5 (Submodule 1)	R/W	0x0000	on page 25-659
0x0064–0x0067	Reserved			
0x0068	OCTRL—Output Control Register (Submodule 1)	R/W	0x0000	on page 25-659
0x006A	STS—Status Register (Submodule 1)	R/W	0x0000	on page 25-660
0x006C	INTEN—Interrupt Enable Register (Submodule 1)	R/W	0x0000	on page 25-661
0x006E	DMAEN—DMA Enable Register (Submodule 1)	R/W	0x0000	on page 25-662
0x0070	TCTRL—Output Trigger Control Register (Submodule 1)	R/W	0x0000	on page 25-663
0x0072	DISMAP—Fault Disable Mapping Register (Submodule 1)	R/W	0xFFFF	on page 25-664
0x0074	DTCNT0—Deadtime Count Register 0 (Submodule 1)	R/W	0x07FF	on page 25-664
0x0076	DTCNT1—Deadtime Count Register 1 (Submodule 1)	R/W	0x07FF	on page 25-664
0x0078–0x009F	Reserved			
0x00A0	CNT—Counter Register (Submodule 2)	R	0x0000	on page 25-651
0x00A2	INIT—Initial Count Register (Submodule 2)	R/W	0x0000	on page 25-651
0x00A4	CTRL2—Control 2 Register (Submodule 2)	R/W	0x0000	on page 25-652
0x00A6	CTRL1—Control 1 Register (Submodule 2)	R/W	0x0000	on page 25-654
0x00A8	VAL0—Value Register 0 (Submodule 2)	R/W	0x0000	on page 25-656
0x00AA	VAL1—Value Register 1 (Submodule 2)	R/W	0x0000	on page 25-657
0x00AC	VAL2—Value Register 2 (Submodule 2)	R/W	0x0000	on page 25-657
0x00AE	VAL3—Value Register 3 (Submodule 2)	R/W	0x0000	on page 25-658
0x00B0	VAL4—Value Register 4 (Submodule 2)	R/W	0x0000	on page 25-658
0x00B2	VAL5—Value Register 5 (Submodule 2)	R/W	0x0000	on page 25-659
0x00B4–0x00B7	Reserved			
0x00B8	OCTRL—Output Control Register (Submodule 2)	R/W	0x0000	on page 25-659
0x00BA	STS—Status Register (Submodule 2)	R/W	0x0000	on page 25-660
0x00BC	INTEN—Interrupt Enable Register (Submodule 2)	R/W	0x0000	on page 25-661
0x00BE	DMAEN—DMA Enable Register (Submodule 2)	R/W	0x0000	on page 25-662
0x00C0	TCTRL—Output Trigger Control Register (Submodule 2)	R/W	0x0000	on page 25-663
0x00C2	DISMAP—Fault Disable Mapping Register (Submodule 2)	R/W	0xFFFF	on page 25-664
0x00C4	DTCNT0—Deadtime Count Register 0 (Submodule 2)	R/W	0x07FF	on page 25-664
0x00C6	DTCNT1—Deadtime Count Register 1 (Submodule 2)	R/W	0x07FF	on page 25-664
0x00C8–0x00EF	Reserved			
0x00F0	CNT—Counter Register (Submodule 3)	R	0x0000	on page 25-651

Table 342. FlexPWM memory map (continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Access	Reset value	Location
0x00F2	INIT—Initial Count Register (Submodule 3)	R/W	0x0000	*on page 25-651*
0x00F4	CTRL2—Control 2 Register (Submodule 3)	R/W	0x0000	*on page 25-652*
0x00F6	CTRL1—Control 1 Register (Submodule 3)	R/W	0x0000	*on page 25-654*
0x00F8	VAL0—Value Register 0 (Submodule 3)	R/W	0x0000	*on page 25-656*
0x00FA	VAL1—Value Register 1 (Submodule 3)	R/W	0x0000	*on page 25-657*
0x00FC	VAL2—Value Register 2 (Submodule 3)	R/W	0x0000	*on page 25-657*
0x00FE	VAL3—Value Register 3 (Submodule 3)	R/W	0x0000	*on page 25-658*
0x0100	VAL4—Value Register 4 (Submodule 3)	R/W	0x0000	*on page 25-658*
0x0102	VAL5—Value Register 5 (Submodule 3)	R/W	0x0000	*on page 25-659*
0x0104–0x0107	Reserved			
0x0108	OCTRL—Output Control Register (Submodule 3)	R/W	0x0000	*on page 25-659*
0x010A	STS—Status Register (Submodule 3)	R/W	0x0000	*on page 25-660*
0x010C	INTEN—Interrupt Enable Register (Submodule 3)	R/W	0x0000	*on page 25-661*
0x010E	DMAEN—DMA Enable Register (Submodule 3)	R/W	0x0000	*on page 25-662*
0x0110	TCTRL—Output Trigger Control Register (Submodule 3)	R/W	0x0000	*on page 25-663*
0x0112	DISMAP—Fault Disable Mapping Register (Submodule 3)	R/W	0xFFFF	*on page 25-664*
0x0114	DTCNT0—Deadtime Count Register 0 (Submodule 3)	R/W	0x07FF	*on page 25-664*
0x0116	DTCNT1—Deadtime Count Register 1 (Submodule 3)	R/W	0x07FF	*on page 25-664*
0x0118–0x013F	Reserved			
0x0140	OUTEN—Output Enable Register	R/W	0x0000	*on page 25-665*
0x142	MASK—Output Mask Register	R/W	0x0000	*on page 25-666*
0x0144	SWCOUT—Software Controlled Output Register	R/W	0x0000	*on page 25-667*
0x0146	DTSRCSEL—Deadtime Source Select Register	R/W	0x0000	*on page 25-668*
0x0148	MCTRL—Master Control Register	R/W	0x0000	*on page 25-670*
0x014A	Reserved			
0x014C	FCTRL—Fault Control Register	R/W	0x0000	*on page 25-671*
0x014E	FSTS—Fault Status Register	R/W	0x0303	*on page 25-672*
0x0150	FFILT—Fault Filter Register	R/W	0x0000	*on page 25-673*
0x0152–0x3FFF	Reserved			

25.6.2 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.

25.6.3 Submodule registers

These registers are repeated for each PWM submodule.

Counter Register (CNT)

Figure 335. Counter Register (CNT)

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

This read-only register displays the state of the signed 16-bit submodule counter. This register is not byte accessible.

Initial Count Register (INIT)

Figure 336. Initial Count Register (INIT)

Address:
Base + 0x0002 (Submodule 0)
Base + 0x0052 (Submodule 1)
Base + 0x00A2 (Submodule 2)
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	INIT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The 16-bit signed value in this register defines the initial count value for the PWM in PWM clock periods. This is the value 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 LDOK bit is set and the next PWM load cycle begins. 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.

Control 2 Register (CTRL2)

Figure 337. Control 2 Register (CTRL2)

Address:
Base + 0x0004 (Submodule 0)
Base + 0x0054 (Submodule 1)
Base + 0x00A4 (Submodule 2)
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	DBGEN	WAITEN	INDEP	PWMA_INIT	PWMB_INIT	PWMX_INIT	INIT_SEL		FRCEN	0	FORCE_SEL			RELOAD_SEL	CLK_SEL	
W										FORCE						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 343. CTRL2 field descriptions

Field	Description
0 DBGEN	Debug Enable When this bit is set, the PWM will continue 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.
1 WAITEN	WAIT Enable When this bit is set, the PWM continues to run while the device is in WAIT/HALT mode. In this mode, the peripheral clock continues to run, but the CPU clock does not. If the device enters WAIT/HALT mode and this bit is cleared, then the PWM outputs are disabled until WAIT/HALT 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/HALT 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/HALT mode. The key point is PWM parameter updates will not occur in this mode. Any motors requiring such updates should be disabled during WAIT/HALT mode. If in doubt, leave this bit cleared.
2 INDEP	Independent or Complementary Pair Operation This bit determines whether the PWMA and PWMB channels will be independent PWMs or a complementary PWM pair. 0 PWMA and PWMB form a complementary PWM pair. 1 PWMA and PWMB outputs are independent PWMs.
3 PWMA_INIT	PWMA 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.

Table 343. CTRL2 field descriptions (continued)

Field	Description
4 PWMB_INIT	PWMB 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.
5 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.
6:7 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 will force 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 will force the INIT signal to logic 0. 11 EXT_SYNC causes initialization.
8 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.
9 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 will assume values based on the SELA and SELB bits. – If the FRCEN bit is set, the counter value will be initialized with the INIT register value.
10:12 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 will hold 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 will hold 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 will hold the FORCE OUTPUT signal to logic 0. 110 The external force signal, EXT_FORCE, from outside the PWM module causes updates. 111 Reserved.

Table 343. CTRL2 field descriptions (continued)

Field	Description
13 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 will be 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 will force the RELOAD signal to logic 0.
14:15 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 will force the clock to logic 0. 11 Reserved.

Control 1 Register (CTRL1)

Figure 338. Control 1 Register (CTRL1)

Address: Base + 0x0006 (Submodule 0)
Base + 0x0056 (Submodule 1)
Base + 0x00A6 (Submodule 2)
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	0	0	DBLEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 344. CTRL1 field descriptions

Field	Description
0:3 LDFQ	Load Frequency These buffered read/write bits select the PWM load frequency according to *Table 345*. 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 345*.
4 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 half way through the PWM cycle. 0 Half-cycle reloads disabled. 1 Half-cycle reloads enabled.

Table 344. CTRL1 field descriptions (continued)

Field	Description
5 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.
6:7 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.
9:11 PRSC	Prescaler These buffered read/write bits select the divide ratio of the PWM clock frequency selected by CLK_SEL as illustrated in *Table 346*
15 DBLEN	Double Switching Enable This read/write bit enables the double switching PWM behavior. 0 Double switching disabled. 1 Double switching enabled.

Table 345. PWM reload frequency

LDFQ	PWM reload frequency
0000	Every PWM opportunity
0001	Every 2 PWM opportunities
0010	Every 3 PWM opportunities
0011	Every 4 PWM opportunities
0100	Every 5 PWM opportunities
0101	Every 6 PWM opportunities
0110	Every 7 PWM opportunities
0111	Every 8 PWM opportunities
1000	Every 9 PWM opportunities
1001	Every 10 PWM opportunities
1010	Every 11 PWM opportunities
1011	Every 12 PWM opportunities
1100	Every 13 PWM opportunities
1101	Every 14 PWM opportunities
1110	Every 15 PWM opportunities
1111	Every 16 PWM opportunities

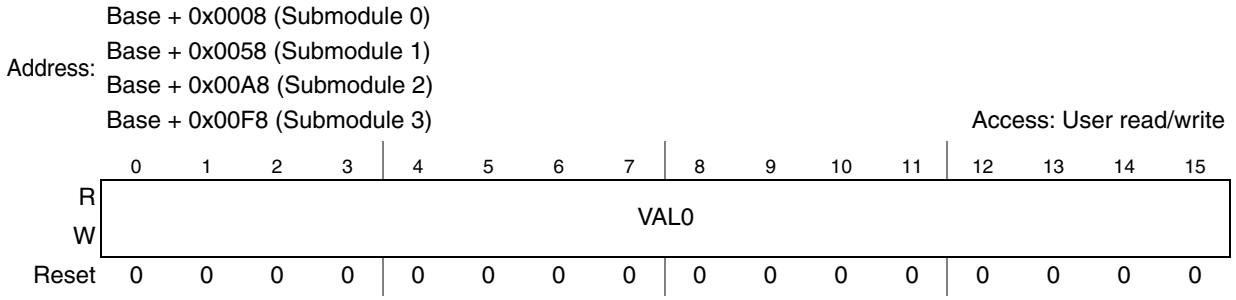
Table 346. PWM prescaler

PRSC	PWM clock frequency
000	f_{clk}
001	$f_{clk}/2$
010	$f_{clk}/4$
011	$f_{clk}/8$
100	$f_{clk}/16$
101	$f_{clk}/32$
110	$f_{clk}/64$
111	$f_{clk}/128$

Note: Reading the PRSCx bits reads the buffered values and not necessarily the values currently in effect. The PRSCx bits take effect at the beginning of the next PWM cycle and only when the load okay bit, LDOK, is set. This field cannot be written when LDOK is set.

Value register 0 (VAL0)

Figure 339. Value Register 0 (VAL0)

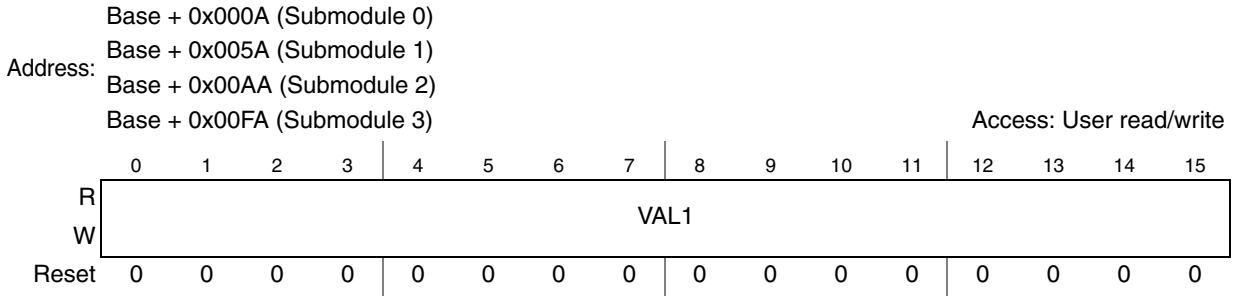


The 16-bit signed value in this buffered, read/write register defines the mid-cycle reload point for the PWM in PWM clock periods. This register is not byte accessible.

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. 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.

Value register 1 (VAL1)

Figure 340. Value Register 1 (VAL1)

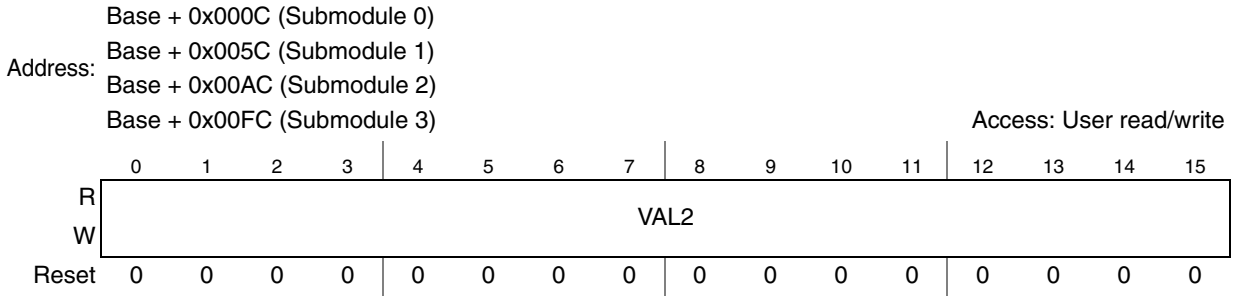


The 16-bit signed value written to this register defines the modulo count value (maximum count) for the submodule counter. Upon reaching this count value, the counter will reload itself with the contents of the INIT register. This register is not byte accessible.

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. 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.

Value register 2 (VAL2)

Figure 341. Value register 2 (VAL2)



The 16-bit signed value in this register defines the count value to set PWMA high (*Figure 334*). This register is not byte accessible.

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. 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.

Value register 3 (VAL3)

Figure 342. 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

The 16-bit signed value in this register defines the count value to set PWMA low (*Figure 334*). This register is not byte accessible.

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. 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.

Value register 4 (VAL4)

Figure 343. 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

The 16-bit signed value in this register defines the count value to set PWMB high (*Figure 334*). This register is not byte accessible.

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. 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.

Value register 5 (VAL5)

Figure 344. Value register 5 (VAL5)

Address:
Base + 0x0012 (Submodule 0)
Base + 0x0062 (Submodule 1)
Base + 0x00B2 (Submodule 2)
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

The 16-bit signed value in this register defines the count value to set PWMB low (*Figure 334*). This register is not byte accessible.

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. 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.

Output Control register (OCTRL)

Figure 345. Output Control register (OCTRL)

Address:
Base + 0x0018 (Submodule 0)
Base + 0x0068 (Submodule 1)
Base + 0x00B8 (Submodule 2)
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

Table 347. OCTRL field descriptions

Field	Description
0 PWMA_IN	PWMA Input This read only bit shows the logic value currently being driven into the PWMA input.
1 PWMB_IN	PWMB Input This read only bit shows the logic value currently being driven into the PWMB input.
2 PWMX_IN	PWMX Input This read only bit shows the logic value currently being driven into the PWMX input.
5 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.

Table 347. OCTRL field descriptions (continued)

Field	Description
6 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.
7 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.
10:11 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/HALT 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. 1x Output is tristated.
12:13 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/HALT 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. 1x Output is tristated.
14:15 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/HALT 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. 1x Output is tristated.

Status register (STS)

Figure 346. Status register (STS)

Address: Base + 0x001A (Submodule 0)
Base + 0x006A (Submodule 1)
Base + 0x00BA (Submodule 2)
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	0	0	CMPF					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 348. STS field descriptions

Field	Description
1 RUF	Registers Updated Flag This read only flag is set when one of the INIT, VALx, 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.
2 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 one to the REF bit. Reset clears REF. 0 No reload error occurred. 1 Reload signal occurred with non-coherent data and LDOK = 0.
3 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 one 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.
10:15 CMPF	Compare Flags These bits are set when the submodule counter value matches the value of one of the VALx registers. Clear these bits by writing a 1 to a bit position. 0 No compare event has occurred for a particular VALx value. 1 A compare event has occurred for a particular VALx value.

Interrupt Enable register (INTEN)

Figure 347. Interrupt Enable register (INTEN)

Address: Base + 0x001C (Submodule 0)
Base + 0x006C (Submodule 1)
Base + 0x00BC (Submodule 2)
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	0	0	CMPIE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 349. INTEN field descriptions

Field	Description
2 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.
3 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.
10:15 CMPIE	Compare Interrupt Enables These bits enable the CMPF flags to cause a compare interrupt request to the CPU. 0 The corresponding CMPF bit will not cause an interrupt request 1 The corresponding CMPF bit will cause an interrupt request.

DMA Enable register (DMAEN)

Figure 348. DMA Enable register (DMAEN)

Address: Base + 0x001E (Submodule 0)
Base + 0x006E (Submodule 1)
Base + 0x00BE (Submodule 2)
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	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

Table 350. DMAEN field descriptions

Field	Description
6 VALDE	Value Registers DMA Enable This read/write bit enables DMA write requests for the VALx registers when RF is set. Reset clears VALDE. 0 DMA write requests disabled. 1 DMA write requests for the VALx registers enabled.
7 FAND	FIFO Watermark AND Control This read/write 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.

Output Trigger Control register (TCTRL)

Figure 349. Output Trigger Control register (TCTRL)

Address: Base + 0x0020 (Submodule 0)
Base + 0x0070 (Submodule 1)
Base + 0x00C0 (Submodule 2)
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

Table 351. TCTRL field descriptions

Field	Description
10:15 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 and 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 VALx 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_TRIGx will not set when the counter value matches the VALx value. 1 OUT_TRIGx will set when the counter value matches the VALx value.

Fault Disable Mapping register (DISMAP)

This register determines which PWM pins are disabled by the fault protection inputs, illustrated in *Table 350* in *Section 25.8.12, "Fault protection*. Reset sets all of the bits in the fault disable mapping register.

Figure 350. Fault Disable Mapping register (DISMAP)

Address: Base + 0x0022 (Submodule 0)
Base + 0x0072 (Submodule 1)
Base + 0x00C2 (Submodule 2)
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

Table 352. DISMAP field descriptions

Field	Description
4:7 DISX	PWMX Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMX output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISX field. A reset sets all DISX bits. Note: DISX[3:2] not used on SPC560P40/34.
8:11 DISB	PWMB Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMB output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISB field. A reset sets all DISB bits. Note: DISB[3:2] not used on SPC560P40/34.
12:15 DISA	PWMA Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMA output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISA field. A reset sets all DISA bits. Note: DISA[3:2] not used on SPC560P40/34.

Deadtime Count registers (DTCNT0, DTCNT1)

Deadtime operation is only applicable to complementary channel operation. Reset sets the deadtime count registers to a default value of 0x07FF, selecting a deadtime of 4095 peripheral clock cycles. These registers are not byte accessible.

Figure 351. Deadtime Count Register 0 (DTCNT0)

Address: Base + 0x0024 (Submodule 0)
Base + 0x0074 (Submodule 1)
Base + 0x00C4 (Submodule 2)
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 352. Deadtime Count register 1 (DTCNT1)

Address: Base + 0x0026 (Submodule 0)
Base + 0x0076 (Submodule 1)
Base + 0x00C6 (Submodule 2)
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

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.

25.6.4 Configuration registers

The base address of the configuration registers is equal to the base address of the PWM plus as offset of 0x140.

Output Enable register (OUTEN)

Figure 353. 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

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 353. OUTEN field descriptions

Field	Description
4:7 PWMA_EN[3:0]	PWMA Output Enables These bits enable the PWMA outputs of each submodule. 0 PWMA output disabled. 1 PWMA output enabled.
8:11 PWMB_EN[3:0]	PWMB Output Enables These bits enable the PWMB outputs of each submodule. 0 PWMB output disabled. 1 PWMB output enabled.
12:15 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 deadtime correction. 0 PWMX output disabled. 1 PWMX output enabled.

Mask register (MASK)

Figure 354. 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

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 354. MASK field descriptions

Field	Description
4:7 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.

Table 354. MASK field descriptions (continued)

Field	Description
8:11 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.
12:15 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 MASKx bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 373 to see how FORCE_OUT is generated. Reading the MASK bits reads the buffered value and not necessarily the value currently in effect.

Software Controlled Output Register (SWCOUT)

Figure 355. 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	OUTA_3	OUTB_3	OUTA_2	OUTB_2	OUTA_1	OUTB_1	OUTA_0	OUTB_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 355. SWCOUT field descriptions

Field	Description
8 OUTA_3	Software Controlled Output A_3 This bit is only used when SELA for submodule 3 is set to 0b10. It allows software control of 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 PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWMA.
9 OUTB_3	Software Controlled Output B_3 This bit is only used when SELB for submodule 3 is set to 0b10. It allows software control of 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 PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWMB.
10 OUTA_2	Software Controlled Output A_2 This bit is only used when SELA for submodule 2 is set to 0b10. It allows software control of 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 PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWMA.

Table 355. SWCOUT field descriptions (continued)

Field	Description
11 OUTB_2	Software Controlled Output B_2 This bit is only used when SELB for submodule 2 is set to 0b10. It allows software control of 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 PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWMB.
12 OUTA_1	Software Controlled Output A_1 This bit is only used when SELA for submodule 1 is set to 0b10. It allows software control of 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 PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWMA.
13 OUTB_1	Software Controlled Output B_1 This bit is only used when SELB for submodule 1 is set to 0b10. It allows software control of 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 PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWMB.
14 OUTA_0	Software Controlled Output A_0 This bit is only used when SELA for submodule 0 is set to 0b10. It allows software control of 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 PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWMA.
15 OUTB_0	Software Controlled Output BA_0 This bit is only used when SELB for submodule 0 is set to 0b10. It allows software control of 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 PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWMB.

Note: These bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 373 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

Deadtime Source Select Register (DTSRCSEL)

Figure 356. 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	SELA_3		SELB_3		SELA_2		SELB_2		SELA_1		SELB_1		SELA_0		SELB_0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 356. DTSRCSEL field descriptions

Field	Description
1:0 SELA_3	PWMA_3 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_3 signal is used by the deadtime logic. 01 Inverted generated PWMA_3 signal is used by the deadtime logic. 10 OUTA_3 bit is used by the deadtime logic. 11 Reserved
2:3 SELB_3	PWMB_3 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_3 signal is used by the deadtime logic. 01 Inverted generated PWMB_3 signal is used by the deadtime logic. 10 OUTB_3 bit is used by the deadtime logic. 11 Reserved
4:5 SELA_2	PWMA_2 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_2 signal is used by the deadtime logic. 01 Inverted generated PWMA_2 signal is used by the deadtime logic. 10 OUTA_2 bit is used by the deadtime logic. 11 Reserved
6:7 SELB_2	PWMB_2 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_2 signal is used by the deadtime logic. 01 Inverted generated PWMB_2 signal is used by the deadtime logic. 10 OUTB_2 bit is used by the deadtime logic. 11 Reserved
8:9 SELA_1	PWMA_1 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_1 signal is used by the deadtime logic. 01 Inverted generated PWMA_1 signal is used by the deadtime logic. 10 OUTA_1 bit is used by the deadtime logic. 11 Reserved
10:11 SELB_1	PWMB_1 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_1 signal is used by the deadtime logic. 01 Inverted generated PWMB_1 signal is used by the deadtime logic. 10 OUTB_1 bit is used by the deadtime logic. 11 Reserved

Table 356. DTSRCSEL field descriptions (continued)

Field	Description
12:13 SELA_0	PWMA_0 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_0 signal is used by the deadtime logic. 01 Inverted generated PWMA_0 signal is used by the deadtime logic. 10 OUTA_0 bit is used by the deadtime logic. 11 Reserved
14:15 SELB_0	PWMB_0 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_0 signal is used by the deadtime logic. 01 Inverted generated PWMB_0 signal is used by the deadtime logic. 10 OUTB_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 373 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

Master Control Register (MCTRL)

Figure 357. 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	IPOL				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

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 357. MCTRL field descriptions

Field	Description
0:3 IPOL[3:0]	Current Polarity This buffered read/write bit selects between PWMA and PWMB as the source for the generation of the complementary PWM pair output. IPOL is ignored in independent mode. PWMB (*Figure 334*) generates complementary PWM pairs. PWMA (*Figure 334*) generates complementary PWM pairs. The IPOL bit does not take effect until a FORCE_OUT event takes place in the appropriate submodule. Reading the IPOL bit reads the buffered value and not necessarily the value currently in effect.
4:7 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. For proper initialization of the LDOK and RUN bits, see *Section 25.9.5, "Initialization.*
8:11 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 will not be performed and LDOK will be cleared. This bit is self clearing and always reads as a 0.
12:15 LDOK[3:0]	Load Okay This read/set bit loads the PRSC bits of CTRL1 and the INIT, and VALx 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 zero and then writing a logic one to it. The VALx, 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, VALx, fields. Reset clears LDOK. 0 Do not load new values. 1 Load prescaler, modulus, and PWM values. For proper initialization of the LDOK and RUN bits, see *Section 25.9.5, "Initialization.*

25.6.5 Fault channel registers

Fault Control Register (FCTRL)

Figure 358. 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

Table 358. FCTRL field descriptions

Field	Description
0:3 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.
4:7 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 FFLAGx 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 FFPINx bit is clear at the start of a half cycle without regard to the state of FFLAGx bit.
8:11 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 FFLAGx bit is clear at the start of a half cycle without regard to the state of the FFPINx bit. The PWM outputs disabled by this fault input will not be re-enabled until the actual FAULTx input signal de-asserts since the fault input will combinationally disable the PWM outputs (as programmed in DISMAP). 1 Safe mode. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear and the FFPINx bit is clear at the start of a half cycle. The FFPINx bit may indicate a fault condition still exists even though the actual fault signal at the FAULTx pin is clear due to the fault filter latency.
12:15 FIE	Fault Interrupt Enables This read/write bit enables CPU interrupt requests generated by the FAULTx pins. A reset clears FIE. 0 FAULTx CPU interrupt requests disabled. 1 FAULTx CPU interrupt requests enabled. 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.

Fault Status Register (FSTS)

Figure 359. 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	1	1	0	0	0	0	0	0	1	1

Table 359. FSTS field descriptions

Field	Description
3 FTEST	Fault Test These read/write bits simulate a fault condition. Setting this bit will cause a simulated fault to be sent into all of the fault filters. The condition will propagate 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.
4:7 FFPIN	Filtered Fault Pins These read-only bits reflect the current state of the filtered FAULTx pins converted to high polarity. A logic 1 indicates a fault condition exists on the filtered FAULTx pin. A reset has no effect on FFPIN.
12:15 FFLAG	Fault Flags These read-only flags are set within 2 CPU cycles after a transition to active on the FAULTx pin. Clear FFLAGx by writing a logic one to it. A reset clears FFLAG. 0 No fault on the FAULTx pin. 1 Fault on the FAULTx pin. The FFLAG[3:0] flags will be set out of reset. They should be cleared before enabling the Fault Control feature.

Fault Filter Register (FFILT)

Figure 360. 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	0	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

The settings in this register are shared among each of the fault input filters.

Table 360. FFILT field descriptions

Field	Description
5:7 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 , "Input filter considerations.*
8:15 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 , "Input filter considerations.*

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 will only corrupt 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) x FILT_PER x 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 will be seen in the time to set the FFLAG and FFPIN bits of the FSTS register.

25.7 Functional description

25.7.1 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 361*.

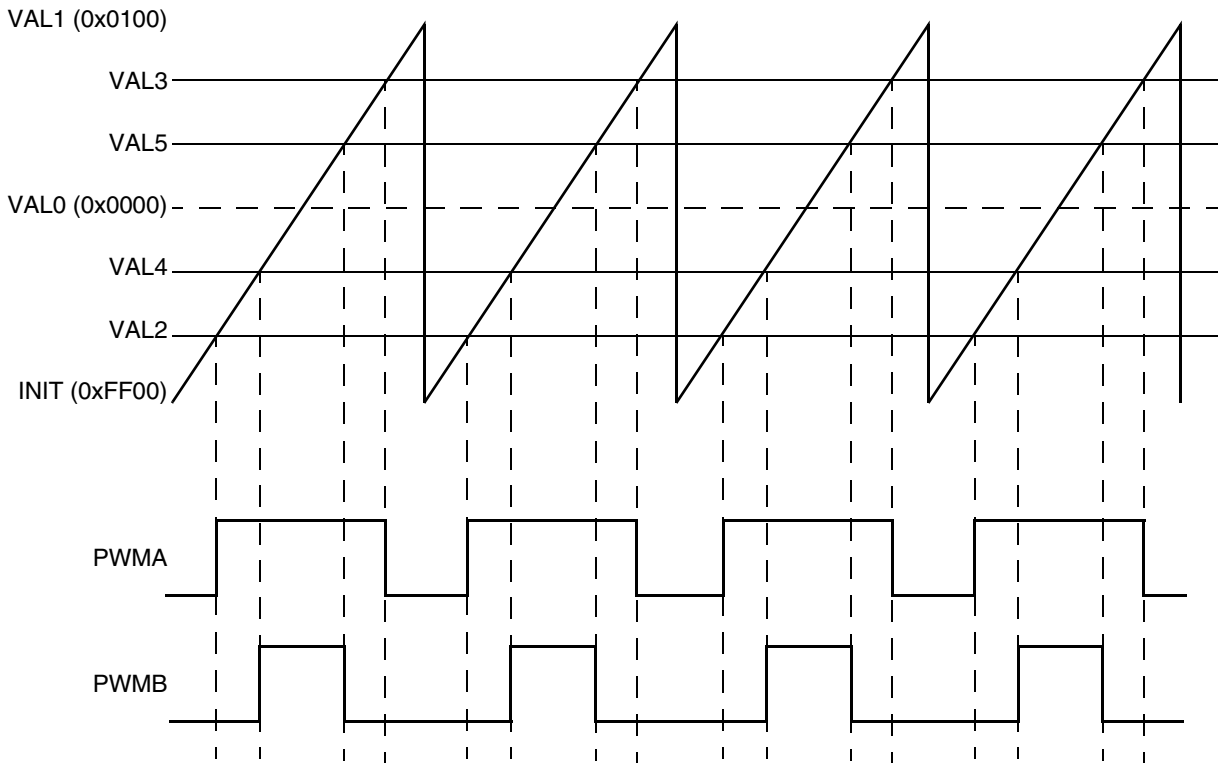


Figure 361. Center-aligned example

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 361 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 happens to be 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 will be 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 will result 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 will be 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.

25.7.2 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 362*. Therefore, only the turn off edge value needs to be periodically updated to change the pulse width.

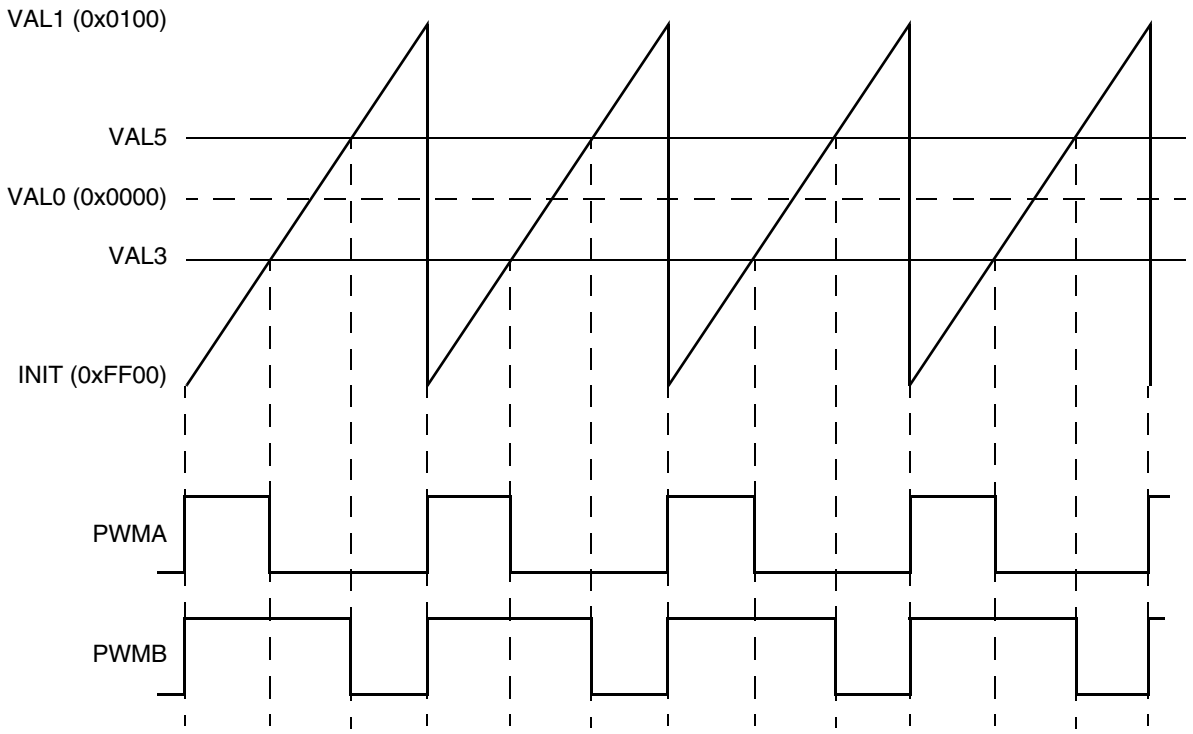


Figure 362. Edge-aligned example (INIT = VAL2 = VAL4)

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 0 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.

25.7.3 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 will be phase shifted with respect to each other, as illustrated in *Figure 363*. 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.

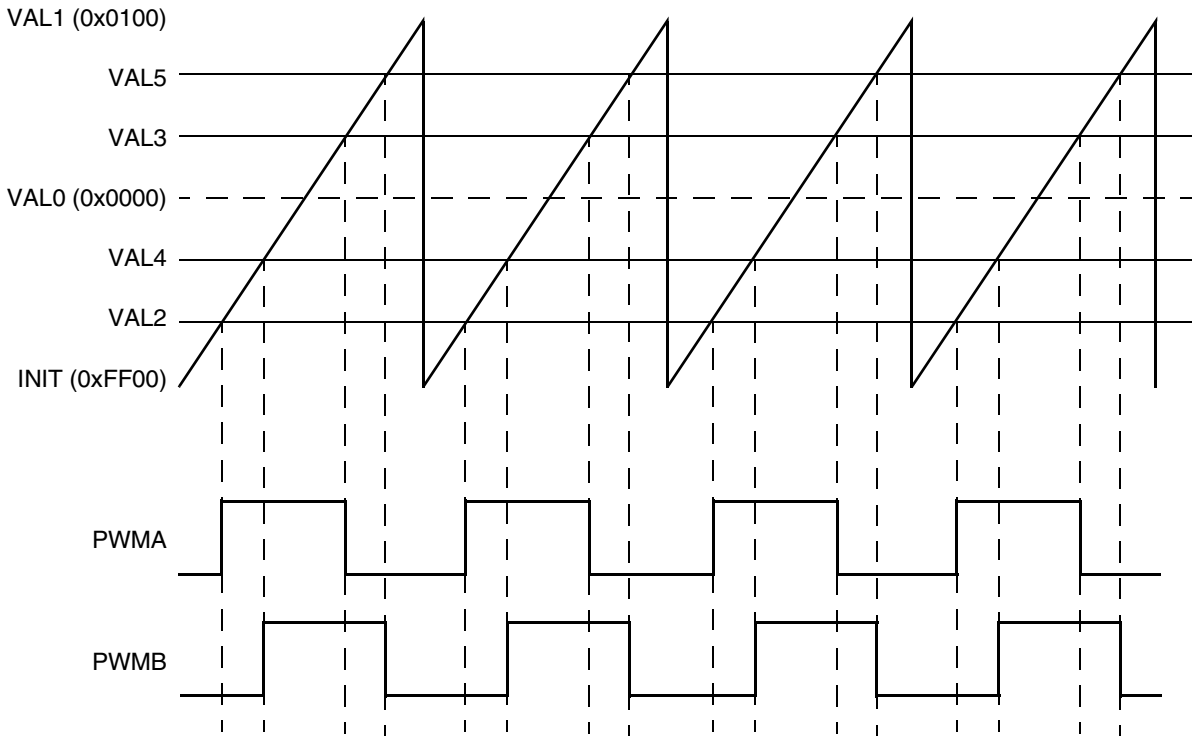


Figure 363. Phase-shifted outputs example

An additional benefit of phase-shifted PWMs can be seen in *Figure 364*. In this case, an H-bridge circuit is driven by four 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 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.

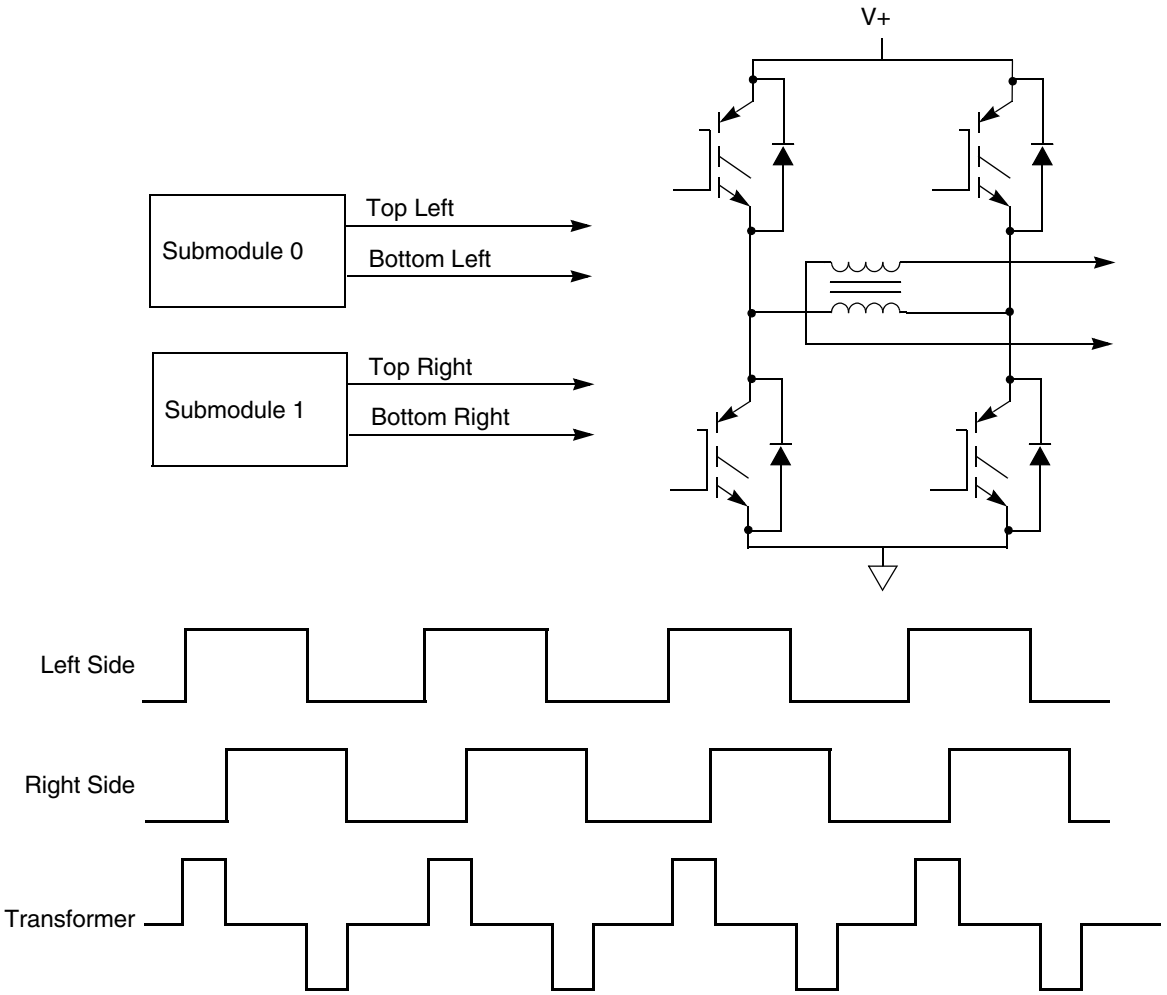


Figure 364. Phase-shifted PWMs applied to a transformer primary

25.7.4 Double switching PWMs

Double switching PWM output is supported to aid in single shunt current measurement and three phase reconstruction. This method support two independent rising edges and two independent falling edges per PWM cycle. The VAL2 and VAL3 registers generate the even channel (labeled as PWMA in the figure) while VAL4 and VAL5 generate the odd channel. The two channels are combined using XOR logic (see *Figure 373*) as shown in *Figure 365*. The DBLPWM signal can be run through the deadtime insertion logic.

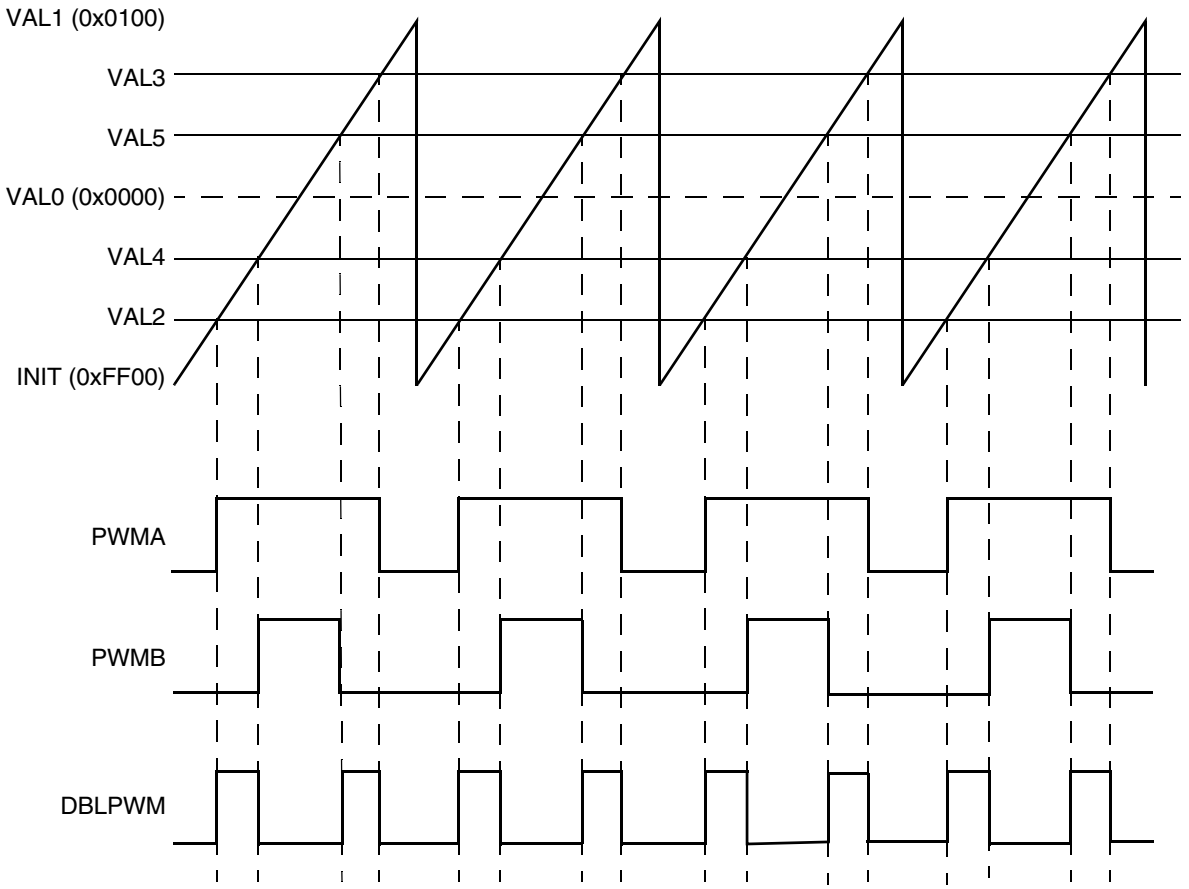


Figure 365. Double switching output example

25.7.5 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 366* shows how this is accomplished. When specifying complimentary 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 does not need to respond quickly after the first conversion to set up other conversions that must occur in the same PWM cycle.

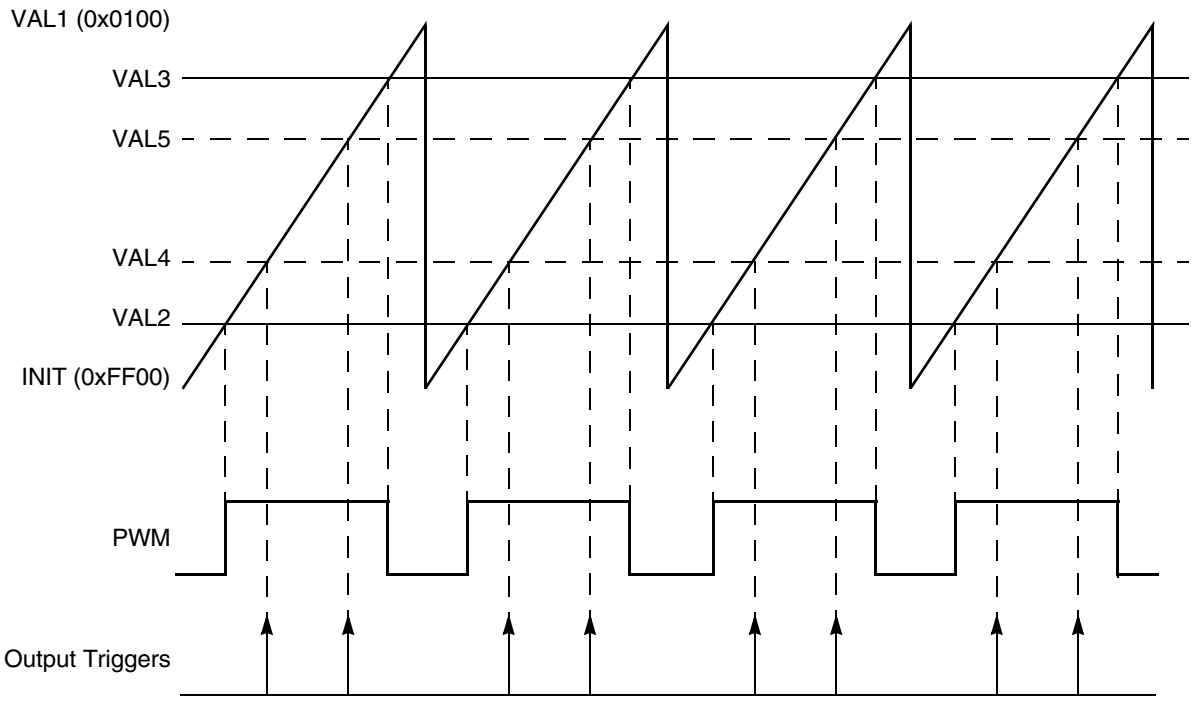


Figure 366. Multiple output trigger generation in hardware

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 submodule 0. *Figure 367* 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 367*, ALL submodule comparators are shown being used for ADC trigger generation.

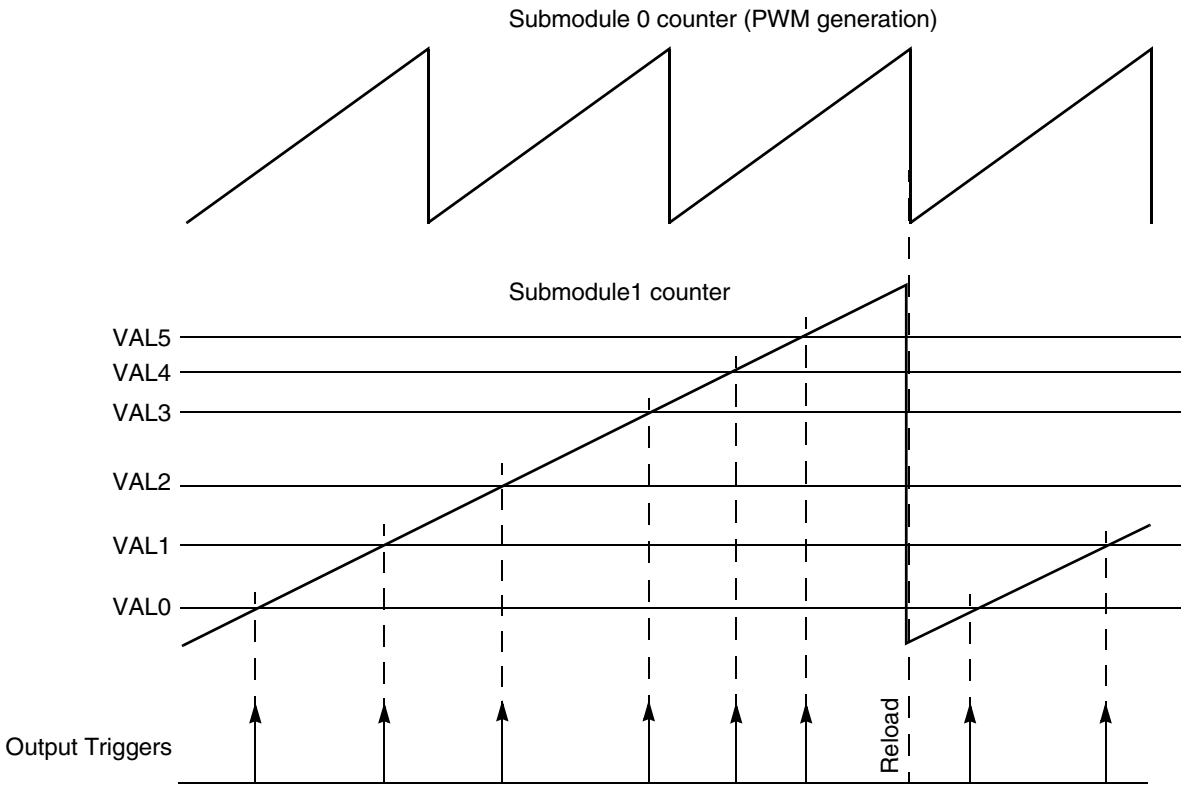


Figure 367. Multiple output triggers over several PWM cycles

25.7.6 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 submodule 0, 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. 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 368 shows a popular application that can benefit from this feature. On a brushless DC motor it is desirable on 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 368* 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.

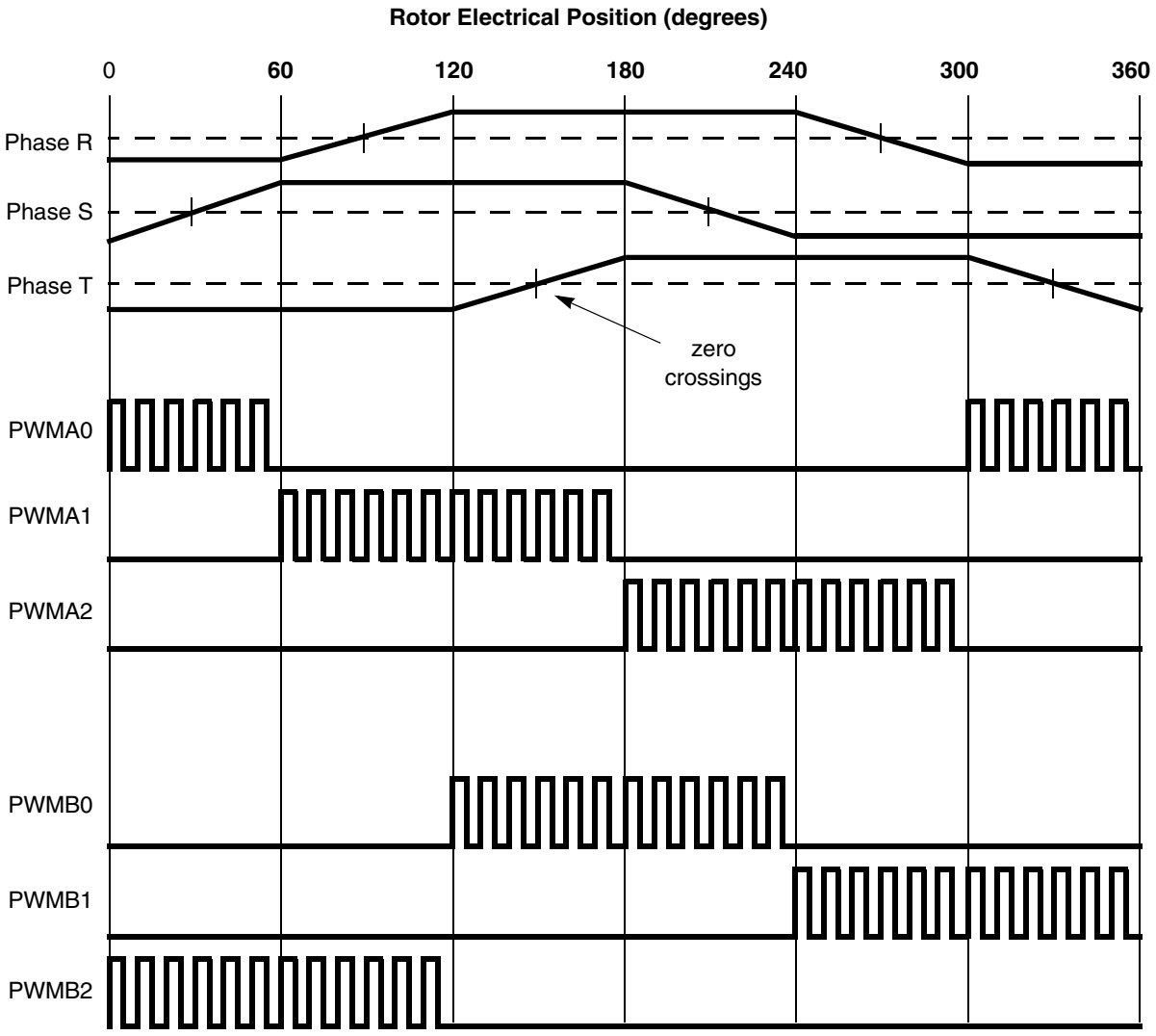


Figure 368. Sensorless BLDC commutation using the force out function

25.8 Functional details

This section describes the implementation of various features of the PWM in greater detail.

25.8.1 PWM clocking

Figure 369 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 submodule 0 and can be selected as the clock source by other submodules so that the 8-bit prescaler and RUN bit from submodule 0 can control all of the submodules. When AUX_CLK is selected as the source for the submodule clock, the RUN bit from submodule 0 is used instead of the local RUN bit from this submodule.

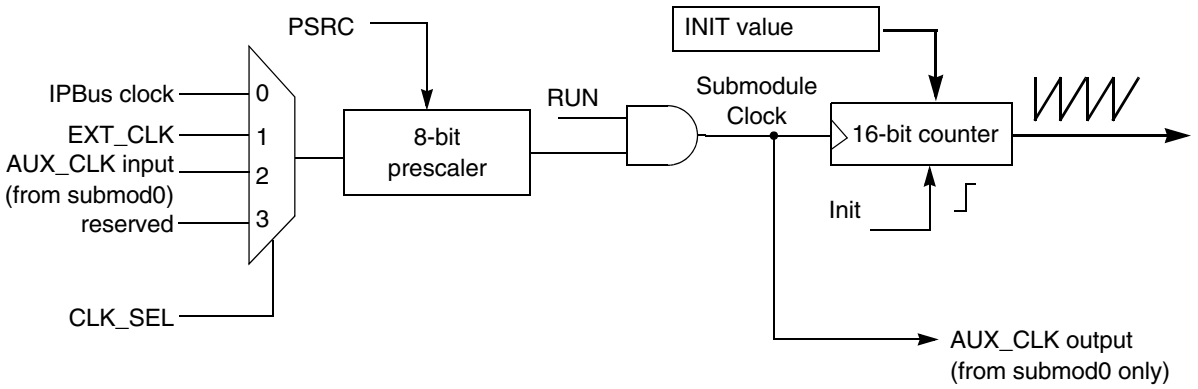


Figure 369. Clocking block diagram for each PWM submodule

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 will not be used by the PWM generator until the LDOK bit is set and a new PWM reload cycle begins.

25.8.2 Register reload logic

The register reload logic determines when the outer set of registers for all double buffered register pairs will be 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 370* the reload signal from submodule 0 can be broadcast as the Master Reload signal allowing the reload logic from submodule 0 to control the reload of registers in other submodules.

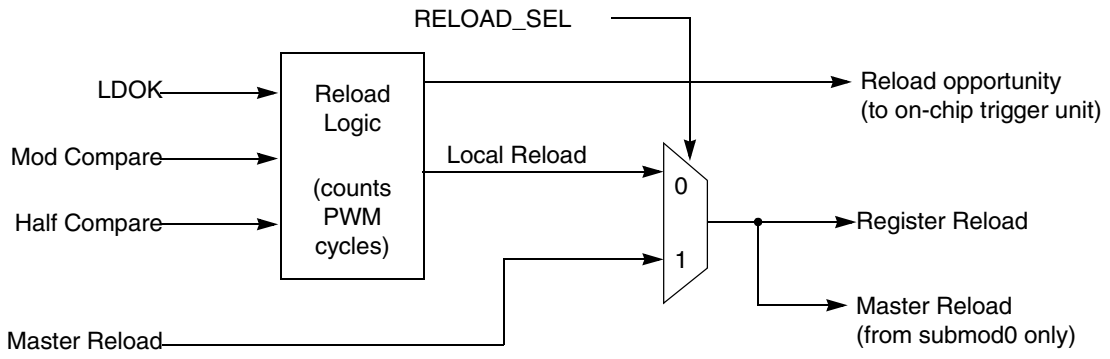


Figure 370. Register reload logic

25.8.3 Counter synchronization

Referring to *Figure 371*, the 16-bit counter will count 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.

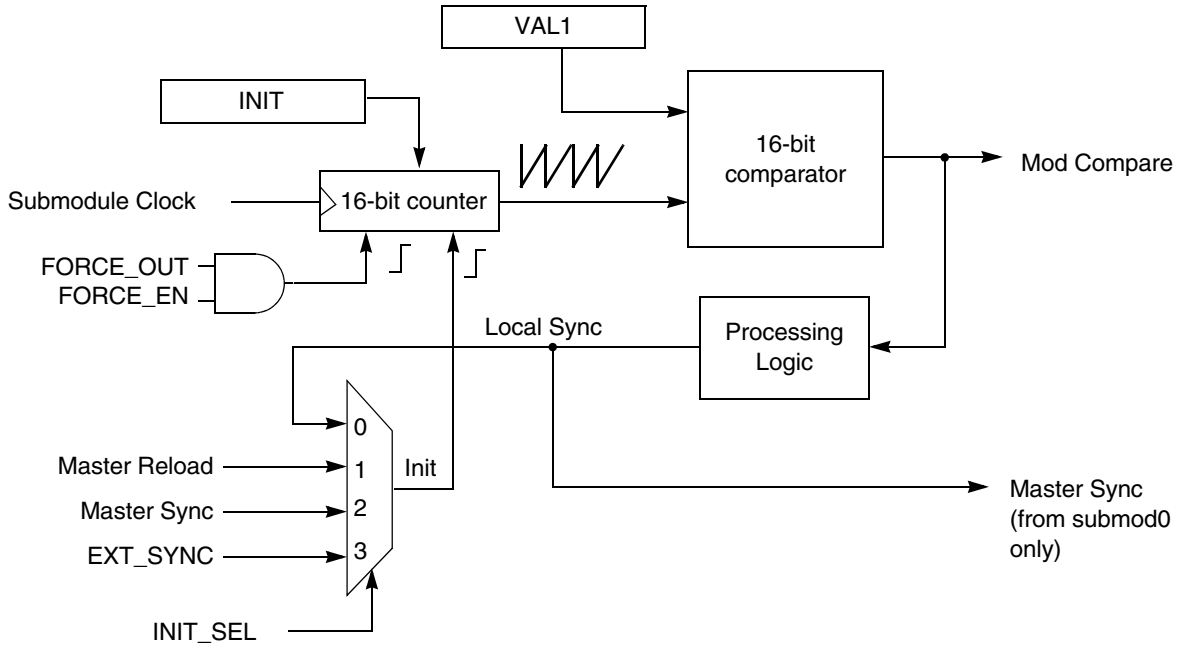


Figure 371. Submodule timer synchronization

The Master Sync signal originates as the Local Sync from submodule 0. If configured to do so, the timer period of any submodule can be locked to the period of the timer in submodule 0. The VAL1 register and associated comparator of the other submodules can then be freed up for other functions such as PWM generation, output compares, or output triggers.

The EXT_SYNC signal originates either on- 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 will be locked to the register reload frequency of submodule 0. Since the reload frequency is usually commensurate to the sampling frequency of the software control algorithm, the submodule counter period will therefore equal 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 submodule 0.

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 371*, 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 will be 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 will be an oscillation caused by the beating between the PWM frequency and the commutation frequency.

25.8.4 PWM generation

Figure 372 illustrates how PWM generation is accomplished in each submodule. In each case, two comparators and associated VALx registers are utilized for each PWM output signal. One comparator and VAL*x* register control the turn-on edge, while a second comparator and VAL*y* register control the turn-off edge.

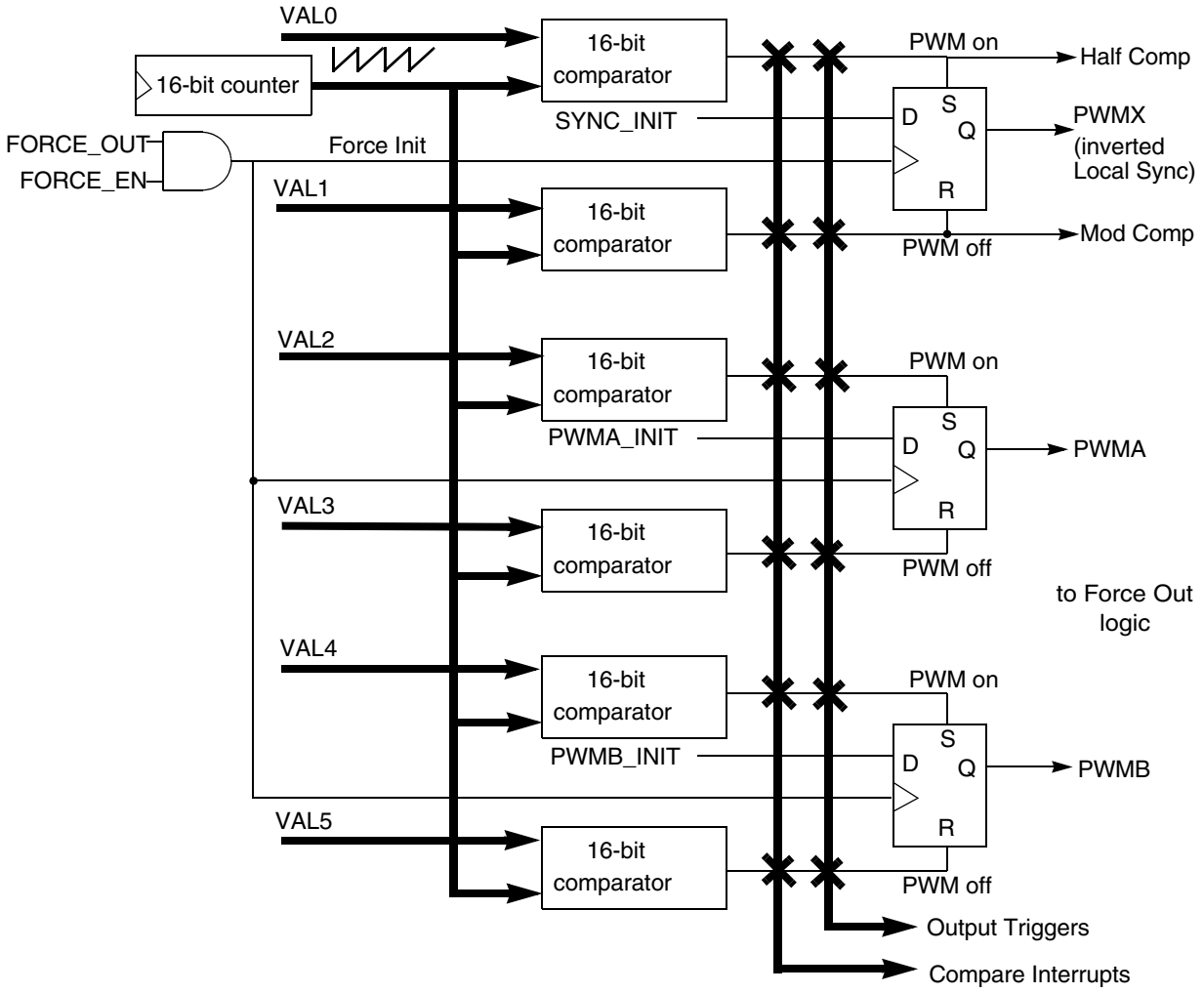


Figure 372. PWM generation hardware

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 will occur exactly half way through the timer count period and the Local Sync will have 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 deadtime distortion correction. Including the Local Sync signal, each submodule is capable of generating three 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 372* 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.

25.8.5 Output compare capabilities

By using the VALx 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 372*, an output compare is initiated by programming a VALx 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.

25.8.6 Force out logic

For each submodule software can select between seven signal sources for the FORCE_OUT signal: the local FORCE bit, the Master Force signal from submodule 0, the local Reload signal, the Master Reload signal from submodule 0, the Local Sync signal, the Master Sync signal from submodule 0, or the EXT_FORCE signal from on or off chip depending on the device 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 373 illustrates the Force logic. The SELA and SELB fields each choose from one of four signals that can be supplied to the submodule outputs: the PWM signal, the inverted PWM signal, a binary level specified by software via the OUTA and OUTB 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.

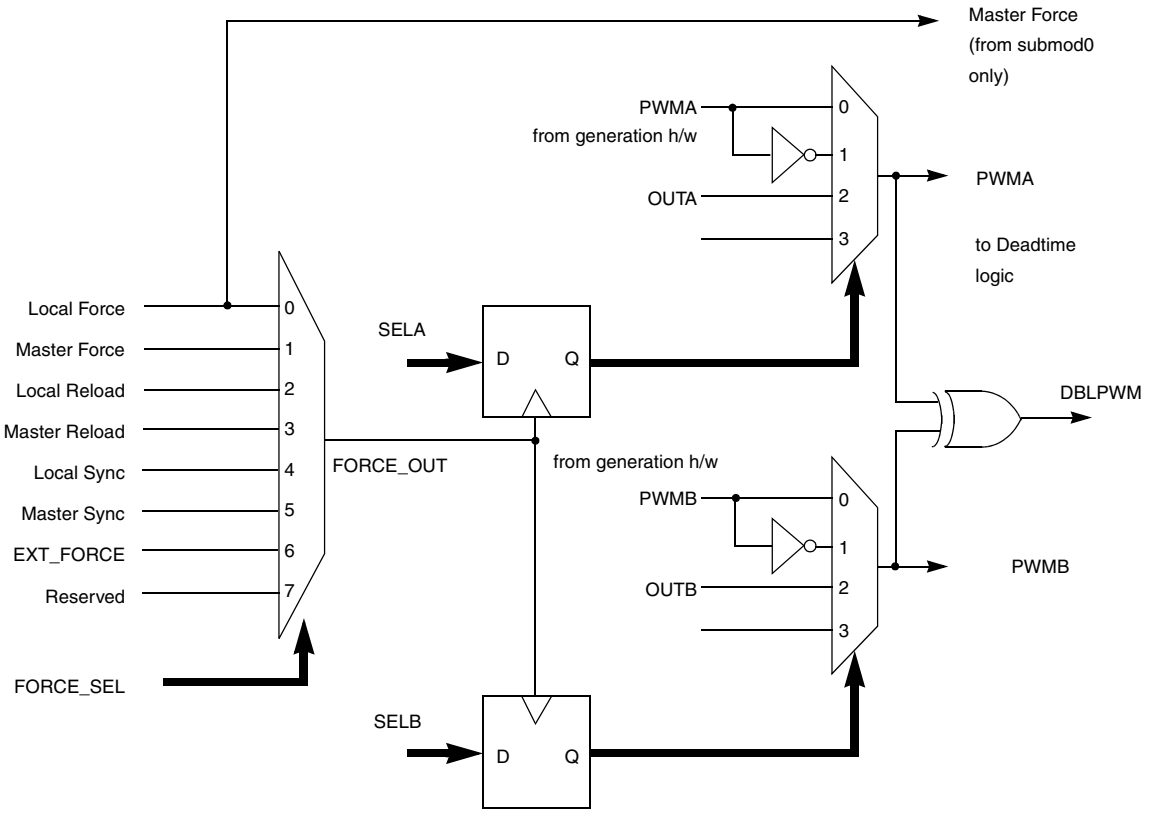


Figure 373. Force out logic

The local Force signal of submodule 0 can be broadcast as the Master Force signal to other submodules. This feature allows the local FORCE bit of submodule 0 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.

25.8.7 Independent or complementary channel operation

Writing a logic one 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 VALx pair operating independently of the other output.

Writing a logic zero to the INDEP bit configures the PWM output as a pair of complementary channels. The PWM pins are paired as shown in *Figure 374* in complementary channel operation. The IPOL bit determines which signal is connected to the output pin (PWMA or PWMB).

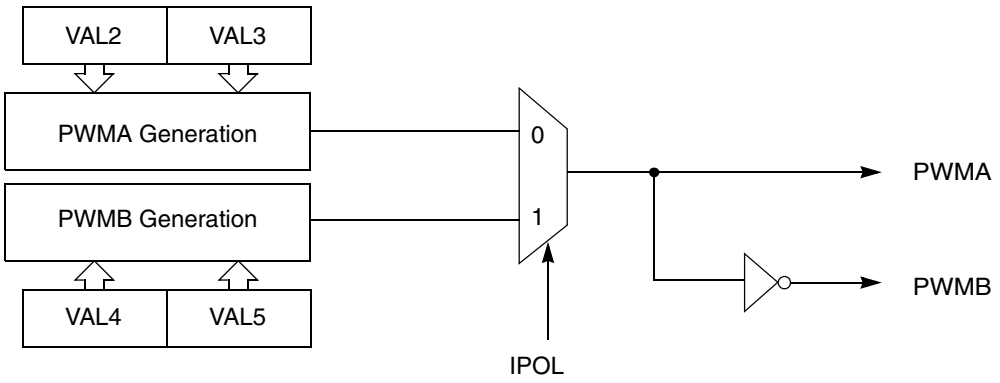


Figure 374. Complementary channel pair

The complementary channel operation is for driving top and bottom transistors in a motor drive circuit, such as the one in *Figure 375*.

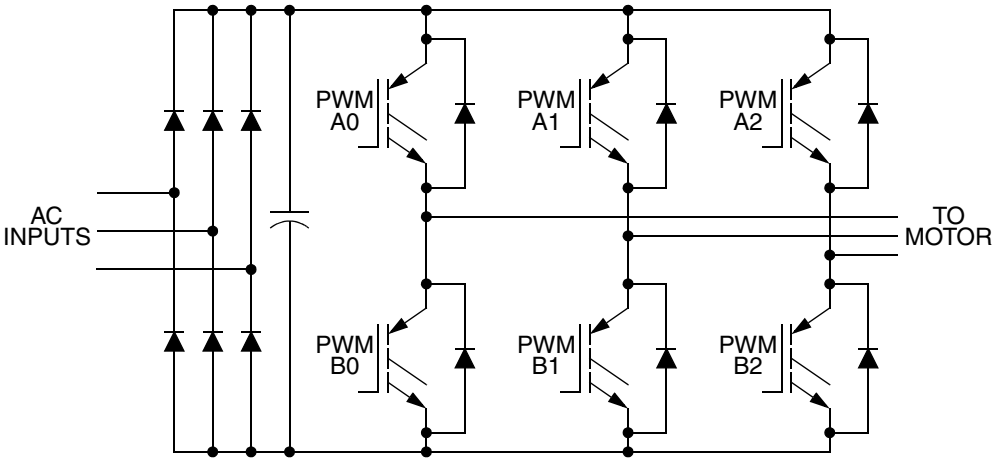


Figure 375. Typical 3-phase AC motor drive

Complementary operation allows the use of the deadtime insertion feature.

25.8.8 Deadtime insertion logic

Figure 376 shows the deadtime insertion logic of each submodule, which creates non-overlapping complementary signals when not in independent mode.

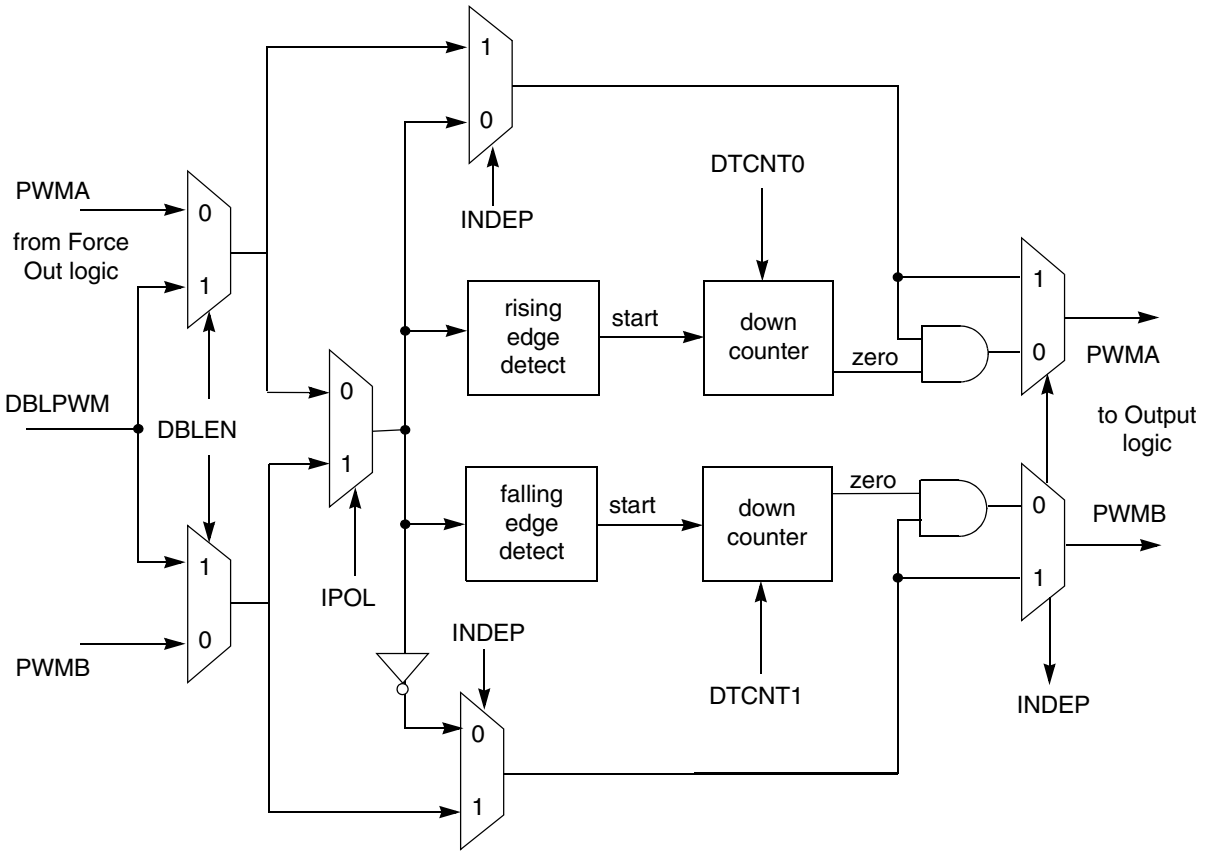


Figure 376. Deadtime insertion and fine control logic

While in the complementary mode, a PWM pair can be used to drive top/bottom transistors, as shown in *Figure 376*. 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. However, the transistor's characteristics may cause its switching-off time to be longer than its 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 377.

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 will have 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.

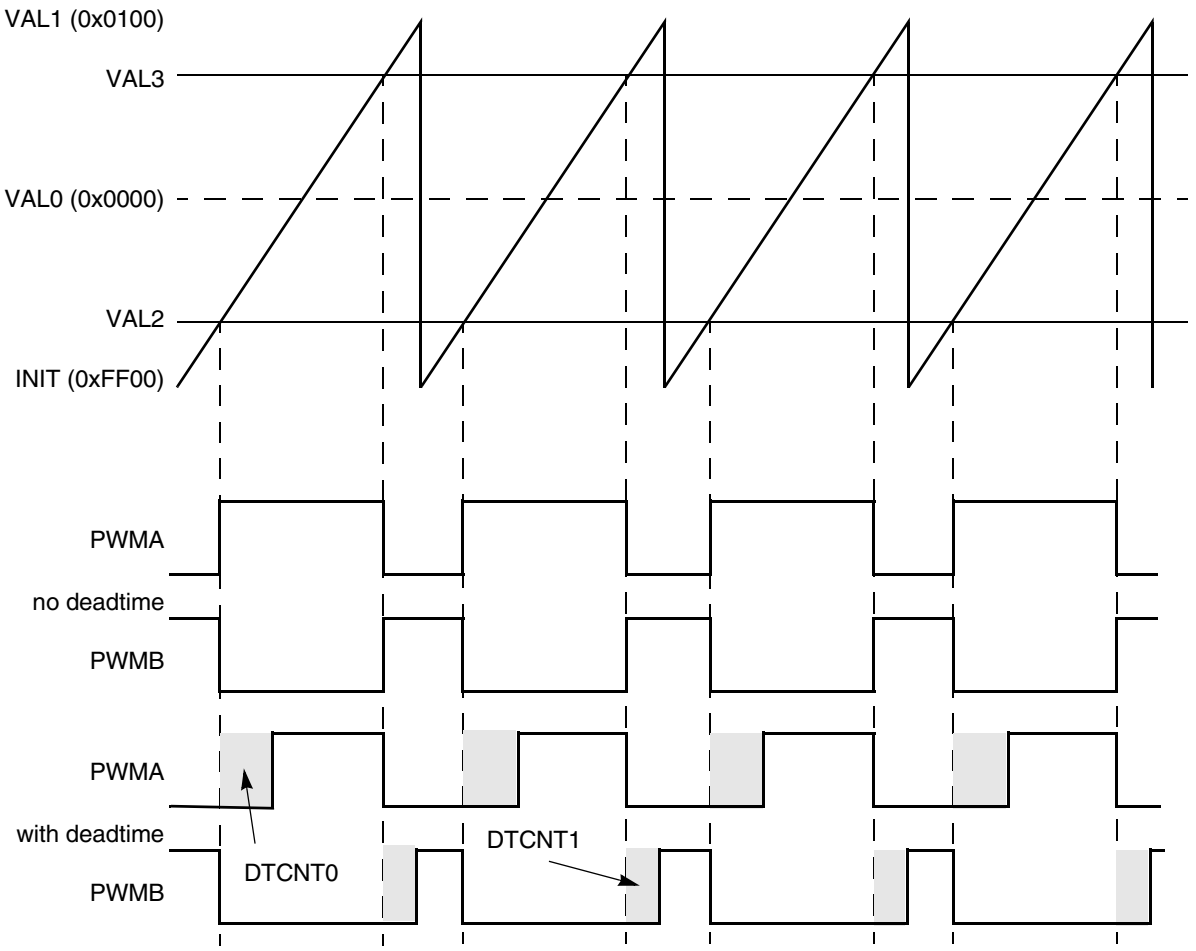


Figure 377. Deadtime insertion

25.8.9 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 378*. On AC induction motors running open-loop, the distortion typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

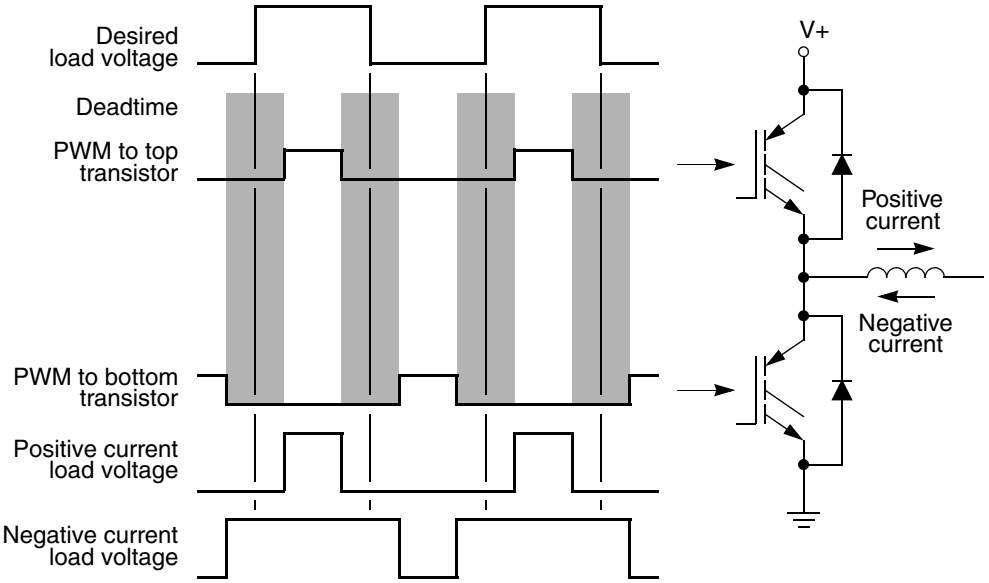


Figure 378. Deadtime distortion

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 where shortened by deadtime insertion, the averaged sinusoidal output will be 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 will be effective in controlling the output voltage at any given time. This depends on the direction of the motor current for that pair. See *Figure 379*. 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 VALx 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 either:

- The state of the current status pin, PWMx, for that driver
- The state of the odd/even correction bit, IPOL, for that driver

To correct deadtime distortion, software can decrease or increase the value in the appropriate VALx 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.

25.8.10 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. Software can then set the IPOL bit to switch between VAL2/VAL3 and VAL4/VAL5 register pairs according to DTx values.

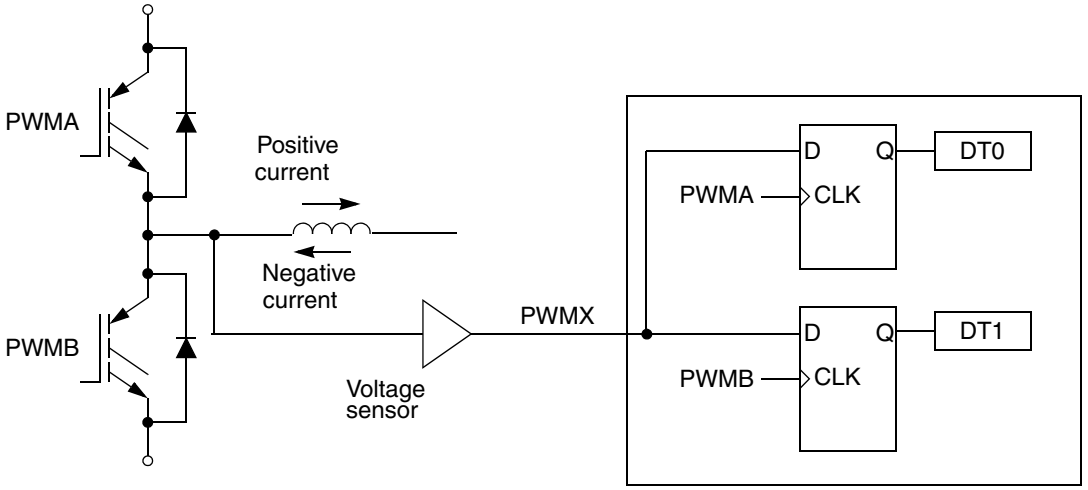


Figure 379. Current-status sense scheme for deadtime correction

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 379*. 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.

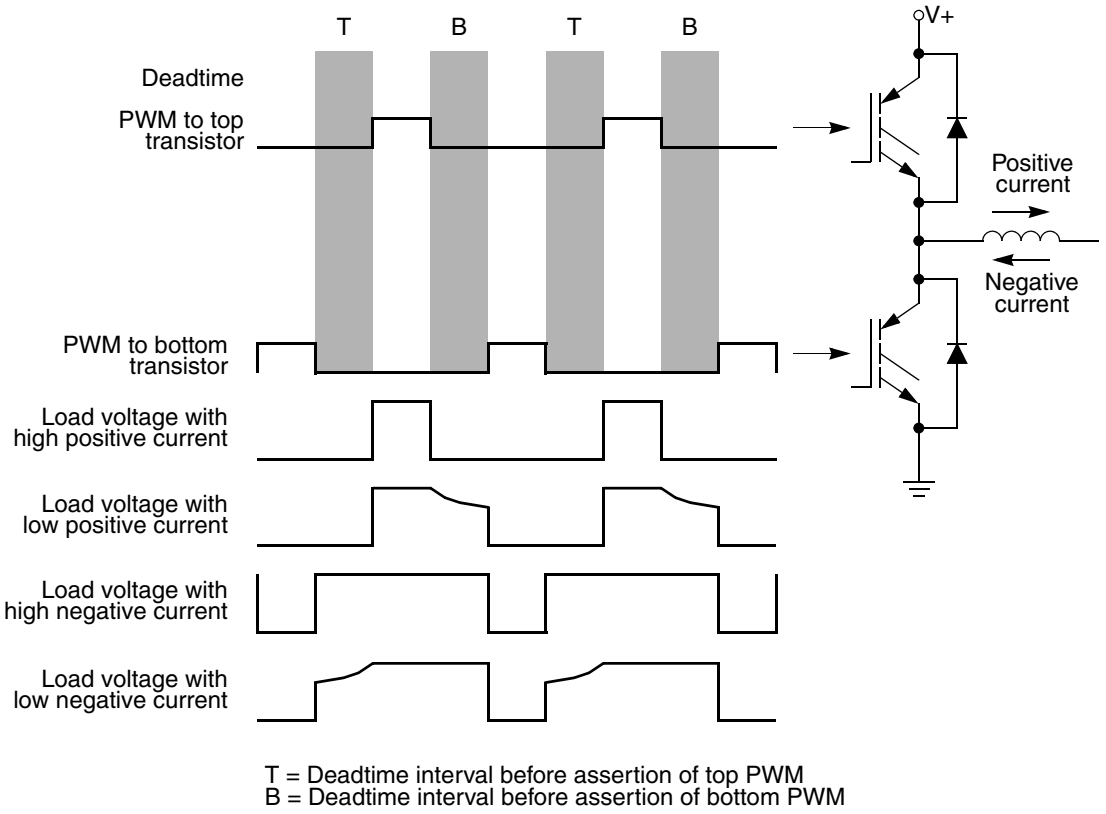


Figure 380. Output voltage waveforms

25.8.11 Output logic

Figure 381 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 PWMA and PWMB signals that are output from the deadtime logic in *Figure 381* 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.

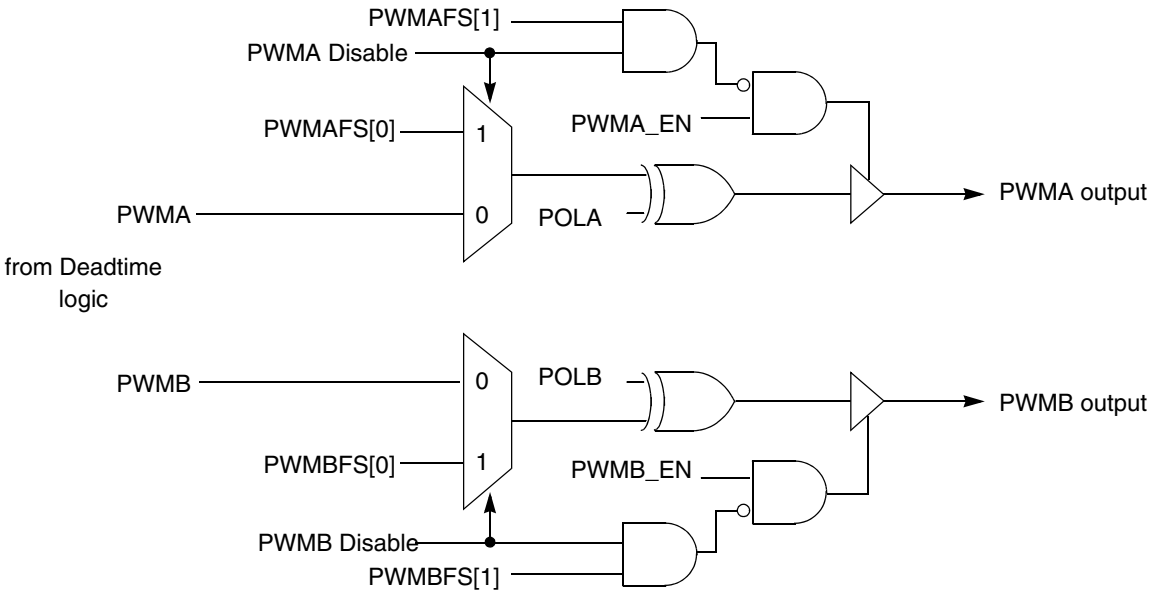


Figure 381. Output logic section

25.8.12 Fault protection

Fault protection can control any combination of PWM output pins. Faults are generated by a logic one on any of the FAULTx pins. This polarity can be changed via the FLVL bits. Each FAULTx 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 the values of the PWMxFS bits.

The fault decoder disables PWM pins selected by the fault logic and the disable mapping register (DISMAP). See *Figure 382* for an example of the fault disable logic. Each bank of bits in DISMAP control the mapping for a single PWM pin. Refer to *Table 361*.

The fault protection is enabled even when the PWM module is not enabled; therefore, a fault will be latched in and must be cleared in order to prevent an interrupt when the PWM is enabled.

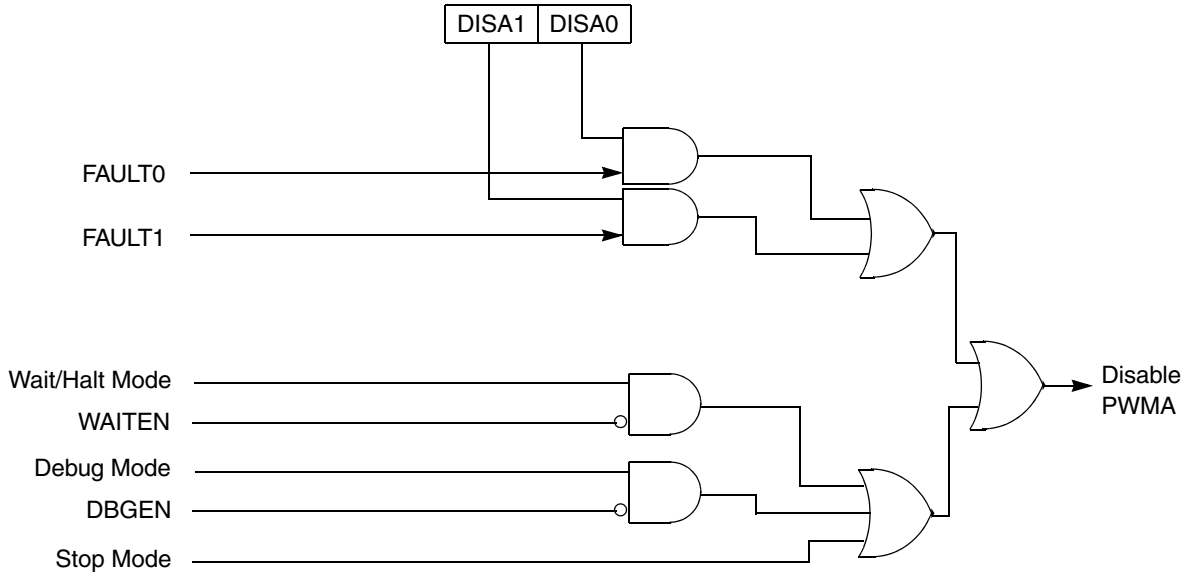


Figure 382. Fault decoder for PWMA

Table 361. Fault mapping

PWM pin	Controlling register bits
PWMA	DISA[1:0]
PWMB	DISB[1:0]
PWMX	DISX[1:0]

25.8.13 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 FFILTx 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 FAULTx pin.

Upon detecting a logic 0 on the filtered FAULTx pin (or a logic 1 if FLVLx is set), the corresponding FFPINx and fault flag, FFLAGx, bits are set. The FFPINx bit remains set as long as the filtered FAULTx pin is zero. Clear FFLAGx by writing a logic 1 to FFLAGx.

If the FIEx, FAULTx pin interrupt enable bit is set, the FFLAGx flag generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears the FFLAGx flag by writing a logic one to the bit
- Software clears the FIEx bit by writing a logic zero to it
- A reset occurs

Even with the filter enabled, there is a combinational path from the FAULTx 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.

25.8.14 Automatic fault clearing

Setting an automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for automatic clearing.

When FAUTOx is set, disabled PWM pins are enabled when the FAULTx pin returns to logic one and a new PWM either full or half cycle begins. See *Figure 383*. Clearing the FFLAGx flag does not affect disabled PWM pins when FAUTOx is set.

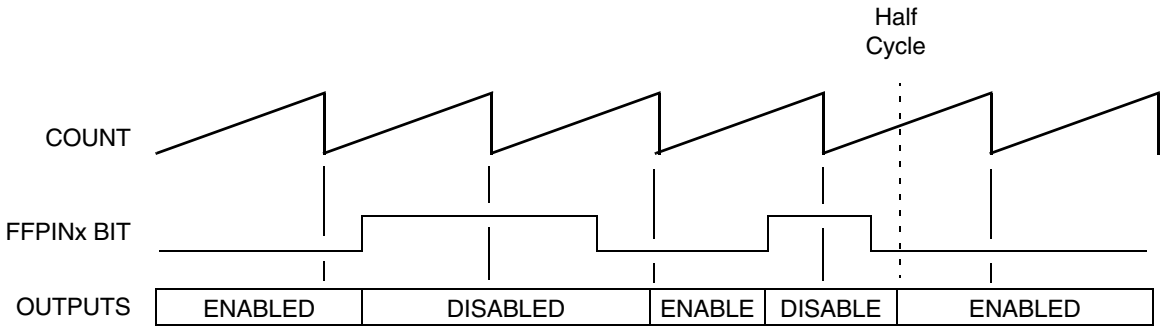


Figure 383. Automatic fault clearing

25.8.15 Manual fault clearing

Clearing the automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for manual clearing:

- If the fault safety mode bits, FSAFEx, are clear, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The pins are enabled when the next PWM half cycle begins regardless of the logic level detected by the filter at the FAULTx pin. See *Figure 384*.
- If the fault safety mode bits, FSAFEx, are set, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The filter detects a logic one on the FAULTx pin at the start of the next PWM half cycle boundary. See *Figure 385*.

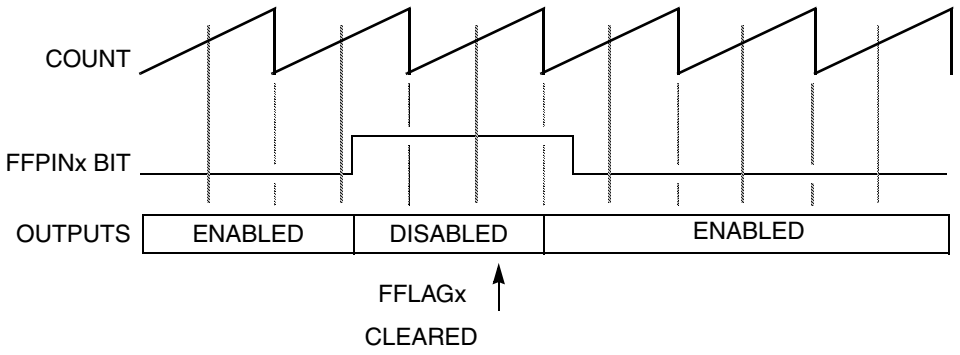


Figure 384. Manual fault clearing (FSAFE = 0)

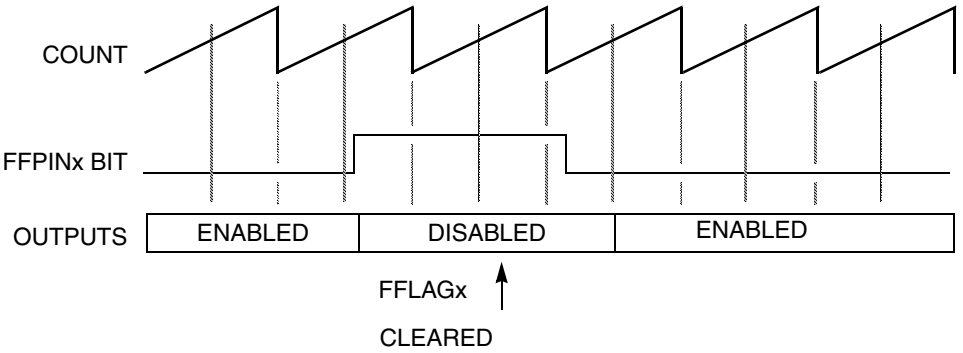


Figure 385. Manual fault clearing (FSAFE = 1)

Note: Fault protection also applies during software output control when the SELA and SELB fields are set to select OUTA and OUTB 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.

25.8.16 Fault testing

The FTEST bit simulates a fault condition on each of the fault inputs.

25.9 PWM generator loading

25.9.1 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 VALx registers

LDOK allows software to finish calculating all of these PWM parameters so they can be synchronously updated. The PSRC, INIT, and VALx 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. Set LDOK by reading it when it is a logic zero and then writing a logic one to it. After loading, LDOK is automatically cleared.

25.9.2 Load frequency

The LDFQ bits in the CTRL1 register select an integral loading frequency of one to 16 PWM reload opportunities. The LDFQ bits take effect at every PWM reload opportunity, regardless 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.

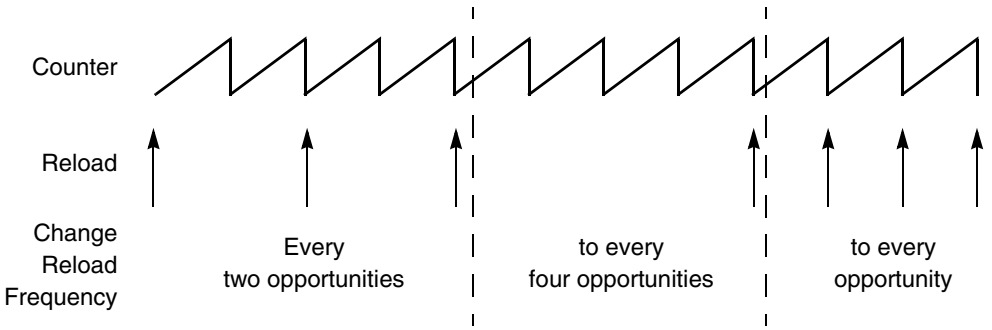


Figure 386. Full cycle reload frequency change

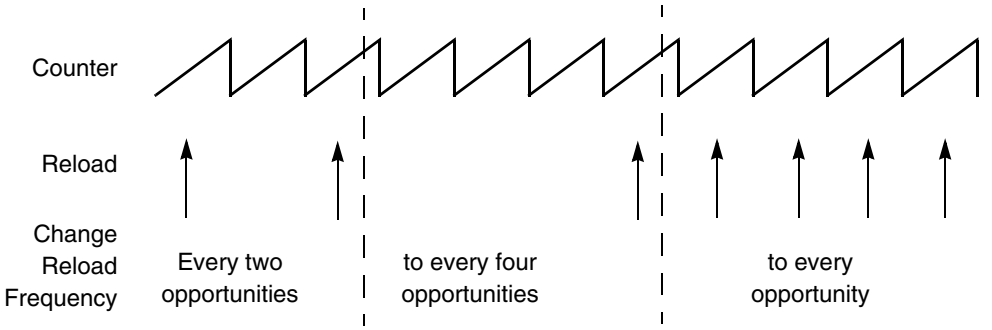


Figure 387. Half cycle reload frequency change

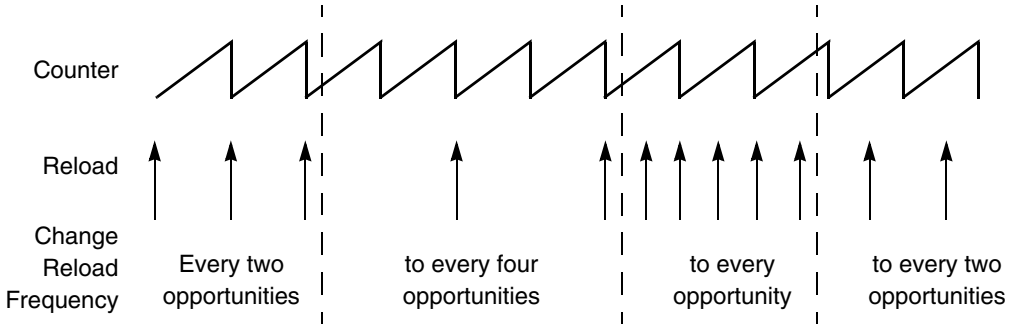


Figure 388. Full and half cycle reload frequency change

25.9.3 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.

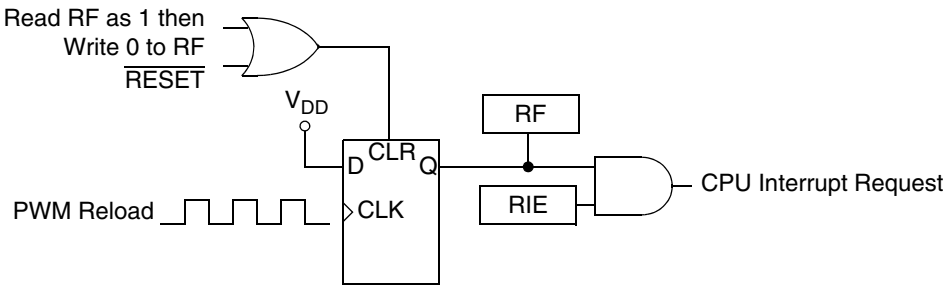


Figure 389. PWMF reload interrupt request

25.9.4 Reload errors

Whenever one of the VALx, or PSRC registers is updated, the RUF flag is set to indicate that the data is not coherent. RUF will be 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 will be flagged.

25.9.5 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

25.10 Clocks

25.11 Interrupts

Each of the submodules within the PWM 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 362. Interrupt summary

Core interrupt flag	Interrupt flag	Interrupt enable	Name	Description
COF0	CMPF_0	CMPIE_0	Submodule 0 compare interrupt	Compare 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
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
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
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

25.12 DMA

Each submodule can request a DMA write request for its double buffered VALx registers.

Table 363. DMA summary

DMA request	DMA enable	Name	Description
Submodule 0 write request	VALDE_0	VALx write request	VALx registers need to be updated
Submodule 1 write request	VALDE_1	VALx write request	VALx registers need to be updated
Submodule 2 write request	VALDE_2	VALx write request	VALx registers need to be updated
Submodule 3 write request	VALDE_3	VALx write request	VALx registers need to be updated

26 eTimer

26.1 Introduction

The eTimer module contains six identical counter/timer channels and one watchdog timer function. 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.

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 "snapshot" 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.

26.2 Features

The eTimer module design includes these features:

- Six 16-bit counters/timers
- Count up/down
- Cascadeable counters
- 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
- Preloadable counters
- Preloadable compare registers
- 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

26.3 Module block diagram

The eTimer block diagram is shown in *Figure 390*.

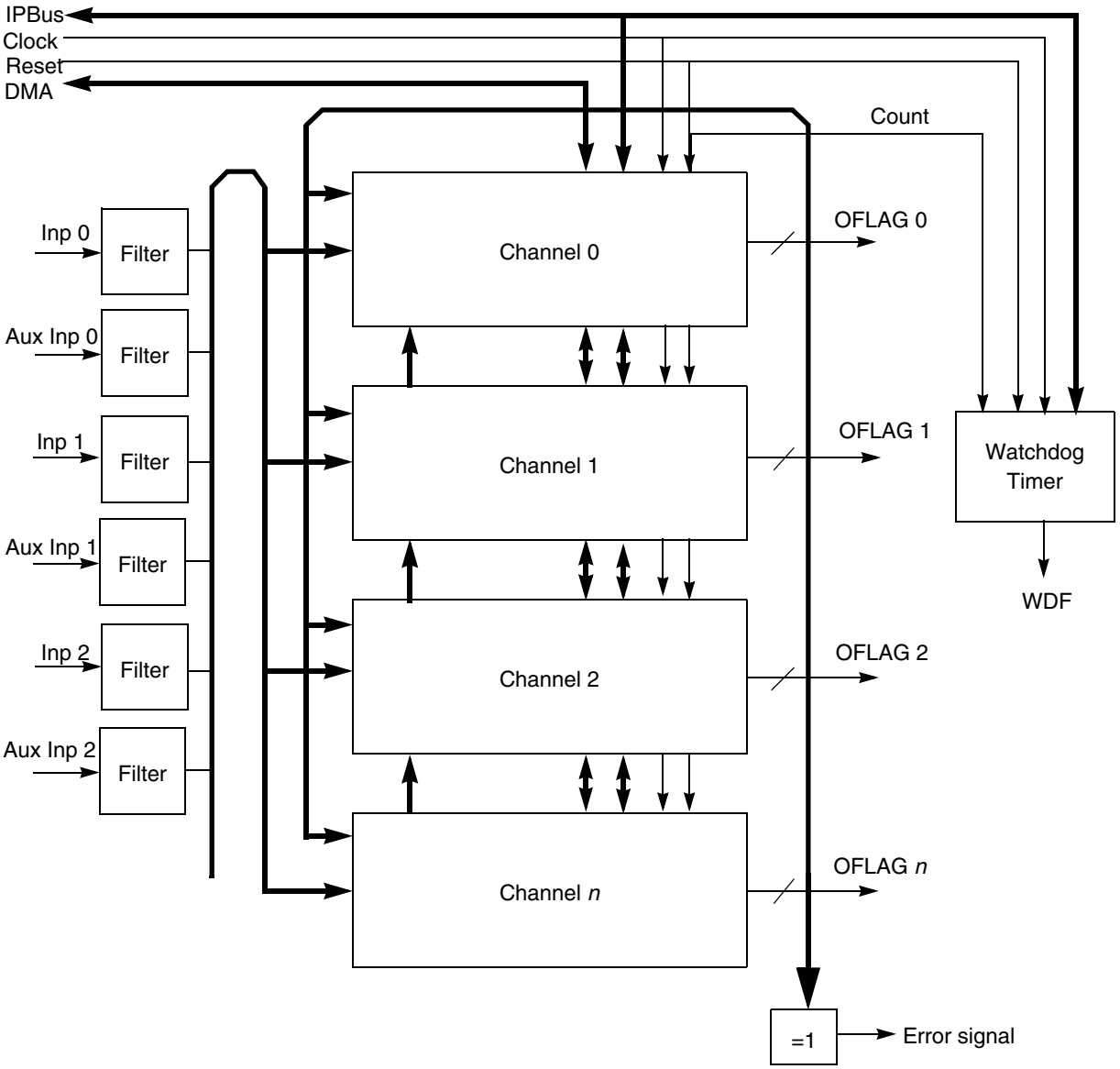


Figure 390. eTimer block diagram

26.4 Channel block diagram

Each of the timer/counter channels within the eTimer are shown in *Figure 391*.

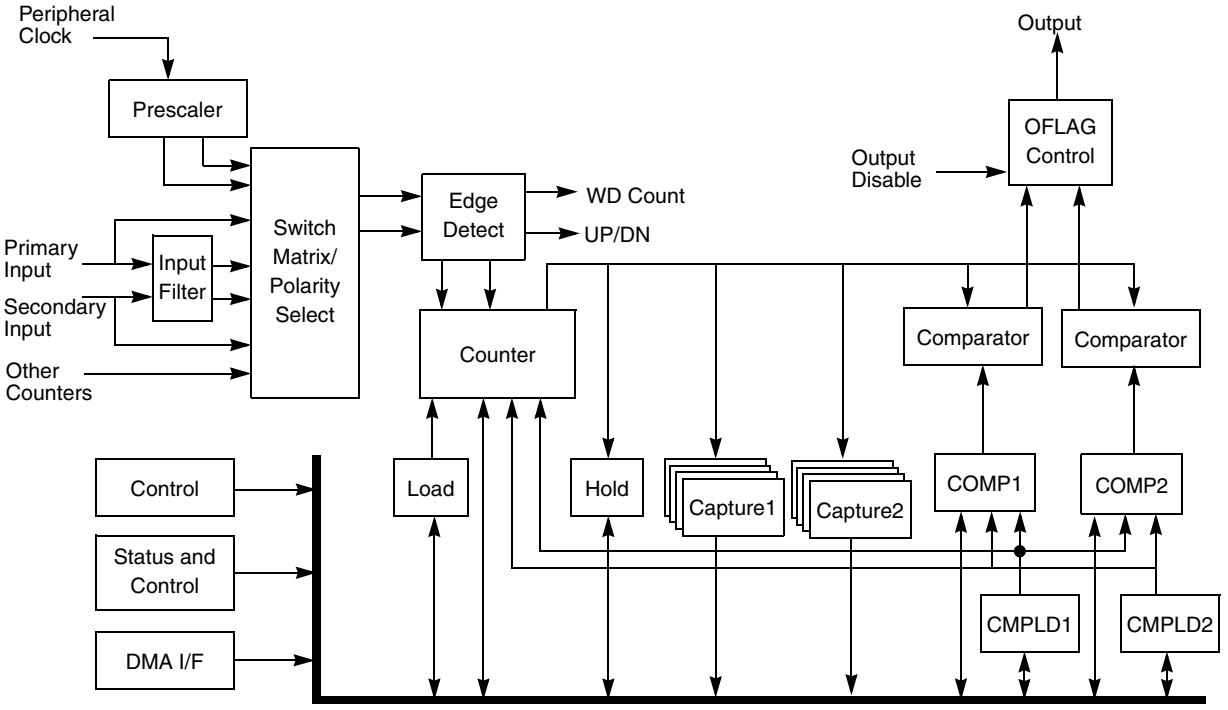


Figure 391. eTimer channel block diagram

26.5 External signal descriptions

The eTimer module has six external signals that can be used as either inputs or outputs.

26.5.1 ETC[5:0]—eTimer input/outputs

These pins can be independently configured to be either timer input sources or output flags.

26.6 Memory map and registers

26.6.1 Overview

Table 364 shows the memory map for the eTimer module.

Table 364. eTimer memory map

Offset from eTIMER0_BASE (FFE1_8000)	Register	Location
eTimer Channel 0		
0x0000	COMP1—Compare Register 1	*on page 26-710*
0x0002	COMP2—Compare Register 2	*on page 26-711*
0x0004	CAPT1—Capture Register 1	*on page 26-711*
0x0006	CAPT2—Capture Register 2	*on page 26-712*
0x0008	LOAD—Load Register	*on page 26-712*
0x000A	HOLD—Hold Register	*on page 26-713*
0x000C	CNTR—Counter Register	*on page 26-713*
0x000E	CTRL1—Control Register 1	*on page 26-714*
0x0010	CTRL2—Control Register 2	*on page 26-716*
0x0012	CTRL3—Control Register 3	*on page 26-718*
0x0014	STS—Status Register	*on page 26-719*
0x0016	INTDMA—Interrupt and DMA Enable Register	*on page 26-721*
0x0018	CMPLD1—Comparator Load Register 1	*on page 26-722*
0x001A	CMPLD2—Comparator Load Register 2	*on page 26-722*
0x001C	CCCTRL—Compare and Capture Control Register	*on page 26-723*
0x001E	FILT—Input Filter Register	*on page 26-725*
eTimer Channel 1		
0x0020	COMP1—Compare Register 1	*on page 26-710*
0x0022	COMP2—Compare Register 2	*on page 26-711*
0x0024	CAPT1—Capture Register 1	*on page 26-711*
0x0026	CAPT2—Capture Register 2	*on page 26-712*
0x0028	LOAD—Load Register	*on page 26-712*
0x002A	HOLD—Hold Register	*on page 26-713*
0x002C	CNTR—Counter Register	*on page 26-713*
0x002E	CTRL1—Control Register 1	*on page 26-714*
0x0030	CTRL2—Control Register 2	*on page 26-716*
0x0032	CTRL3—Control Register 3	*on page 26-718*
0x0034	STS—Status Register	*on page 26-719*
0x0036	INTDMA—Interrupt and DMA Enable Register	*on page 26-721*
0x0038	CMPLD1—Comparator Load Register 1	*on page 26-722*
0x003A	CMPLD2—Comparator Load Register 2	*on page 26-722*
0x003C	CCCTRL—Compare and Capture Control Register	*on page 26-723*

Table 364. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000)	Register	Location
0x003E	FILT—Input Filter Register	on page 26-725
eTimer Channel 2		
0x0040	COMP1—Compare Register 1	on page 26-710
0x0042	COMP2—Compare Register 2	on page 26-711
0x0044	CAPT1—Capture Register 1	on page 26-711
0x0046	CAPT2—Capture Register 2	on page 26-712
0x0048	LOAD—Load Register	on page 26-712
0x004A	HOLD—Hold Register	on page 26-713
0x004C	CNTR—Counter Register	on page 26-713
0x004E	CTRL1—Control Register 1	on page 26-714
0x0050	CTRL2—Control Register 2	on page 26-716
0x0052	CTRL3—Control Register 3	on page 26-718
0x0054	STS—Status Register	on page 26-719
0x0056	INTDMA—Interrupt and DMA Enable Register	on page 26-721
0x0058	CMPLD1—Comparator Load Register 1	on page 26-722
0x005A	CMPLD2—Comparator Load Register 2	on page 26-722
0x005C	CCCTRL—Compare and Capture Control Register	on page 26-723
0x005E	FILT—Input Filter Register	on page 26-725
eTimer Channel 3		
0x0060	COMP1—Compare Register 1	on page 26-710
0x0062	COMP2—Compare Register 2	on page 26-711
0x0064	CAPT1—Capture Register 1	on page 26-711
0x0066	CAPT2—Capture Register 2	on page 26-712
0x00068	LOAD—Load Register	on page 26-712
0x006A	HOLD—Hold Register	on page 26-713
0x006C	CNTR—Counter Register	on page 26-713
0x006E	CTRL1—Control Register 1	on page 26-714
0x0070	CTRL2—Control Register 2	on page 26-716
0x0072	CTRL3—Control Register 3	on page 26-718
0x0074	STS—Status Register	on page 26-719
0x0076	INTDMA—Interrupt and DMA Enable Register	on page 26-721
0x0078	CMPLD1—Comparator Load Register 1	on page 26-722
0x007A	CMPLD2—Comparator Load Register 2	on page 26-722

Table 364. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000)	Register	Location
0x007C	CCCTRL—Compare and Capture Control Register	on page 26-723
0x007E	FILT—Input Filter Register	on page 26-725
eTimer Channel 4		
0x0080	COMP1—Compare Register 1	on page 26-710
0x0082	COMP2—Compare Register 2	on page 26-711
0x0084	CAPT1—Capture Register 1	on page 26-711
0x0086	CAPT2—Capture Register 2	on page 26-712
0x0088	LOAD—Load Register	on page 26-712
0x008A	HOLD—Hold Register	on page 26-713
0x008C	CNTR—Counter Register	on page 26-713
0x008E	CTRL1—Control Register 1	on page 26-714
0x0090	CTRL2—Control Register 2	on page 26-716
0x0092	CTRL3—Control Register 3	on page 26-718
0x0094	STS—Status Register	on page 26-719
0x0096	INTDMA—Interrupt and DMA Enable Register	on page 26-721
0x0098	CMPLD1—Comparator Load Register 1	on page 26-722
0x009A	CMPLD2—Comparator Load Register 2	on page 26-722
0x009C	CCCTRL—Compare and Capture Control Register	on page 26-723
0x009E	FILT—Input Filter Register	on page 26-725
eTimer Channel 5		
0x00A0	COMP1—Compare Register 1	on page 26-710
0x00A2	COMP2—Compare Register 2	on page 26-711
0x00A4	CAPT1—Capture Register 1	on page 26-711
0x00A6	CAPT2—Capture Register 2	on page 26-712
0x00A8	LOAD—Load Register	on page 26-712
0x00AA	HOLD—Hold Register	on page 26-713
0x00AC	CNTR—Counter Register	on page 26-713
0x00AE	CTRL1—Control Register 1	on page 26-714
0x00B0	CTRL2—Control Register 2	on page 26-716
0x00B2	CTRL3—Control Register 3	on page 26-718
0x00B4	STS—Status Register	on page 26-719
0x00B6	INTDMA—Interrupt and DMA Enable Register	on page 26-721
0x00B8	CMPLD1—Comparator Load Register 1	on page 26-722

Table 364. eTimer memory map (continued)

Offset from eTIMER0_BASE (FFE1_8000)	Register	Location
0x00BA	CMPLD2—Comparator Load Register 2	on page 26-722
0x00BC	CCCTRL—Compare and Capture Control Register	on page 26-723
0x00BE	FILT—Input Filter Register	on page 26-725
0x00C0–0x00FF	Reserved	
0x0100	WDTOL—Watchdog Time-out Low Register	on page 26-726
0x0102	WDTOH—Watchdog Time-out High Register	on page 26-726
0x0104–0x010B	Reserved	
Watchdog and Configuration registers		
0x010C	ENBL—Channel Enable Register	on page 26-726
0x0110	DREQ0—DMA Request 0 Select Register	on page 26-727
0x0112	DREQ1—DMA Request 1 Select Register	on page 26-727
0x0114–0x3FFF	Reserved	

26.6.2 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.

Compare register 1 (COMP1)

The COMP1 register stores the value used for comparison with the counter value. More explanation on the use of COMP1 can be found in *Section , "Usage of compare registers*.

Figure 392. Compare register 1 (COMP1)

Address: Base + 0x0000 (eTimer0), Base + 0x0020 (eTimer1), Base + 0x0040 (eTimer2), Base + 0x0060 (eTimer3), Base + 0x0080 (eTimer4), Base + 0x00A0 (eTimer5)

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

Table 365. COMP1 field descriptions

Field	Description
COMP1[15:0]	Compare 1 Stores the value used for comparison with the counter value. This register is not byte accessible.

Compare register 2 (COMP2)

The COMP2 register stores the value used for comparison with the counter value. More explanation on the use of COMP2 can be found in *Section , "Usage of compare registers.*

Figure 393. Compare register 2 (COMP2)

Address: Base + 0x0002 (eTimer0), Base + 0x0022 (eTimer1), Base + 0x0042 (eTimer2), Base + 0x0062 (eTimer3), Base + 0x0082 (eTimer4), Base + 0x00A2 (eTimer5) — 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

Table 366. COMP2 field descriptions

Field	Description
COMP2[15:0]	Compare 2 Stores the value used for comparison with the counter value. This register is not byte accessible.

Capture register 1 (CAPT1)

The CAPT1 register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT1MODE bits in the Compare and Capture Control (CCCTRL) register. This is actually a 2-deep FIFO and not a single register.

Figure 394. Capture register 1 (CAPT1)

Address: Base + 0x0004 (eTimer0), Base + 0x0024 (eTimer1), Base + 0x0044 (eTimer2), Base + 0x0064 (eTimer3), Base + 0x0084 (eTimer4), Base + 0x00A4 (eTimer5) — 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

Table 367. CAPT1 field descriptions

Field	Description
CAPT1[15:0]	Capture 1 Stores the value captured from the counter. This register is not byte accessible.

Capture register 2 (CAPT2)

This read only register stores the value captured from the counter. Exactly when a capture occurs is defined by the CPT2MODE bits. This is actually a 2-deep FIFO and not a single register. This register is not byte accessible.

Figure 395. Capture register 2 (CAPT2)

Address: Base + 0x0006 (eTimer0), Base + 0x0026 (eTimer1), Base + 0x0046 (eTimer2), Base + 0x0066 (eTimer3), Base + 0x0086 (eTimer4), Base + 0x00A6 (eTimer5)

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

Table 368. CAPT2 field descriptions

Field	Description
CAPT2[15:0]	Capture 2 Stores the value captured from the counter. This register is not byte accessible.

Load register (LOAD)

This read/write register stores the value used to initialize the counter. This register is not byte accessible.

Figure 396. Load register (LOAD)

Address: Base + 0x0008 (eTimer0), Base + 0x0028(eTimer1), Base + 0x0048 (eTimer2), Base + 0x0068 (eTimer3), Base + 0x0088 (eTimer4), Base + 0x00A8 (eTimer5)

Access: User read/write

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

Table 369. LOAD field descriptions

Field	Description
LOAD[15:0]	Load Stores the value used to initialize the counter. This register is not byte accessible.

Hold register (HOLD)

This read-only register stores the counter's value whenever any of the other counters within a module are read. This supports coherent reading of cascaded counters.

Figure 397. Hold register (HOLD)

Address: Base + 0x000A (eTimer0), Base + 0x002A (eTimer1), Base + 0x004A (eTimer2), Base + 0x006A (eTimer3), Base + 0x008A (eTimer4), Base + 0x00AA (eTimer5) — 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

Table 370. HOLD field descriptions

Field	Description
HOLD[15:0]	Stores the counter's value whenever any of the other counters within a module are read. The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

Counter register (CNTR)

This read/write register is the counter for this channel of the timer module. This register is not byte accessible.

Figure 398. Counter register (CNTR)

Address: Base + 0x000C (eTimer0), Base + 0x002C (eTimer1), Base + 0x004C (eTimer2), Base + 0x006C (eTimer3), Base + 0x008C (eTimer4), Base + 0x00AC (eTimer5) — 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

Table 371. CNTR field descriptions

Field	Description
CNTR[15:0]	Contains the count value for this channel of the eTimer module. This register is not byte accessible.

Control register 1 (CTRL1)

Figure 399. Control register 1 (CTRL1)

Address: Base + 0x000E(eTimer0), Base + 0x002E (eTimer1), Base + 0x004E (eTimer2), Base + 0x006E (eTimer3), Base + 0x008E (eTimer4), Base + 0x00AE (eTimer5)

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

Table 372. CTRL1 field descriptions

Field	Description
CNTMODE[2:0]	Count Mode These bits control the basic counting and behavior of the counter. 000 No Operation. 001 Count rising edges of primary source. Rising edges counted only 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 can not 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 rising edges, secondary source specifies direction (1 = minus). Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. 110 Edge of secondary source triggers primary count till compare. 111 Cascaded counter mode, up/down. Primary count source must be set to one of the counter outputs.
PRISRC	Primary Count Source These bits select the primary count source. See *Table 373*. A timer cannot select its own output as its primary count source. If this is done, the timer will not count.
ONCE	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 reinitializes after reaching the COMP1 value and continues to count to the COMP2 value, then stops.

Table 372. CTRL1 field descriptions (continued)

Field	Description
LENGTH	Count Length This bit determines whether the counter counts to the compare value and then reinitializes itself to the value specified in the LOAD, CMPLD1, or CMPLD2 registers, or the counter continues counting past the compare value, to the binary roll over. 0 Continue counting to roll over. 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 will reinitialize 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 will have 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, reinitializes, then counts until COMP2 value is reached, reinitializes, then counts until COMP1 value is reached, etc.
DIR	Count Direction This bit selects either the normal count direction up, or the reverse direction, down. 0Count up. 1Count down.
SECSRC	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.

Table 373. Count source values

Value	Meaning	Value	Meaning
00000	Counter #0 input pin	10000	Counter #0 output
00001	Counter #1 input pin	10001	Counter #1 output
00010	Counter #2 input pin	10010	Counter #2 output
00011	Counter #3 input pin	10011	Counter #3 output
00100	Counter #4 input pin	10100	Counter #4 output
00101	Counter #5 input pin	10101	Counter #5 output
00110	*Reserved*	10110	*Reserved*
00111	*Reserved*	10111	*Reserved*
01000	Auxiliary input #0 pin	11000	IP Bus clock divide by 1 prescaler
01001	Auxiliary input #1 pin	11001	IP Bus clock divide by 2 prescaler
01010	Auxiliary input #2 pin	11010	IP Bus clock divide by 4 prescaler
01011	*Reserved*	11011	IP Bus clock divide by 8 prescaler

Table 373. Count source values (continued)

Value	Meaning	Value	Meaning
01100	*Reserved*	11100	IP Bus clock divide by 16 prescaler
01101	*Reserved*	11101	IP Bus clock divide by 32 prescaler
01110	*Reserved*	11110	IP Bus clock divide by 64 prescaler
01111	*Reserved*	11111	IP Bus clock divide by 128 prescaler

Control register 2 (CTRL2)

Figure 400. Control register 2 (CTRL2)

Address: Base + 0x0010 (eTimer0), Base + 0x0030 (eTimer1), Base + 0x0050 (eTimer2), Base + 0x0070 (eTimer3), Base + 0x0090 (eTimer4), Base + 0x00B0 (eTimer5)

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

Table 374. 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 will see the driven value. The polarity of the signal will be 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). 0Disable redundant channel checking. 1Enable 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.

Table 374. CTRL2 field descriptions (continued)

Field	Description
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). 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 reinitialization of this channel when the other channel has an active compare event. 00 Other channels cannot force reinitialization of this channel. 01 Other channels may force a reinitialization of this channel's counter using the LOAD reg. 10 Other channels may force a reinitialization 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 374. CTRL2 field descriptions (continued)

Field	Description
MSTR	Master Mode This bit enables the compare function's output to be broadcast 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 roll-over 1000 Set on successful compare on COMP1, clear 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

Control register 3 (CTRL3)

Figure 401. Control register 3 (CTRL3)

Address: Base + 0x0012 (eTimer0), Base + 0x0032 (eTimer1), Base + 0x0052 (eTimer2), Base + 0x0072 (eTimer3), Base + 0x0092 (eTimer4), Base + 0x00B2 (eTimer5)

Access: User read/write

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

Table 375. CTRL3 field descriptions

Field	Description
STPEN	Stop Actions Enable This bit allows the tristating 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.
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 device 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.

Status register (STS)

Figure 402. Status register (STS)

Address: Base + 0x0014 (eTimer0), Base + 0x0034 (eTimer1), Base + 0x0054 (eTimer2), Base + 0x0074 (eTimer3), Base + 0x0094 (eTimer4), Base + 0x00B4 (eTimer5)

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

Table 376. STS field descriptions

Field	Description
WDF	Watchdog Time-out Flag This bit is set when the watchdog times out by counting down to zero. The watchdog must be enabled for time-out 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. This bit is used in channel 0 only.
RCF	Redundant Channel Flag This bit is set when there is a miscompare 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, 4 and 5, or 6 and 7. This bit can only be set if the RDNT bit is set. This bit is cleared by writing a 1 to this bit. This bit is used in even channels (0, 2, 4, and 6) only.
ICF2	Input Capture 2 Flag This bit is set when an input capture event (as defined by CPT2MODE) occurs while the counter is enabled and the word count of the CAPT2 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a 1 to this bit 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 1 to this bit 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. This bit is cleared by writing a 1 to this bit.
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. This bit is cleared by writing a 1 to this bit.
TOF	Timer Overflow Flag This bit is set when the counter rolls over its maximum value 0xFFFF or 0x0000 (depending on count direction). This bit is cleared by writing a 1 to this bit.
TCF2	Timer Compare 2 Flag This bit is set when a successful compare occurs with COMP2. This bit is cleared by writing a 1 to this bit.
TCF1	Timer Compare 1 Flag This bit is set when a successful compare occurs with COMP1. This bit is cleared by writing a 1 to this bit.
TCF	Timer Compare Flag This bit is set when a successful compare occurs. This bit is cleared by writing a 1 to this bit.

Interrupt and DMA enable register (INTDMA)

Figure 403. Interrupt and DMA enable register (INTDMA)

Address: Base + 0x0016 (eTimer0), Base + 0x0036 (eTimer1), Base + 0x0056 (eTimer2), Base + 0x0076 (eTimer3), Base + 0x0096 (eTimer4), Base + 0x00B6 (eTimer5) — 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

Table 377. 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 either the CNTR, COMP1, or COMP2 registers.
WDFIE	Watchdog Flag Interrupt Enable Setting this bit enables interrupts when the WDF bit is set. This bit is used in channel 0 only.
RCFIE	Redundant Channel Flag Interrupt Enable Setting this bit enables interrupts when the RCF bit is set. This bit is used in even channels (0, 2, 4) only.
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.
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.

Table 377. INTDMA field descriptions (continued)

Field	Description
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.

Comparator Load register 1 (CMPLD1)

This read/write register 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 , "Usage of Compare Load registers*.

Figure 404. Comparator Load 1 (CMPLD1)

Address: Base + 0x0018 (eTimer0), Base + 0x0038 (eTimer1), Base + 0x0058 (eTimer2), Base + 0x0078 (eTimer3), Base + 0x0098 (eTimer4), Base + 0x00B8 (eTimer5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 378. CMPLD1 field descriptions

Field	Description
CMPLD1[15:0]	Specifies the preload value for the COMP1 register.

Comparator Load register 2 (CMPLD2)

This read/write register 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 , "Usage of Compare Load registers*.

Figure 405. Comparator Load 2 (CMPLD2)

Address: Base + 0x001A (eTimer0), Base + 0x003A (eTimer1), Base + 0x005A (eTimer2), Base + 0x007A (eTimer3), Base + 0x009A (eTimer4), Base + 0x00BA (eTimer5)

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

Table 379. CMPLD2 field descriptions

Field	Description
CMPLD2[15:0]	Specifies the preload value for the COMP2 register.

Compare and Capture Control register (CCCTRL)

Figure 406. Compare and Capture Control register (CCCTRL)

Address: Base + 0x001C (eTimer0), Base + 0x003C (eTimer1), Base + 0x005C (eTimer2), Base + 0x007C (eTimer3), Base + 0x009C (eTimer4), Base + 0x00BC (eTimer5) 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]			CMPMODE [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

Table 380. 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.
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.

Table 380. CCCTRL field descriptions (continued)

Field	Description
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. 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 will be performed until the ARM bit is set again. If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARM bit is then cleared. 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. 0 Free-running mode is selected 1 One-shot mode is selected.
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.

Input Filter Register (FILT)

Figure 407. Input Filter register (FILT)

Address: Base + 0x001E (eTimer0), Base + 0x003E (eTimer1), Base + 0x005E (eTimer2), Base + 0x007E (eTimer3), Base + 0x009E (eTimer4), Base + 0x00BE (eTimer5) — 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

Table 381. FILT field descriptions

Field	Description
FILT_CNT[2:0]	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 , "Input filter considerations*.
FILT_PER[7:0]	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 , "Input filter considerations*.

Input filter considerations

The FILT_PER value should be set such that the sampling period is larger the period of the expected noise. This way a noise spike will only corrupt 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: $\{[(\text{FILT\_CNT} + 3) \times \text{FILT\_PER}] + 2\}$ IPBus clock periods.

26.6.3 Watchdog timer registers

The base address of the Watchdog Timer registers is equal to the base address of the eTimer plus an offset of 0x100.

Watchdog Time-Out registers (WDTOL and WDTOH)

Figure 408. Watchdog Time-out Low Word register (WDTOL)

Address: Base + 0x0100 Access: User read/write

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

Figure 409. Watchdog Time-Out High Word register (WDTOH)

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

Table 382. WDTOL, WDTOH field descriptions

Field	Description
WDTO	Watchdog Time-out Note: These registers are combined to form the 32-bit time-out count for the Timer watchdog function. This time-out count is used to monitor 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 time-out down counter is loaded whenever WDTOH is written. These registers are not byte accessible. See *Section , "Watchdog timer* for more information on the use of the watchdog timer.

26.6.4 Configuration registers

The base address of the configuration registers is equal to the base address of the eTimer plus an offset of 0x010C.

Channel Enable register (ENBL)

Figure 410. 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

Table 383. 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 will start 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)

DMA Request Select registers (DREQ0, DREQ1)

Figure 411. DMA Request 0 Select register (DREQ0)

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[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 412. DMA Request 1 Select register (DREQ1)

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[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 384. DREQ*n* field descriptions

Field	Description
DREQ*n*_EN	DMA Request Enable Use these bits to enable each of the four module level DMA request outputs. Program the DREQ fields prior to setting the corresponding enable bit. Clearing this enable bit will remove the request but wíll not clear the flag that is causing the request. 1 = DMA request enabled. 0 = DMA request disabled.
DREQ*n*	DMA Request Select Use these fields to select which DMA request source will be muxed onto one of the two module level DMA request outputs. Make sure each of the DREQ registers is programmed with a different value else a single DMA source will cause multiple DMA requests. Enable a DMA request in the channel specific INTDMA register after the DREQ registers are programmed. 00000Channel 0 CAPT1 DMA read request 00001Channel 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 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

26.7 Functional description

26.7.1 General

Each channel has two basic modes of operation: it can count 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 can count the rising, falling, or both edges of the selected input pin.
- The counter can decode and count quadrature encoded input signals.
- The counter can count up and down using dual inputs in a “count with direction” format.
- The counter’s terminal count value (modulo) is programmable.
  - The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter can count repeatedly, or it can stop after completing one count cycle.
- The counter 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
- Timer commands
- They can trigger the current counter value to be “captured”
- They can 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 broadcast 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.

26.7.2 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.

STOP mode

When the CNTMODE field is set to 000, the counter is inert. No counting will occur. Stop mode will also disable the interrupts caused by input transitions on a selected input pin.

COUNT mode

When the CNTMODE field is set to 001, the counter will count 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 , “CASCADE-COUNT mode* through *Section , “VARIABLE-FREQUENCY PWM mode* for additional capabilities of this operating mode.

EDGE-COUNT mode

When the CNTMODE field is set to 010, the counter will count 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.

GATED-COUNT mode

When the CNTMODE field is set to 011, the counter will count 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 counter will count while the selected secondary input is low.

QUADRATURE-COUNT mode

When the CNTMODE field is set to 100, the counter will decode 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 413 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

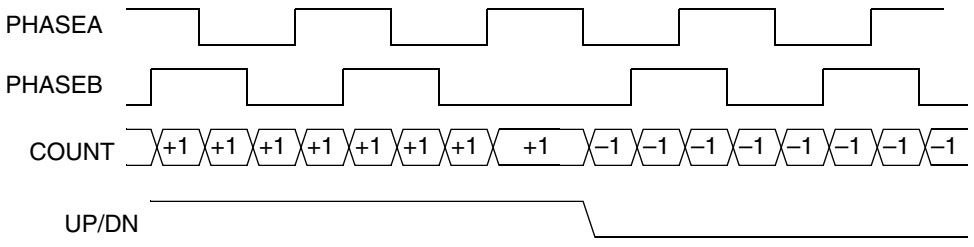


Figure 413. Quadrature incremental position encoder

SIGNED-COUNT mode

When the CNTMODE field is set to 101, the counter counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

TRIGGERED-COUNT mode

When the CNTMODE field is set to 110, the counter will begin counting the primary clock source after a positive transition (negative if SIPS = 1) of the secondary input occurs. The counting will continue until a compare event occurs or another positive input transition is detected. Subsequent secondary positive input transitions will continue to restart and stop the counting until a compare event occurs.

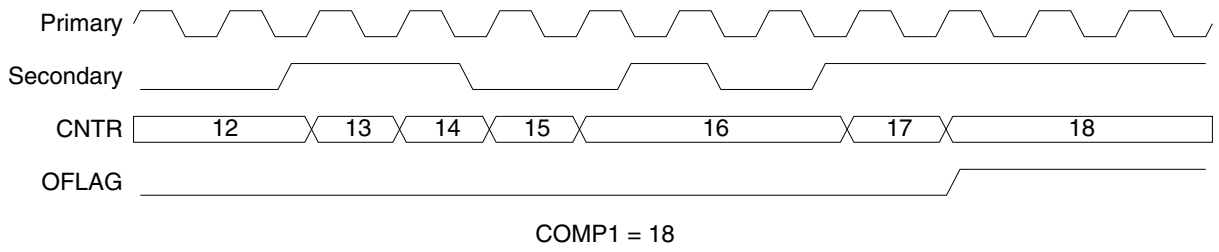


Figure 414. Triggered Count mode (length = 1)

ONE-SHOT mode

When the CNTMODE field is set to 110 and the counter 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 counter works in ONE-SHOT mode. If an external events causes the counter to count, when terminal count is reached, the output is asserted. This delayed output can be used to provide timing delays.

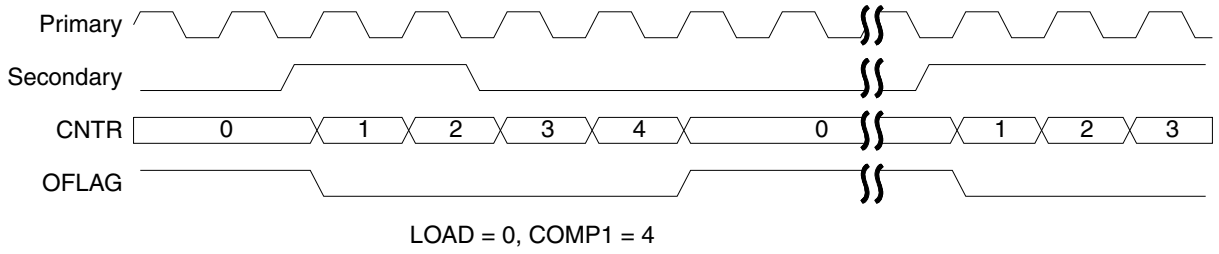


Figure 415. One-Shot mode (length = 1)

CASCADE-COUNT mode

When the CNTMODE field is set to 111, the counter's input is connected to the output of another selected counter. The counter will count up and down as compare events occur in the selected source counter. This cascaded or "daisy-chained" mode enables multiple counters 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 counter is counting up and it experiences a compare event, the counter will be incremented. If the selected source counter is counting down and it experiences a compare event, the counter will be decremented.

Either one or 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 will transition a clock later than a purely synchronous design.

One channel can be cascaded with any other channel, but channels cannot be cascaded more than two deep. Separate cascades of pairs of channels can be created. For example, channels 0 and 1 can be cascaded, and channels 5 and 4 cascaded separately. However, channels 0, 1, and 5 cannot be cascaded.

PULSE-OUTPUT mode

When the counter is set up with CNTMODE = 001, and the OFLAG OUTMODE is set to 1111 (gated clock output), and the ONCE bit is set, then the counter will output a pulse 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.

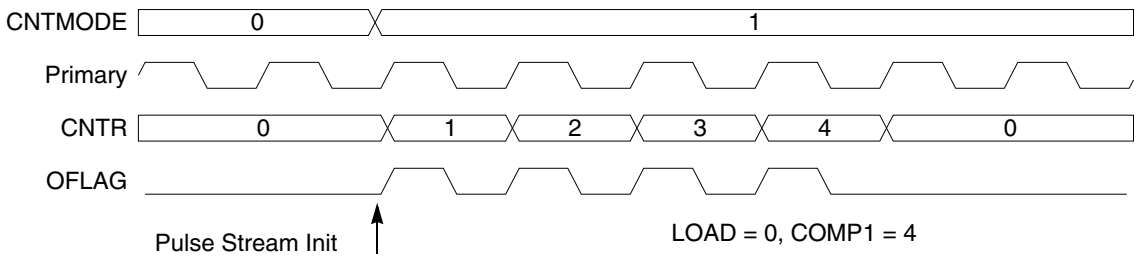


Figure 416. Pulse Output mode

FIXED-FREQUENCY PWM mode

When the counter is set up for CNTMODE = 001, count through roll-over (LENGTH = 0), continuous count (ONCE = 0) and the OFLAG OUTMODE is 0111 (set on compare, cleared on counter roll-over) 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.

VARIABLE-FREQUENCY PWM mode

When the counter is setup for CNTMODE = 001, count till compare (LENGTH = 1), continuous count (ONCE = 0) and the OFLAG OUTMODE 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 PWM current cycle is underway.

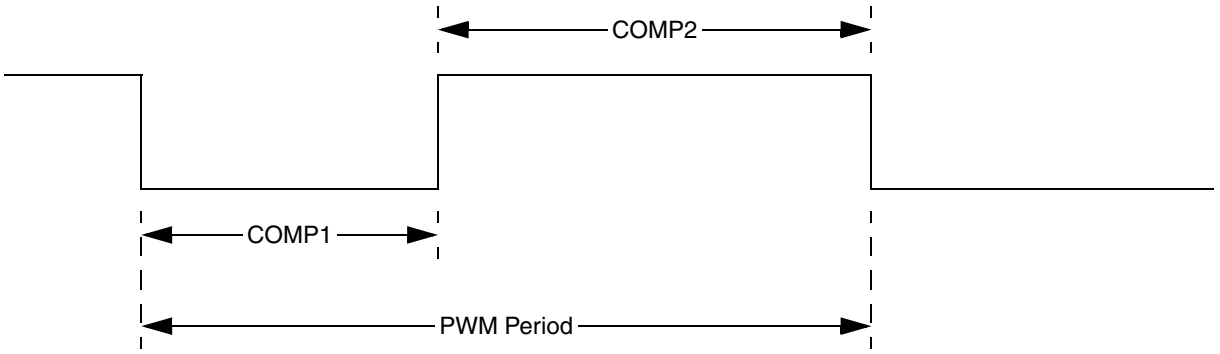


Figure 417. Variable PWM waveform

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 roll-over, and the COMP2 register should be set to the minimum count value or 0x0000 to indicate the minimum unsigned value prior to roll-under.

When the output mode is set to 0100, the OFLAG will toggle 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.

Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it will count to 0xFFFF or 0x0000, roll over, then begin counting toward the new value. The check is: CNTR = COMPx, *not* CNTR > COMP1 or CNTR < COMP2.

Using the CMPLD1 and CMPLD2 registers to preload compare values helps to minimize this problem.

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, the following methods are recommended.

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, there is the possibility 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 reinitialized to the value stored in the LOAD register. The compare load feature allows the user to calculate new compare values and store them in to 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 417*.

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 roll-over (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).

26.7.3 Other features

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.

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.

Input capture mode

Input capture measures 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 shown in *Figure 406*.

The arming logic controls the operation of the capture circuits to allow captures to be performed in a free-running (continuous) or one-shot fashion. In free-running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will 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 will be performed. If both capture circuits are enabled, capture circuit 0 is armed first. 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 capable of generating an interrupt to the CPU.

Master/Slave mode

Any timer channel can be assigned as a Master (MSTR = 1). A Master's compare signal can be broadcast 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.

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 time-out 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 0, an interrupt is asserted. The down counter is reloaded to the time-out 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.

26.8 Clocks

The eTimer module implements a protocol clock running at a frequency ≥ 120 MHz.

26.9 Interrupts

Each of the channels within the eTimer can generate an interrupt from several sources. The watchdog 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 385. 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 roll-over or roll-under
	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	WDF	WDFIE	Watchdog time-out interrupt	Watchdog has timed out
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel

26.10 DMA

Table 386. 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

27 Functional Safety

27.1 Introduction

This chapter describes the following modules that help add reliability to the SPC560P40/34.

- Register protection module
- Software watchdog timer (SWT)

27.2 Register protection module

27.2.1 Overview

The register protection 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 register protection module is located between the module under protection and the PBRIDGE. This is shown in *Figure 418*.

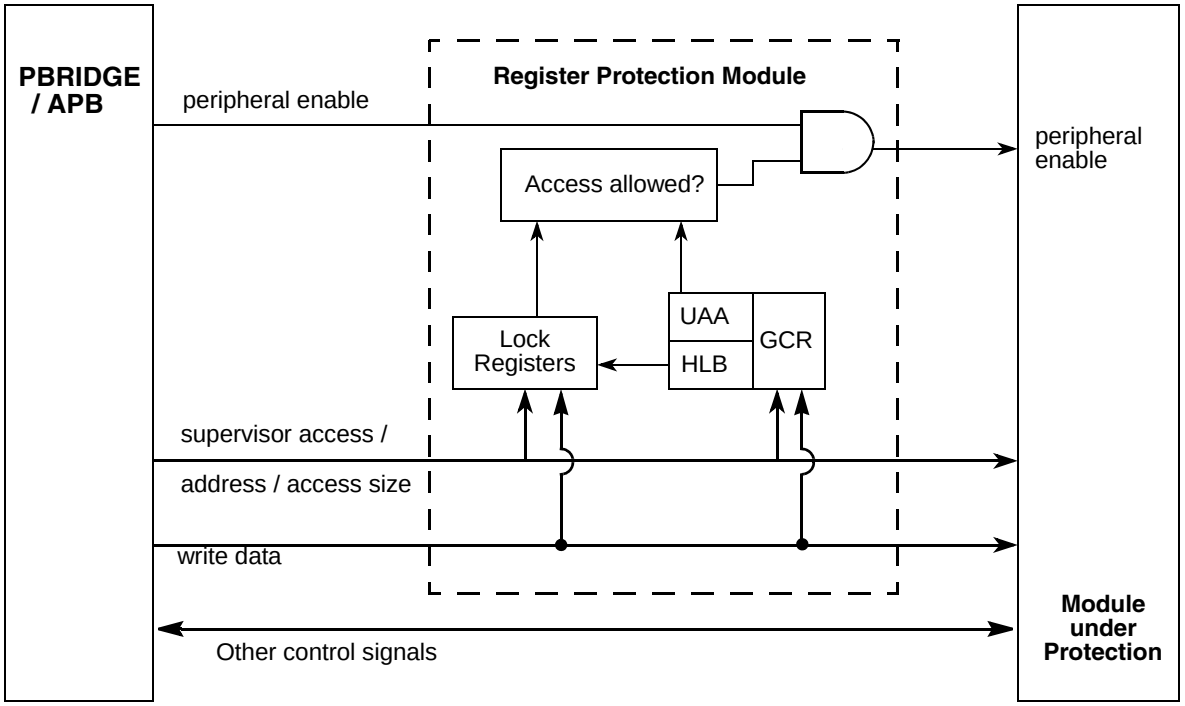


Figure 418. Register protection module block diagram

27.2.2 Features

The register protection module includes these features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock 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

27.2.3 Modes of operation

The register protection module is operable when the module under protection is operable.

27.2.4 External signal description

There are no external signals.

27.2.5 Memory map and registers description

This section provides a detailed description of the memory map of a module using the Register protection. The original 16 KB module memory space is divided into five areas as shown in *Figure 419*.

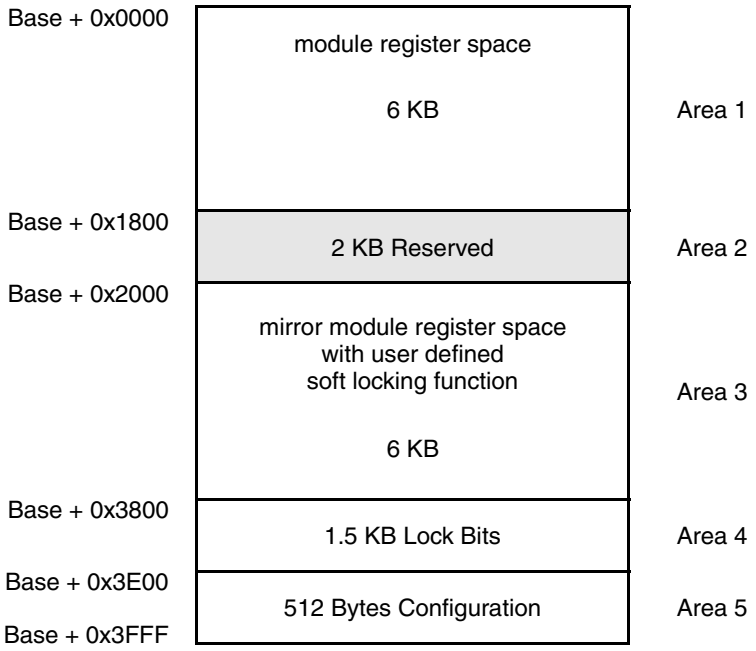


Figure 419. Register protection memory diagram

Area 1 is 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2, 2 KB starting at address 0x1800, is reserved.

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 will read/write the register at address X. As a side effect, a write access to address 0x2000 + X will set 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 will be identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable Soft Lock Bits will be 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 that prevents all further modifications to the Soft Lock Bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in areas 4 and 5 will result in a transfer error.

Register protection memory map

Table 387 shows the registers in the Safety Port.

Table 387. Register protection memory map

Offset from REG_ PROT_BASE (0xFFFE_8000)	Register	Location
0x0000–0x17FF	Module Register 0 (MR0)–Module Register 6143(MR6143)	*on page 27-740*
0x1800–0x1FFF	Reserved	
0x2000–0x37FF	Module Register 0 (MR0) + Set Soft Lock Bit 0 (LMR0)– Module Register 6143 (MR6143) + Set Soft Lock Bit 6143 (LMR6143)	*on page 27-740*
0x3800–0x3DFF	Soft Lock Bit Register 0 (SLBR0): Soft Lock Bits 0:3– Soft Lock Bit Register 1535 (SLBR1535): Soft Lock Bits 6140:6143	*on page 27-740*
0x3E00–0x3FFB	Reserved	
0x3FFC	Global Configuration Register (GCR)	*on page 27-741*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

Registers description

This section describes in address order all the register protection 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.

Module registers (MR0–6143)

This is the lower 6 KB module memory space that holds all the functional registers of the module that is protected by the register protection module.

Module Register and Set Soft Lock Bit (LMR0–6143)

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 a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR*n*[SLB*m*], according to the mapping described in *Table 388*.

Soft Lock Bit Register (SLBR0–1535)

These registers hold the Soft Lock Bits for the protected registers in memory area #1.

Figure 420. Soft Lock Bit Register (SLBR*n*)

Address: Base + 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

Table 388. 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[($n \times 4$) + 0] SLB1 can block accesses to MR[($n \times 4$) + 1] SLB2 can block accesses to MR[($n \times 4$) + 2] SLB3 can block accesses to MR[($n \times 4$) + 3] 0 Associated MR*n* byte is unprotected and writeable. 1 Associated MR*n* byte is locked against write accesses.

Table 389 gives some examples how SLBR*n*[SLB] and SLBR*n*[MR*n*] go together:

Table 389. 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
...	...

Global Configuration Register (GCR)

The Global Configuration Register (GCR) controls global configurations related to register protection.

Figure 421. Global Configuration Register (GCR)

Address: Base + 0x3FFC Access: Read Always; Supervisor write

	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

Table 390. 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 can not 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 Register protection module.

27.2.6 Functional description

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)

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 software is writing to.

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*] bit 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.

Change lock settings directly via area #4

In memory area #4 the lock bits are located. They can be modified by writing to them. Each SLBR*n*[SLB*m*] bit has a corresponding SLBR*n*[WE*m*] mask bit, which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 422 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]].

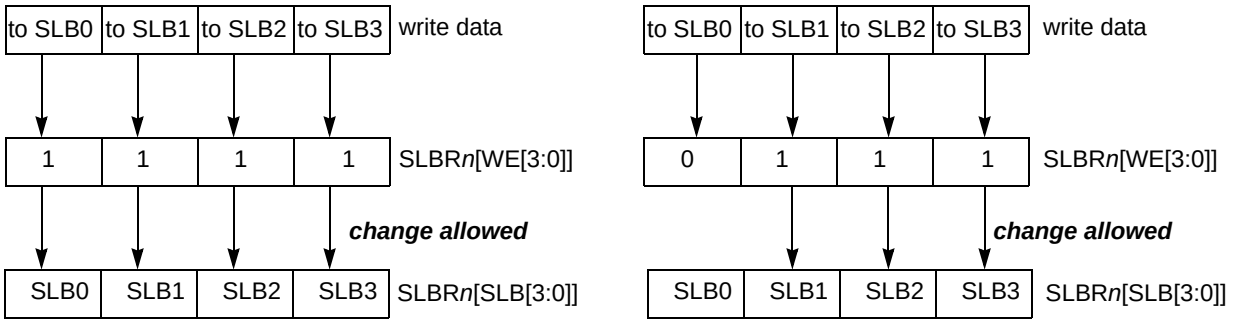


Figure 422. Change lock settings directly via area #4

Figure 422 showed four registers that can be protected 8-bit wise. In *Figure 423* registers with 16-bit protection and in *Figure 424* registers with 32-bit protection are shown.

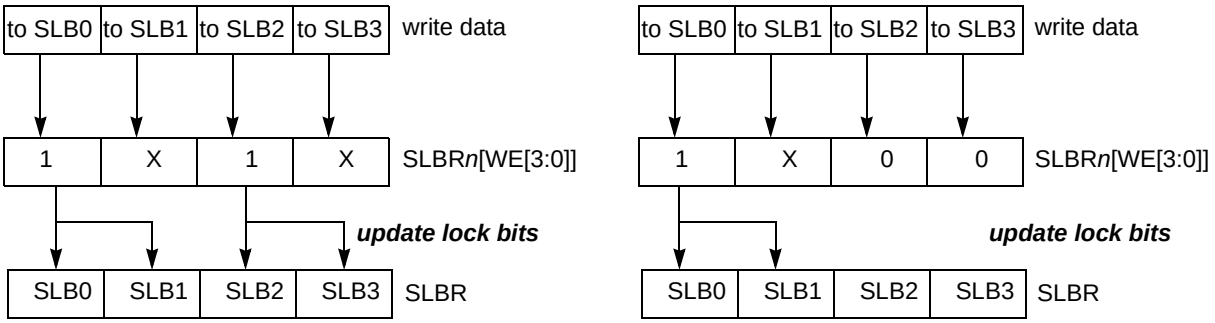


Figure 423. Change lock settings for 16-bit protected addresses

On the right side of *Figure 423* it is shown that the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB1] also. This is done as the address reflected by SLBR*n*[SLB0] is protected 16-bit wise. Note that in this case the write enable SLBR*n*[WE0] must be set while SLBR*n*[WE1] 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 423* the data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] and the data written to SLBR*n*[SLB2] is mirrored to SLBR*n*[SLB3] as for both registers the write enables are set.

In *Figure 424* a 32-bit wise protected register is shown. When SLBR*n*[WE0] is set the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB[3:1]] also. Otherwise SLBR*n*[SLB[3:0]] remains unchanged.

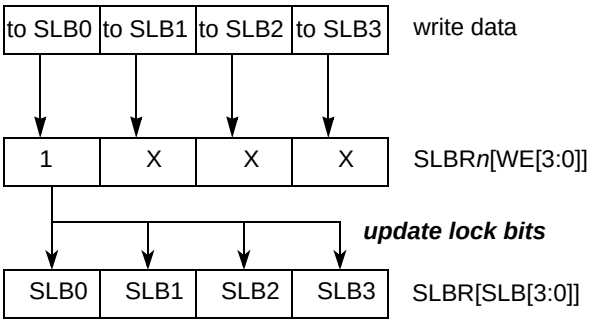


Figure 424. Change lock settings for 32-bit protected addresses

Figure 425 shows an example that has a mixed protection size configuration.

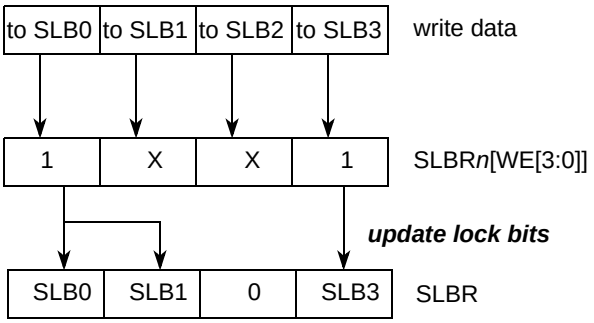


Figure 425. Change lock settings for mixed protection

The data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] as the corresponding register is 16-bit protected. The data written to SLBR*n*[SLB2] is blocked as the corresponding register is unprotected. The data written to SLBR*n*[SLB3] is written to SLBR*n*[SLB3].

Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so the mirrored module address space must be used. *Figure 426* shows one example.

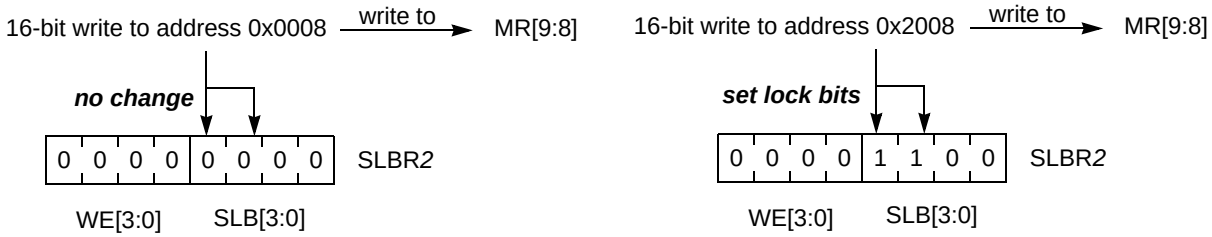


Figure 426. Enable locking via mirror module space (area #3)

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 423*).

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 423*).

Figure 427 shows an example where some addresses are protected and some are not.

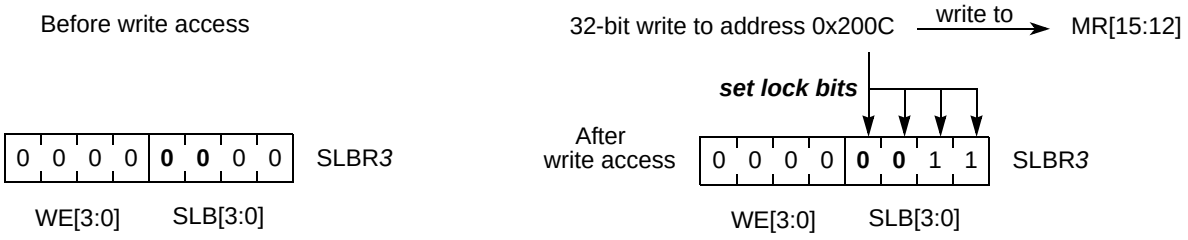


Figure 427. Enable locking for protected and unprotected addresses

In the example in *Figure 427*, 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 0b0.

Note: Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in *Section , "Change lock settings directly via area #4,* and *Section , "Enable locking via mirror module space (area #3)* is only possible as long as the GCR[HLB] bit is cleared. Once this bit is set the locking bits can no longer be modified until there was a system reset.

Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 419*.

1. If accessing area #1 or area #3, the protection module will pass on any access error from the underlying Module under Protection.
   - If user mode is not allowed, user writes to all areas will assert a transfer error and the writes will be blocked.
   - If accessing the reserved area #2, a transfer error will be asserted.
   - If accessing unimplemented 32-bit registers in area #4 and area #5 a transfer error will be asserted.
   - 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 will be blocked. Also the complete write operation to non-protected bytes in this word is ignored.
   - If writing to a Soft Lock Register in area #4 with the Hard Lock Bit being set a transfer error is asserted.
   - Any write operation in any access mode to area #3 while Hard Lock Bit GCR[HLB] is set

27.2.7 Reset

The reset state of each individual bit is shown in *Section , "Registers description.* In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in Supervisor Mode only.

27.3 Software Watchdog Timer (SWT)

27.3.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing sequence. Writing the sequence resets the timer to a specified time-out 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 time-out, a reset is always generated on a second consecutive time-out.

The SWT provides a window functionality. When this functionality is programmed, the servicing action should take place within the defined window. When occurring outside the defined period, the SWT will generate a reset.

27.3.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- The unique SWT counter clock is the undivided low power internal oscillator (IRC 16 MHz), no other clock source can be selected
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- The SWT is started on exit of power-on phase (RGM phase 2) to monitor flash boot sequence phase. It is then reset during RGM phase 3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG_EN).

27.3.3 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. As soon as out of stop mode, SWT will continue from the state it was before entering this mode.

Software watchdog is not available during stand-by. As soon as out of stand-by, the SWT behaves as in a usual “out of reset” situation.

27.3.4 External signal description

The SWT module does not have any external interface signals.

27.3.5 SWT memory map and registers description

The SWT programming model has six 32-bit registers, listed in *Table 391*. 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 and 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 and SWT_WN registers are read only.

Table 391. SWT memory map

Offset from SWT_BASE 0xFFF3_8000 (SWT_0) 0x8FF3_8000 (SWT_1)	Register	Location
0x0000	SWT_CR—SWT Control Register	*on page 27-747*
0x0004	SWT_IR—SWT Interrupt Register	*on page 27-749*
0x0008	SWT_TO—SWT Time-Out register	*on page 27-749*
0x000C	SWT_WN—SWT Window Register	*on page 27-750*
0x0010	SWT_SR—SWT Service Register	*on page 27-751*
0x0014	SWT_CO—SWT Counter Output register	*on page 27-751*
0x0018	SWT_SK—SWT Service Key register	*on page 27-752*
0x001C–0x03FF	Reserved	

SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT_CR[WEN] bit during the boot process. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 428. SWT Control Register (SWT_CR)

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

The default reset value for SWT_CR is 0xFF00_011B, corresponding to MAP1 = 1 (only data bus access allowed), RIA = 1 (reset on invalid SWT access), SLK = 1 (soft lock), CSL = 1 (IRC clock source for counter), FRZ = 1 (freeze on debug), WEN = 1 (watchdog enable). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is 0.

Table 392. SWT_CR field descriptions

Field	Description
MAP*n*	Master Access Protection for Master *n*. The platform bus master assignments are device specific. 0 Access for the master is disabled. 1 Access for the master is enabled.

Table 392. SWT_CR field descriptions (continued)

Field	Description
KEY	Keyed Service Mode 0 Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 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 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 time-out. 1 Generate an interrupt on an initial time-out, reset on a second consecutive time-out.
HLK	Hard Lock This bit is only cleared at reset. 0 SWT_CR, SWT_TO and SWT_WN are read/write registers if SLK = 0. 1 SWT_CR, SWT_TO and SWT_WN 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 and SWT_WN are read/write registers if HLK = 0. 1 SWT_CR, SWT_TO and SWT_WN are read only registers.
CSL	Clock Selection Selects the internal 16 MHz IRC oscillator clock that drives the internal timer. CSL bit can be written.The status of the bit has no effect on counter clock selection on the device. 0 System clock. (Not applicable in SPC560P40/34). 1 Oscillator clock.
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.

SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Figure 429. SWT Interrupt Register (SWT_IR)

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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 393. SWT_IR field descriptions

Field	Description
TIF	Time-out 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 time-out.

SWT Time-Out register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Figure 430. SWT Time-Out register (SWT_TO)

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

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

The default counter value (SWT_TO_RST) is 0x0003_A980, which corresponds to approximately 15 ms with the 16 MHz IRC clock.

Table 394. SWT_TO field descriptions

Field	Description
WTO	Watchdog time-out period in clock cycles An internal 32-bit down counter is loaded with this value or 0x0100, whichever is greater when the service sequence is written or when the SWT is enabled.

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.

Figure 431. SWT Window register (SWT_WN)

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															
Reset	0	0	0	0	0	0	0	0	0	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															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 395. 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.

SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service sequence writes used to reset the watchdog timer.

Figure 432. SWT Service Register (SWT_SR)

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	WSC															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 396. SWT_SR field descriptions

Field	Description
WSC	Watchdog Service Code This field services the watchdog and clears the SWT_CR[SLK] soft lock bit. To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the SWT_CR[SLK] soft lock bit, the value 0xC520 followed by 0xD928 is written to the WSC field.

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.

Figure 433. SWT Counter Output register (SWT_CO)

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

	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

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

Table 397. 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 as many as 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.

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.

Figure 434. SWT Service Register (SWT_SR)

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																
Reset	0	0	0	0	0	0	0	0	0	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

Table 398. SWT_SR 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}.

27.3.6 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 if a bus transaction fails to terminate. It includes a a control register (SWT_CR), an interrupt register (SWT_IR), time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR) and a counter output register (SWT_CO).

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 specific1 (enabled). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is 0. 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 time-out period in clock cycles unless the value is less than 0x0100, in which case the time-out period is set to 0x0100. This time-out period is

loaded into an internal 32-bit down counter when the SWT is enabled and a valid service sequence is written. The SWT_CR[CSL] bit selects which clock (system or oscillator) drives the down counter.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO and SWT_WN 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 the watchdog servicing sequence. The service sequence is a write of 0xA602 followed by a write of 0xB480 to the SWT_SR[WSC] field. Writing the service sequence loads the internal down counter with the time-out period. There is no timing requirement between the two writes. The service sequence logic ignores unlock sequence writes and recognizes the 0xA602, 0xB480 sequence regardless of previous writes. 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 as many as 3 system plus 7 counter clock cycles.

If window mode is enabled (SWT_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out 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 time-out 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 as many as 3 system plus 4 counter clock cycles.

The SWT_CR[ITR] interrupt then reset bit controls the action taken when a time-out occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out 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 as many as 6 system plus 8 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 time-out value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

28 Fault Collection Unit (FCU)

28.1 Introduction

The Fault Collection Unit (FCU) module provides functional safety to the device.

28.1.1 Overview

The FCU provides a central capability to collect faults reported by the individual modules of the device. It represents the minimum blocking unit to develop a coherent safety strategy for the chassis family. Selected critical faults are reported to the external device via output pins, if no recovery is provided by the device. The operation of the FCU is independent from the CPU. The FCU provides an independent fault reporting mechanism even in case the CPU behavior is abnormal. The FCU always starts up in init mode. As long as the FCU remains in init mode, testing of the FCU logic (for dormant fault detection) can be performed under software control.

The FCU is developed to increase the level of safety of the system/MCU level.

Functional safety features of the FCU include:

- It is an independent module: If other control modules are behaving abnormally, the user can still trigger actions to prevent a critical situation.
- Collection and external reporting of faults occurring on the device
- Centralized fault collection
- Each fault cause can be treated in a different way
  - No action
  - Alarm—allows hardware or software to recover from fault
  - Fault—communicates directly to an external device that something went wrong
- Three different output protocols available
- Possible to inject fake faults on user request during initialization phase to test the peripheral (dormant fault detection)

General description

The FCU is logically divided in three blocks:

- Input unit—captures faults reported from the device
- Control unit—implemented by a finite state machine (see *Figure 449* for details)
- Output unit—generates output signals

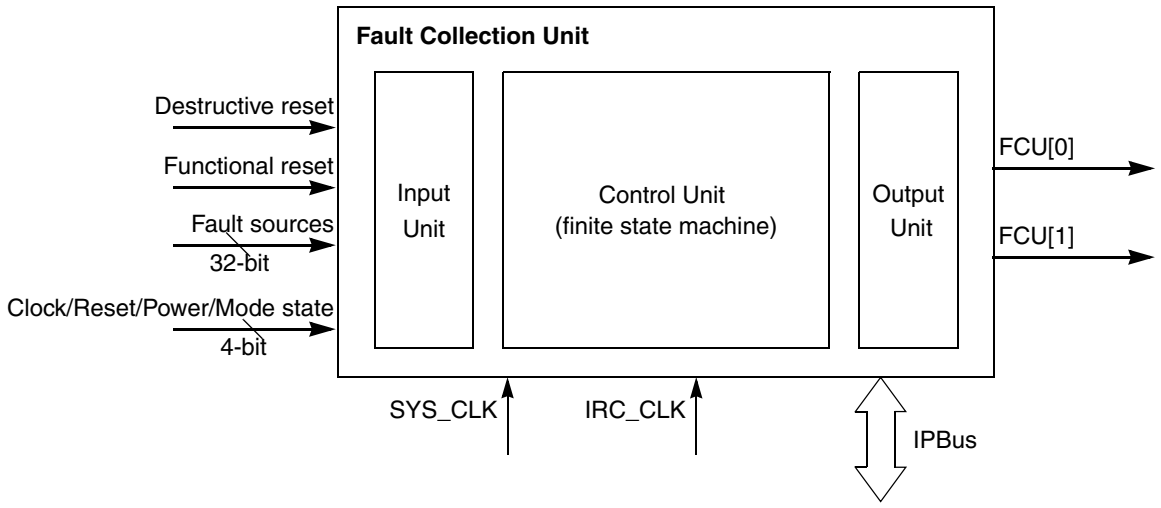


Figure 435. Fault Collection Unit (FCU) block diagram

Figure 436 shows the flow chart of FCU fault handling.

The FCU module and its state machine run on the system clock, making the two modules synchronous. The high speed RC clock (16 MHz) is used only in the Alarm state, in order to compute a deterministic timeout.

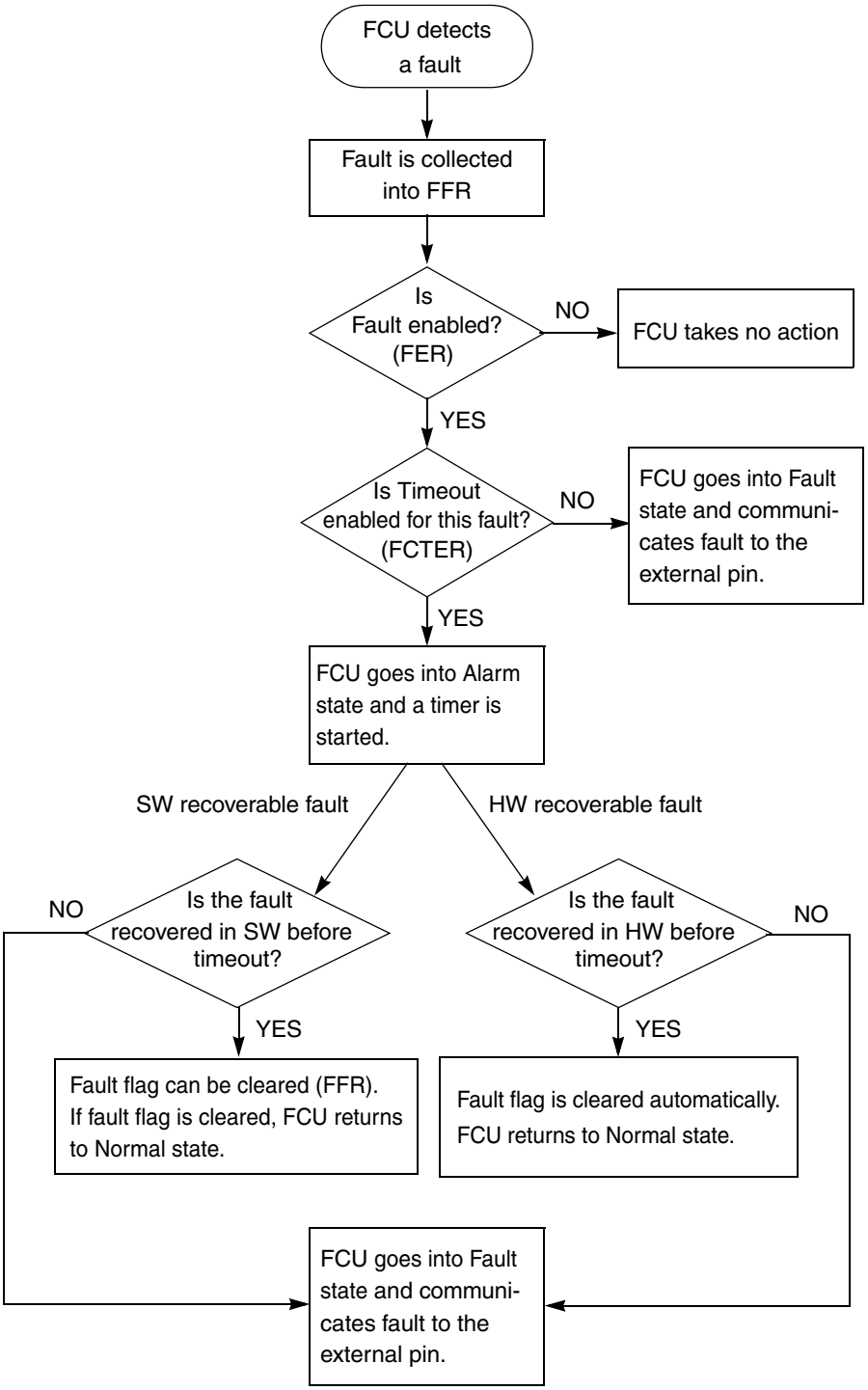


Figure 436. FCU fault handling

28.1.2 Features

The FCU includes the following features:

- Collection of critical faults
- Reporting of selected critical faults to external pins
- Fault flag status kept over non-destructive reset for later analysis (in a “Freeze” register)
- Continuous and synchronous latch of MC state
- MC state kept over non-destructive reset for later analysis (in a “Freeze” register)
- 4-state (Init, Normal, Alarm, Fault) finite state machine
- Different actions can be taken depending on fault type
- Selectable protocols for fault signal indication in Fault state (dual-rail, time-switching, bi-stable)
- Programmable clock prescaler for time-switching output signal generation
- Protection mechanism to avoid unwanted clearing of fault flags
- Internal logic testing, by using a fake fault generator during initialization phase

28.1.3 Modes of operation

This section describes the basic functional modes of the FCU module.

Normal mode

In Normal operation, the FCU captures all the faults in real time and processes them according to the fault type and the configuration set by the user.

Test mode

Test mode provides a testing mechanism of the FCU (for dormant fault detection). Test mode can be entered during the configuration (Init) phase. The user can write to the Fake Fault Generation Register (FCU_FFGR) and can check the behavior while staying in Test mode. During Test mode, the state machine behaves normally. In Test mode, real faults are not detected.

The user can inject fake faults by writing to the FCU_FFGR during Test mode to test the peripheral. Fault flags are cleared when Test mode is exited. Asynchronous software faults written to the FCU_FFGR during Init phase will not be latched. In order to test software faults, the FCU_FFGR needs to be cleared during Init phase and written to during Normal phase.

28.2 Memory map and register definition

The following sections define the FCU memory map, register layout and functionality.

28.2.1 Memory map

Table 399. FCU memory map

Offset from FCU_BASE (0xFFE6_C000)	Register	Location
0x0000	Module Configuration Register (FCU_MCR)	*on page 28-760*
0x0004	Fault Flag Register (FCU_FFR)	*on page 28-761*
0x0008	Frozen Fault Flag Register (FCU_FFFR)	*on page 28-763*
0x000C	Fake Fault Generation Register (FCU_FFGR)	*on page 28-764*
0x0010	Fault Enable Register (FCU_FER)	*on page 28-765*
0x0014	Key Register (FCU_KR)	*on page 28-765*
0x0018	Timeout Register (FCU_TR)	*on page 28-766*
0x001C	Timeout Enable Register (FCU_TER)	*on page 28-767*
0x0020	Module State Register (FCU_MSR)	*on page 28-767*
0x0024	Microcontroller State Register (FCU_MCSR)	*on page 28-768*
0x0028	Frozen MC State Register (FCU_FMCSR)	*on page 28-770*
0x002C–0x3FFF	Reserved	

28.2.2 Register summary

Table 400 shows the register summary.

Table 400. Register summary

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
0x0000_0000 FCU_MCR	R	MCL	TM[1:0]		0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	PS[1:0]		FOM[1:0]		FOP[5:0]					
	W																
0x0000_0004 FCU_FFR	R	SRF0	SRF1	SRF2	SRF3	SRF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	HRF15	HRF14	HRF13	HRF12	HRF11	HRF10	HRF9	HRF8	HRF7	HRF6	HRF5	HRF4	HRF3	HRF2	HRF1	HRF0
	W																

Table 400. Register summary (continued)

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
0x0000_0008 FCU_FFFR	R	FR SRF0	FR SRF1	FR SRF2	FR SRF3	FR SRF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	FRHRF15	FRHRF 14	FRHRF 13	FR HRF12	FR HRF11	FR HRF10	FR HRF9	FR HRF8	FR HRF7	FR HRF6	FR HRF5	FR HRF4	FR HRF3	FR HRF2	FR HRF1	FR HRF0
	W																
0x0000_000C FCU_FFGR	R	FSRF0	F1RF3	FSRF2	FSRF3	FSRF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	F HRF15	F HRF 14	F HRF 13	F HRF12	F HRF11	F HRF10	F HRF9	F HRF8	F HRF7	F HRF6	F HRF5	F HRF4	F HRF3	F HRF2	F HRF1	F HRF0
	W																
0x0000_0010 FCU_FER	R	ESF0	ESF1	ESF2	ESF3	ESF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	EHF15	EHF14	EHF13	EHF12	EHF11	EHF10	EHF9	EHF8	EHF7	EHF6	EHF5	EHF4	EHF3	EHF2	EHF1	EHF0
	W																
0x0000_0014 FCU_KR	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	0
	W																
0x0000_0018 FCU_TR	R	TR[31:16]															
	W																
	R	TR[15:0]															
	W																
0x0000_001C FCU_TER	R	TESF0	TESF1	TESF2	TESF3	TESF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	TEHF15	TEHF14	TEHF13	TEHF12	TEHF11	TEHF10	TEHF9	TEHF8	TEHF7	TEHF6	TEHF5	TEHF4	TEHF3	TEHF2	TEHF1	TEHF0
	W																
0x0000_0020 FCU_MSR	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	S0	S1	S2	S3
	W																

Table 400. Register summary (continued)

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
0x0000_0024 FCU_MCSR	R	0	0	0	0	0	0	0	0	0	0	0	0	MCPS[3:0]			
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	MCAS[3:0]			
	W																
0x0000_0028 FCU_FMCSR	R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCPS[3:0]			
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCAS[3:0]			
	W																

28.2.3 Register descriptions

Module Configuration Register (FCU_MCR)

The FCU_MCR does the following:

- Locks the configuration and lets the FCU go into Normal behavior state
- Enters Test mode (only possible during the Init state) but can be exited during any phase
- Configures protocol, prescaler, and polarity for FCU output signals (only possible before configuration is locked)

In Test mode, the FCU_FFGR can be accessed to emulate software/hardware faults. Fake faults can be generated only when Test mode is entered.

In Test mode, output pin(s) can be enabled or disabled, depending on the value of field TM[1:0]. To exit from Test mode, field TM must be written either ‘00’ or ‘11’. While exiting the Test mode, the FCU must return to the Init state and automatically clear all the fault flags.

Figure 437. Module Configuration Register (FCU_MCR)

Address: 0x0000 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MCL	TM[1:0]		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	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	PS[1:0]		FOM[1:0]		FOP[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Table 401. FCU_MCR field description

Field	Description
0 MCL	Module Configuration Lock 0: Configuration not locked, FCU remains in Init state 1: Configuration locked, FCU moves to Normal state
1-2 TM[1:0]	Test Mode 00: Test Mode not entered 01: Test Mode entered (fake faults can be generated), output pins disabled 10: Test Mode entered (fake faults can be generated), output pins enabled 11: Test Mode not entered
22-23 PS[1:0]	Polarity select 00: FCU[0] has normal polarity, FCU[1] has normal polarity. 01: FCU[0] has inverted polarity, FCU[1] has normal polarity. 10: FCU[0] has normal polarity, FCU[1] has inverted polarity. 11: FCU[0] has inverted polarity, FCU[1] has inverted polarity.
24-25 FOM[1:0]	Fault output mode selection 00: Dual-Rail (default state) 01: Time Switching 10: Bi-Stable 11: Reserved
26-31 FOP[5:0]	Fault Output Prescaler 000000: Input clock frequency is divided by [4096 × (0 + 1) × 2], where 4096 is a fixed prescaler. 000001: Input clock frequency is divided by [4096 × (1 + 1) × 2], where 4096 is a fixed prescaler. 000010: Input clock frequency is divided by [4096 × (2 + 1) × 2], where 4096 is a fixed prescaler. 000011: Input clock frequency is divided by [4096 × (3 + 1) × 2], where 4096 is a fixed prescaler. 000100: Input clock frequency is divided by [4096 × (4 + 1) × 2], where 4096 is a fixed prescaler. ... Default at reset is 0x07 (input clock frequency is divided by [4096 × (7 + 1) × 2]).

Fault Flag Register (FCU_FFR)

The FCU_FFR contains the latched fault indication coming from the device. The FCU reacts on faults only if the respective enable bit for a fault is set in the Fault Enable Register (FCU_FER). In this case, the Alarm or Fault state is entered, depending on the user's selection. To enter Alarm state, the bit for the fault has to be set in the Timeout Enable Register (FCU_TER), otherwise Fault state is entered and output pins are communicating the internal fault.

No faults are latched in Init state (refer to state machine *Figure 449*).

The FCU_FFR is copied into Frozen Fault Flag Register (FCU_FFFR) only when the FCU goes into Fault state.

Each single flag can be cleared:

- In hardware, if the flag disappears due to hardware intervention. Bits 16:31 of the FCU_FFR store hardware-recoverable flags.
- In software, if the user application can recover from a faulty situation. Bits 0:4 of the FCU_FFR store software-recoverable flags.

Notice that software recoverable fault flags must be kept high also if the relative signal does not show anymore a fault. Hardware recoverable fault flags are updated in real time. Reset requests are assumed as hardware-recoverable.

In order to clear a flag in the FCU_FFR via software, the application should access first time the Key Register (FCU_KR) by writing the value of a key (0x618B_7A50) second time resetting the appropriate flag. The following errors are ignored:

- Wrong key inserted in FCU_KR
- Attempt to clear a hardware recoverable flag

If user software tries to clear an already cleared software-recoverable flag, SRF0 is set.

Figure 438. Fault Flag Register (FCU_FFR)

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

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

In addition to the detailed field descriptions in *Table 402*, *Table 403* provides the hardware/software fault descriptions.

Table 402. FCU_FFR field descriptions

Field	Description
0:4 SRF0– SRF4	Software Recoverable Fault [0:4] 0: No error latched 1: Error latched
16:31 HRF15– HRF0	Hardware Recoverable Fault [15:0] 0: No error latched 1: Error latched

Table 403. Hardware/software fault description

Label	Module	Fault type
HRF0	Core	Checkstop mode entered
HRF1	Core	z0h core reset output
HRF2	CMU_0	Loss of crystal
HRF3	FMPLL_0	Loss of lock
HRF4	CMU_0	Frequency out of range

Table 403. Hardware/software fault description (continued)

Label	Module	Fault type
HRF5	Not used	
HRF6	Not used	
HRF7	Flash	Flash Fatal Error
HRF8	SWT	SW Watchdog reset
HRF9	JTAG	JTAG reset (TAP controller)
HRF10	PMU	Comparators
HRF11	PMU	LVD 4.5
HRF12	PMU	LVD 2.7 VREG
HRF13	PMU	LVD 2.7 FLASH
HRF14	PMU	LVD 2.7 I/O
HRF15	PMU	LVD 1.2 digital
SRF0	FCU	FCU error
SRF1	FCU	Software-triggered error
SRF2	SRAM	ECC multi-bit error
SRF3	Data Flash	ECC multi-bit error
SRF4	Code Flash	ECC multi-bit error

Frozen Fault Flag Register (FCU_FFFR)

The Fault Flag Register (FCU_FFR) is copied into Frozen Fault Flag Register (FFFR) every time the FCU goes into Fault state, in order to take a snapshot of the fault flags.

The bit description of the FCU_FFFR is the same as that of the FCU_FFR. The FCU_FFR is read/clear whereas the FCU_FFFR is read-only. See *Figure 439* and *Table 404* for details.

Figure 439. Frozen Fault Flag Register (FCU_FFFR)

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRSRF0	FRSRF1	FRSRF2	FRSRF3	FRSRF4	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	FRHRF15	FRHRF14	FRHRF13	FRHRF12	FRHRF11	FRHRF10	FRHRF9	FRHRF8	FRHRF7	FRHRF6	FRHRF5	FRHRF4	FRHRF3	FRHRF2	FRHRF1	FRHRF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 404 provides a detailed bit description. *Table 403* provides a detailed list of recoverable faults.

Table 404. FCU_FFFR field descriptions

Field	Description
0:4 FRSRF0– FRSRF4	Software Recoverable Fault 0: No error latched 1: Error latched
16:31 FRHRF15 – FRHRF0	Hardware Recoverable Fault 0: No error latched 1: Error latched

Fake Fault Generation Register (FCU_FFGR)

The FCU_FFGR allows the user to emulate a software/hardware recoverable fault in order to test the FCU logic. Once a bit in the FCU_FFGR is set, the FCU should behave exactly in the same way after detecting a real fault. This register can be accessed only when Test mode is entered (check TM field in FCU_MCR). In Test mode, real faults are not detected and fake faults for SRF0 and SRF1 cannot be generated.

Figure 440. Fake Fault Generation Register (FCU_FFGR)

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

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

Table 405. FCU_FFGR field description

Field	Description
0:4 FSRF0– FSRF4	Fake Software Recoverable Fault[0:4] 0: No error latched 1: Error latched
16:31 FHRF15– FHRF0	Fake Hardware Recoverable Fault[15:0] 0: No error latched 1: Error latched

Fault Enable Register (FCU_FER)

When a fault occurs, the FCU goes into either Alarm or Fault state (state is selected in the FCU_TER), if the respective fault enable bit is set in the Fault Enable Register (FCU_FER). This register can be configured only during the Init phase before the configuration is locked.

Figure 441. Fault Enable Register (FCU_FER)

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

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

Table 406. FCU_FER field descriptions

Field	Description
0:4 ESF0–ESF4	Enable Software Recoverable Fault 0: FCU takes no action on Software recoverable Fault [0:4] 1: FCU goes into Alarm/Fault state on Software recoverable Fault [0:4] Note. ESF1 not implemented or usable on this device
16:31 EHF15–EHF0	Enable Hardware Recoverable Fault 0: FCU takes no action on Hardware recoverable Fault [15:0] 1: FCU goes into Alarm/Fault state on Hardware recoverable Fault [15:0]

Key Register (FCU_KR)

In order to clear a software fault flag in the Alarm state, the FCU_KR must be set to the following value: 0x618B_7A50. Then the software fault flag can be cleared.

User software can clear the software fault flag only after the following instruction is executed. If something else is done, instead of clearing the flag, the key must be re-entered.

Any wrong key error is ignored.

Figure 442. Key Register (FCU_KR)

Address: Base + 0x0014 Access: User read-only, Supervisor 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	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Timeout Register (FCU_TR)

Once the FCU goes into Alarm state, a fault can be recovered before the timeout elapses. This timeout should be long enough for hardware or software to recover from the fault. If the fault is not recovered before the timeout elapses, the FCU goes into Fault state.

Figure 443. Timeout Register (FCU_TR)

Address: Base + 0x0018 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TR[0:15]															
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	TR[31:16]															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 407. FCU_TR field descriptions

Field	Description
0-31 TR	FCU Timeout 00000: Timeout is one clock (16 MHz) cycle 00001: Timeout is one clock (16 MHz) cycle 00002: Timeout is two clock (16 MHz) cycles 00003: Timeout is three clock (16 MHz) cycles ... Default at reset is 0x0000_FFFF. Timeout is 65,535 clock cycles (about 4.1 ms at 16 MHz, high speed RC clock)

Timeout Enable Register (FCU_TER)

Once a specific fault is enabled, the user can select to move to Alarm or Fault state when a fault occurs. A timeout enable has no effect if the related fault enable flag is not set.

Figure 444. Timeout Enable Register (FCU_TER)

Address: Base + 0x001C Access: User read/write, Supervisor read/write

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

Table 408. FCU_TER field descriptions

Field	Description
0:4 TESF0–TESF4	Timeout Enable for Software Recoverable Fault 0: FCU goes into Fault state on Software recoverable Fault[0:4] 1: FCU goes into Alarm state on Software recoverable Fault[0:4] Note. TESF1 not implemented or usable on this device
16:31 TEHF15–TEHF0	Timeout Enable for Hardware Recoverable Fault 0: FCU goes into Fault state on Hardware recoverable Fault[15:0] 1: FCU goes into Alarm state on Hardware recoverable Fault[15:0]

Module State Register (FCU_MSR)

The FCU_MSR indicates the current state of the FCU. Only one of these bits can be set at a time.

Figure 445. Module State Register (FCU_MSR)

Address: Base + 0x0020 Access: User read-only, Supervisor 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	S3	S2	S1	S0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 409. FCU_MSR field descriptions

Field	Description
28 S3	When this bit is set, the FCU is in the Fault state.
29 S2	When this bit is set, the FCU is in the Alarm state.
30 S1	When this bit is set, the FCU is in the Normal state.
31 S0	When this bit is set, the FCU is in the Init state.

Microcontroller State Register (FCU_MCSR)

The FCU_MCSR indicates the current state of the microcontroller in the MCSA field and the previous state (before state change) in the MCSP field. The contents of this register are copied into the Frozen MC State Register (FCU_FMCSR) when the FCU goes into Fault state.

Figure 446. MC State Register (FCU_MCSR)

Address: Base + 0x0024 Access: User read-only, Supervisor 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	MCPS[3:0]			
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	MCAS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 410. FCU_MCSR field description

Field	Description
12:15 MCPS[3:0]	MC Previous State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved
28:31 MCAS[3:0]	MC Actual State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved

Frozen MC State Register (FCU_FMCSR)

The FCU_MCSR is copied into the FCU_FMCSR each time the Fault state is entered. The FCU_FMCSR bit description is the same as that of the FCU_MCSR.

Figure 447. Frozen MC State Register (FCU_FMCSR)

Address: Base + 0x0028 Access: User read-only, Supervisor 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	FRMCPS[3:0]			
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	FRMCAS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 411. FCU_FMCSR field description

Field	Description
12:15 FRMCPS [3:0]	MC Previous State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved

Table 411. FCU_FMCSR field description

Field	Description
28:31 FRMCAS [3:0]	MC Actual State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved

28.3 Functional description

The FCU module contains six functional blocks as shown in *Figure 448*.

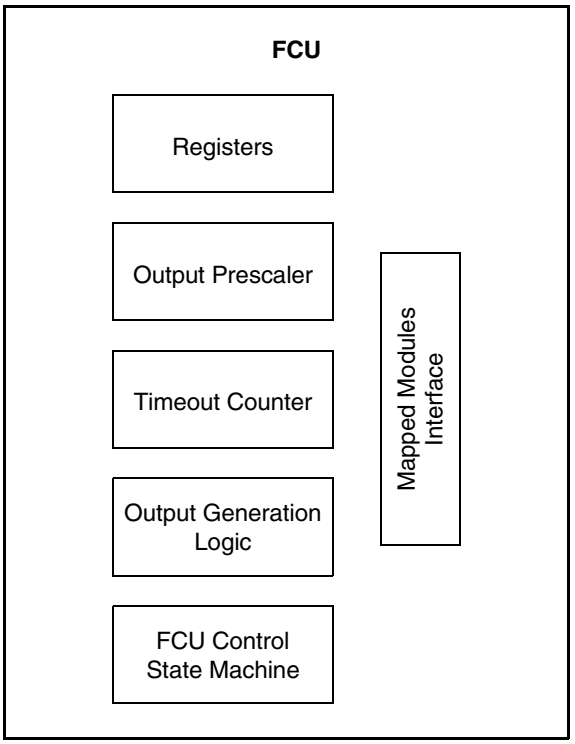


Figure 448. Functional block diagram

The register block implements all the FCU registers including input capture logic. The interface implements the read functionality and generated write and register enable signals. The timeout counter implements the counter to calculate timeout to switch from Alarm state to Fault state. The output prescaler module generates the prescaled clock to be used to generate output sequence. The FCU control implements the FCU state machine. The output generation logic generates two external output signals according to the output generation protocol.

28.3.1 State machine

The FCU provides four states: Init, Normal, Alarm, and Fault.

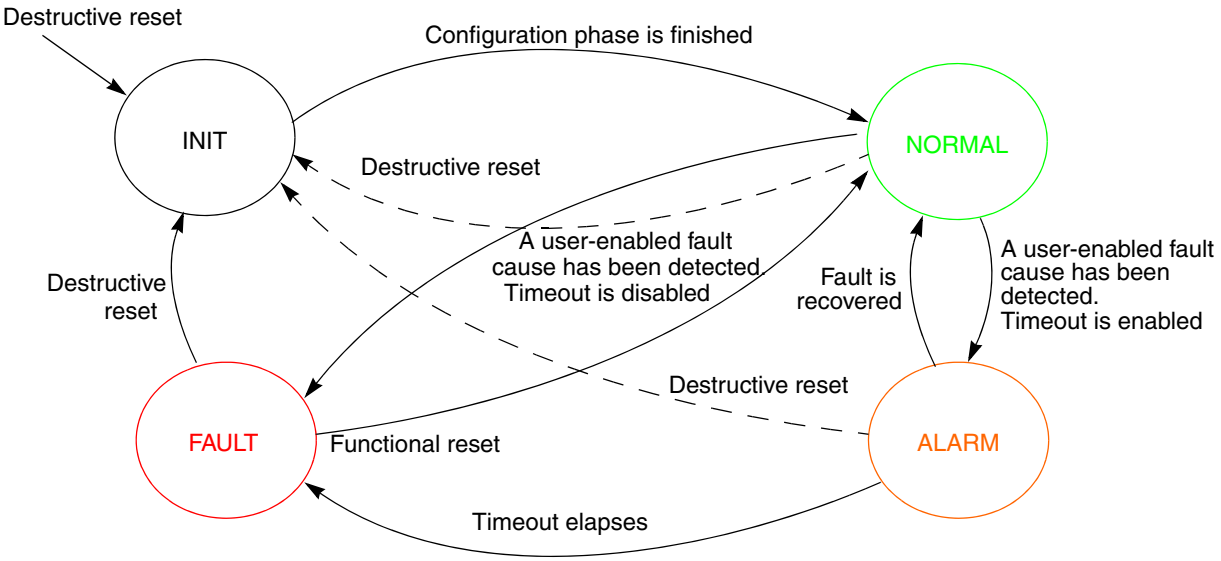


Figure 449. Finite state machine

The Init state is entered after any destructive reset assertion. Once the FCU goes into Fault state, both functional and destructive reset assertion lets the FCU move into Init state.

Once a configuration is locked (see *Section , "Module Configuration Register (FCU_MCR)"*), the FCU goes into Normal state. Then, when a fault occurs, the FCU can move into Alarm/Fault states depending on the Fault Enable Register (FCU_FER). The Fault Flag Register (FCU_FFR) stores any fault that has occurred, even if the FCU is not entering into the Alarm or the Fault state.

When the FCU is in Alarm state, a timer starts counting up to a fixed timeout (see *Section , "Timeout Register (FCU_TR)"*) before going into Fault state.

When the FCU goes into Fault state, the status of the FCU_MCR is copied into the FCU_FMCSR. Also, the FCU_FFR is copied into the FCU_FFFR.

The FCU can be tested by setting field TM[1:0] in the FCU_MCR during initialization. In this way, the FCU_FFGR can be accessed and faults can be emulated, since FCU behavior is the same whether real or fake faults occur. Optionally, the output pins can be disabled in Test mode (by setting the TM field to '01'). To exit from Test mode, the TM field must be set to '00' or '11'.

After a functional reset, the FCU_FFR (not the Frozen Fault Flag Register (FCU_FFFR)) must be cleared and the FCU must return to Normal state.

28.3.2 Output generation protocol

The FCU provides two external output signals. The FCU supports different protocols for fault indication to the external device. Both external signals are used only in dual-rail protocol. In other protocols, the second output is the inverted version of the first output.

In all the protocols, depending on the polarity field (PS) in the MCR, the outputs can be normal polarity (if PS = 0) or inverted polarity (if PS = 1). The LSB of PS sets the polarity of the first signal; the MSB sets the polarity of the second signal.

Dual-rail protocol

Dual-rail encoding is an alternate method for encoding bits. In contrast to classical encoding, where each wire carries a single bit-value, dual-rail encoded circuits use two wires to carry each bit. *Table 412* summarizes the encoding.

Table 412. Dual-rail coding

Logical value	Dual-rail encoding (FCU[0], FCU[1])
non-faulty	(0,1)
non-faulty	(1,0)
faulty	(0,0)
faulty	(1,1)

As long as the FCU is in Normal or Alarm state, the output shows a “non-faulty” signal. FCU[0], FCU[1] toggles between (0,1) and (1,0) with a given frequency, set in the FOP field of the MCR. By default, this value is f = 976 Hz @ 64 MHz (about 1 kHz, see *Equation 34*). The same frequency is used to show a faulty situation (FCU in Fault state).

In the Init state, output pins are set as high impedance by clearing the OBE bits for each pad in its respective PCR in the SIUL module.

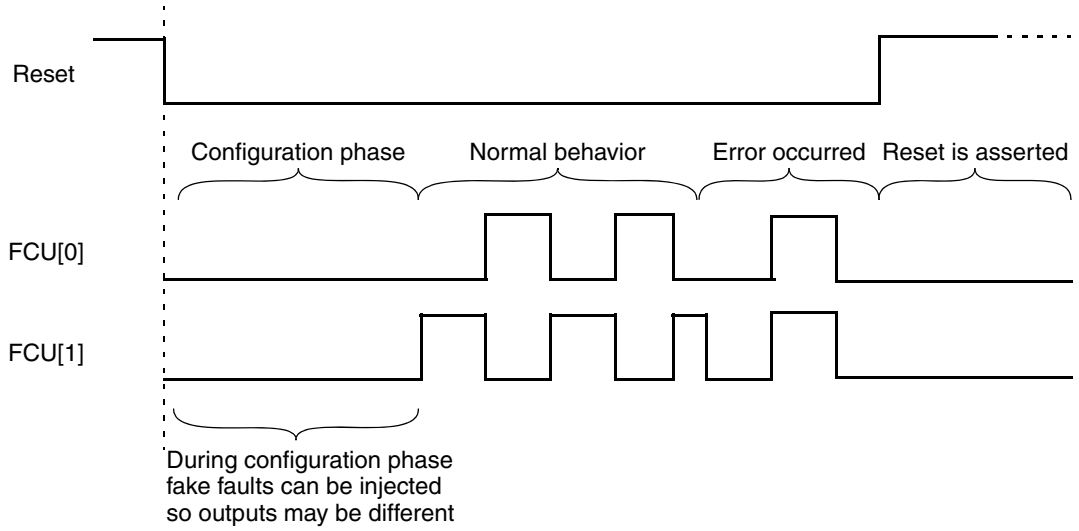


Figure 450. Dual rail coding example

Time switching protocol

FCU[0] is toggled between logic 0 and logic 1 with a defined frequency f = 1 kHz @ 64 MHz (f is approximated, as shown by *Equation 34*) and duty cycle d = 50%.

Equation 34

$$\text{EOUT freq} = \text{SYS\_CLK} \div \{4096 \times [(\text{FOP} + 1) \times 2]\}$$

$$64\ \text{MHz} \div \{4096 \times [(7 + 1) \times 2]\} = 976.5\ \text{Hz}$$

Frequency can be varied by using the same prescaler as used for the dual-rail protocol (FOP field in FCU_MCR). This frequency modulation protocol is violated when the FCU goes into Fault state.

During initialization phase, FCU[0] is set high. When a fault is detected, FCU[0] is set low.

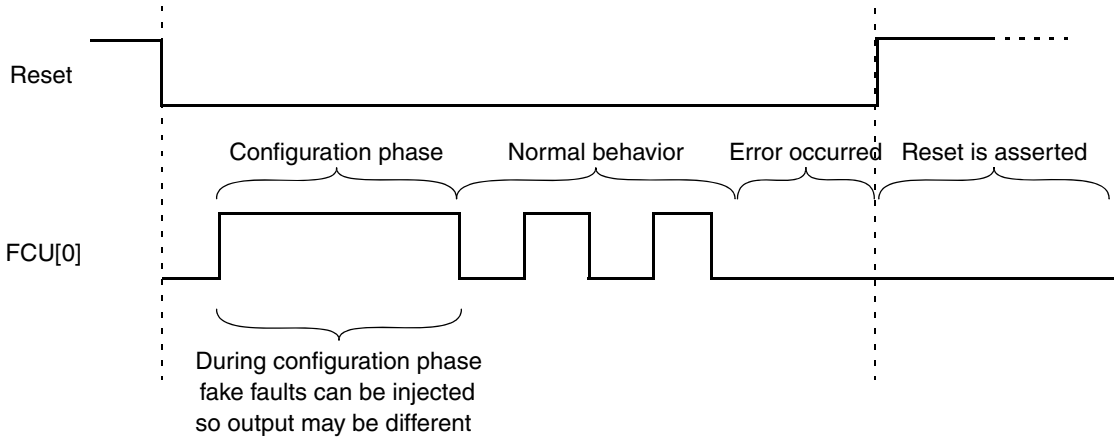


Figure 451. Time switching protocol example

Bi-Stable protocol

In this protocol during the Init and the Fault state, faulty state is indicated. In the Normal/Alarm state, non-faulty state is indicated. *Table 413* shows bi-stable encoding for FCU[0].

Table 413. Bi-stable coding

Logical value	Bi-stable encoding
faulty	0
non-faulty	1

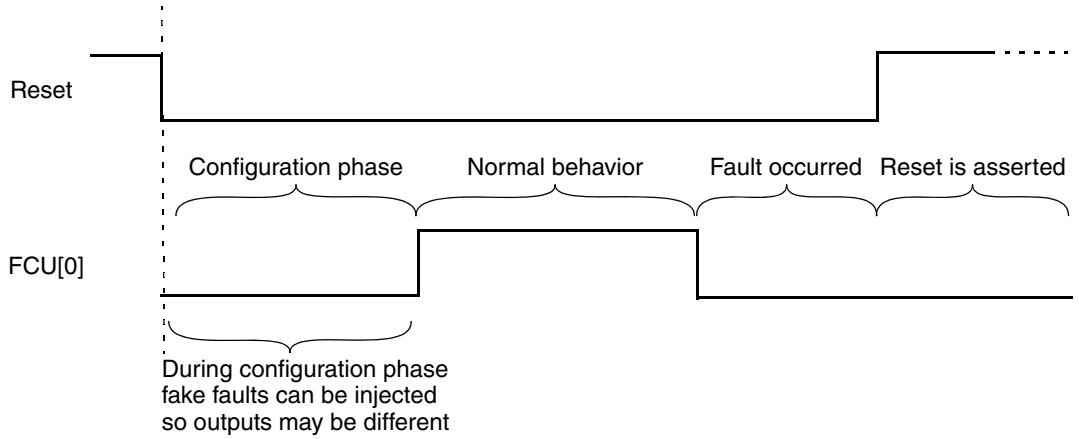


Figure 452. Bi-stable coding example

29 Wakeup Unit (WKPU)

29.1 Overview

The Wakeup Unit (WKPU) supports one external source that causes non-maskable interrupt requests.

29.2 Features

The WKPU provides non-maskable interrupt support with these features:

- 1 NMI source
- 1 analog glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
- Edge detection

29.3 External signal description

The WKPU has one signal input that can be used as non-maskable interrupt.

Note: The user should be aware that the Wake-up pins are enabled in ALL modes, therefore, the Wake-up pins should be correctly terminated to ensure minimal current consumption. Any unused Wake-up signal input should be terminated by using an external pull-up or pull-down, or by internal pull-up enabled at WKUP_WIPUER. Also care has to be taken on packages where the Wake-up signal inputs are not bonded. For these packages the user must ensure the internal pull-up are enabled for those signals not bonded.

29.4 Memory map and registers description

This section provides a detailed description of all registers accessible in the WKPU module.

29.4.1 Memory map

Table 414 lists the WKPU registers.

Table 414. WKPU memory map

Offset from WKPU_BASE (0xC3F9_4000)	Register	Location
0x0000	NSR—NMI Status Flag Register	*on page 29-777*
0x0004–0x0007	Reserved	
0x0008	NCR—NMI Configuration Register	*on page 29-778*
0x000C–0x3FFF	Reserved	

29.4.2 Registers description

This section describes the Wakeup Unit registers.

NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Figure 453. NMI Status Flag Register (NSR)

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

Table 415. NSR field descriptions

Field	Description
0 NIF	NMI Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NCR.NREEor NCR.NFEE is set), NIFcauses an interrupt request. 0: No event has occurred on the pad. 1: An event as defined by NRRC.NREE or NCR.NFEE has occurred.
1 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 (NCR.NREEor NCR.NFEE set), NOVF causes an interrupt request. 0: No overrun has occurred on NMI input. 1: An overrun has occurred on NMI input.

NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Figure 454. NMI Configuration Register (NCR)

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		0	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

Table 416. NCR field descriptions

Field	Description
0 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.
1-2 NDSS	NMI Destination Source Select 00: Non-maskable interrupt 01: Critical interrupt 10: Machine check request 11: Reserved
5 NREE	NMI Rising-edge Events Enable 0: Rising-edge event is disabled 1: Rising-edge event is enabled
6 NFEE	NMI Falling-edge Events Enable 0: Falling-edge event is disabled 1: Falling-edge event is enabled
7 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 NREEand NFEE disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

29.5 Functional description

29.5.1 General

This section provides a complete functional description of the Wakeup Unit.

29.5.2 Non-Maskable Interrupts

The Wakeup Unit supports one non-maskable interrupt, which is allocated to pin 1.

The Wakeup Unit supports the generation of three types of interrupts from the NMI input to the device. The Wakeup Unit supports the capturing of a second event per NMI input before the interrupt is cleared, thus reducing the chance of losing an NMI event.

Each 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.

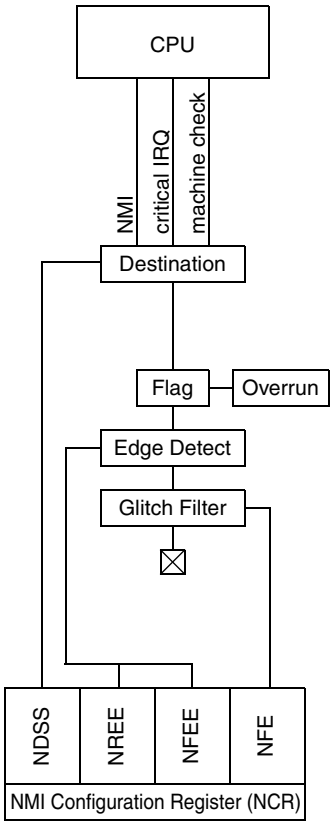


Figure 455. NMI pad diagram

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 454*). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected and should not be configured.

The active NMI edge is controlled by the user 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.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See *Table 416* for details.

An NMI supports a status flag and an overrun flag, which are located in the NSR register (see *Figure 453*). This register is a clear-by-write-1 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, 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 will not be cleared.

30 Periodic Interrupt Timer (PIT)

30.1 Introduction

The Periodic Interrupt Timer (PIT) block implements several timers that can be used for DMA triggering, general purpose interrupts and system wakeup.

Figure 456 shows the PIT block diagram.

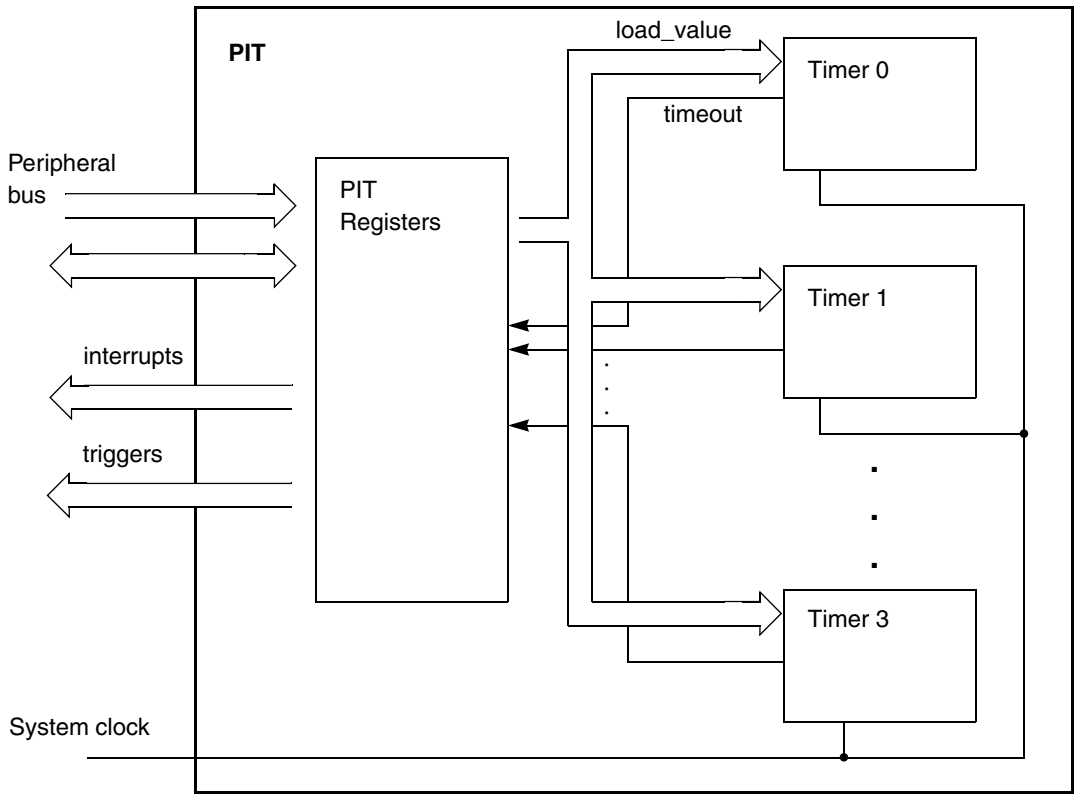


Figure 456. PIT block diagram

30.1.1 Overview

This chapter describes the function of the Periodic Interrupt Timer block (PIT). The PIT is an array of four timers that can be used to raise interrupts and trigger DMA channels.

30.1.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses to initiate DMA transfers with other peripherals (ex: initiate a SPI message transfer sequence)
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

30.2 Signal description

The PIT module has no external pins.

30.3 Memory map and registers description

This section provides a detailed description of all registers accessible in the PIT module.

30.3.1 Memory map

Table 417 gives an overview on all PIT registers.

Table 417. PIT memory map

Offset from PIT_BASE (0xC3FF_0000)	Register	Location
0x0000	*PITMCR—PIT Module Control Register*	*on page 30-783*
0x0004–0x00FF	Reserved	
Timer Channel 0		
0x0100	LDVAL0—Timer 0 Load Value Register	*on page 30-784*
0x0104	CVAL0—Timer 0 Current Value Register	*on page 30-785*
0x0108	TCTRL0—Timer 0 Control Register	*on page 30-786*
0x010C	*TFLG0—Timer 0 Flag Register*	*on page 30-787*
Timer Channel 1		
0x0110	LDVAL1—Timer 1 Load Value Register	*on page 30-784*
0x0114	CVAL1—Timer 1 Current Value Register	*on page 30-785*
0x0118	TCTRL1—Timer 1 Control Register	*on page 30-786*
0x011C	*TFLG1—Timer 1 Flag Register*	*on page 30-787*
Timer Channel 2		
0x0120	LDVAL2—Timer 2 Load Value Register	*on page 30-784*
0x0124	CVAL2—Timer 2 Current Value Register	*on page 30-785*
0x0128	TCTRL2—Timer 2 Control Register	*on page 30-786*
0x012C	*TFLG2—Timer 2 Flag Register*	*on page 30-787*
Timer Channel 3		
0x0130	LDVAL3—Timer 3 Load Value Register	*on page 30-784*
0x0134	CVAL3—Timer 3 Current Value Register	*on page 30-785*
0x0138	TCTRL3—Timer 3 Control Register	*on page 30-786*
0x013C	*TFLG3—Timer 3 Flag Register*	*on page 30-787*
0x0140–0x3FFF	Reserved	

Note: Reserved registers read as 0. Writes have no effect.

30.3.2 Registers description

This section describes in address order all the PIT registers and their individual bits. PIT registers are accessible only when the core is in supervisor mode (see *Section 15.4.3, "ECSM_reg_protection*).

PIT Module Control Register (PITMCR)

This register controls whether the timer clocks are enabled and whether the timers run in debug mode.

Figure 457. PIT Module Control Register (PITMCR)

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	1	0

Table 418. PITMCR field descriptions

Field	Description
MDIS	Module Disable Used to disable 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 (default)
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.

Timer Load Value Register *n* (LDVAL*n*)

These registers select the timeout period for the timer interrupts.

Figure 458. Timer Load Value Register *n* (LDVAL*n*)

Channel Base + 0x0000
LDVAL0 = PIT_BASE + 0x0100
Address: LDVAL1 = PIT_BASE + 0x0110 Access: User read/write
LDVAL2 = PIT_BASE + 0x0120
LDVAL3 = PIT_BASE + 0x0130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TSV 31	TSV 30	TSV 29	TSV 28	TSV 27	TSV 26	TSV 25	TSV 24	TSV 23	TSV 22	TSV 21	TSV 20	TSV 19	TSV 18	TSV 17	TSV 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	TSV 15	TSV 14	TSV 13	TSV 12	TSV 11	TSV 10	TSV 9	TSV 8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 419. LDVAL*n* field descriptions

Field	Description
TSV*n*	Time Start Value Bits These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be 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 463*).

Current Timer Value Register *n* (CVAL*n*)

These registers indicate the current timer position.

Figure 459. Current Timer Value register *n* (CVAL*n*)

Channel Base + 0x0004
CVAL0 = PIT_BASE + 0x0104
Address: CVAL1 = PIT_BASE + 0x0114 Access: User read-only
CVAL2 = PIT_BASE + 0x0124
CVAL3 = PIT_BASE + 0x0134

	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

Table 420. 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 will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see *Figure 457*).

Timer Control Register *n* (TCTRL*n*)

The TCTRL register contains the control bits for each timer.

Figure 460. Timer Control register *n* (TCTRL*n*)

Channel Base + 0x0008
TCTRL0 = PIT_BASE + 0x0108
Address: TCTRL1 = PIT_BASE + 0x0118 Access: User read/write
TCTRL2 = PIT_BASE + 0x0128
TCTRL3 = PIT_BASE + 0x0138

	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

Table 421. TCTRL*n* field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0: Interrupt requests from Timer x are disabled 1: Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0: Timer will be disabled 1: Timer will be active

Timer Flag Register *n* (TFLG*n*)

These registers contain the PIT interrupt flags.

Figure 461. Timer Flag register *n* (TFLG*n*)

Channel Base + 0x000C
TFLG0 = PIT_BASE + 0x010C
Address: TFLG1 = PIT_BASE + 0x011C Access: User read/write
TFLG2 = PIT_BASE + 0x012C
TFLG3 = PIT_BASE + 0x013C

	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

Table 422. 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: Time-out has not yet occurred 1: Time-out has occurred

30.4 Functional description

30.4.1 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 will be available on a separate interrupt line.

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 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse, and set 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 462*).

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 463*).

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 will then be loaded after the next trigger event (see *Figure 464*).

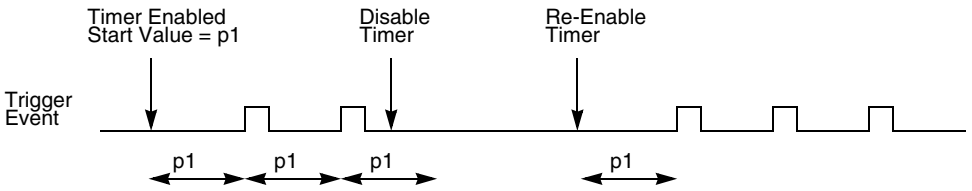


Figure 462. Stopping and starting a timer

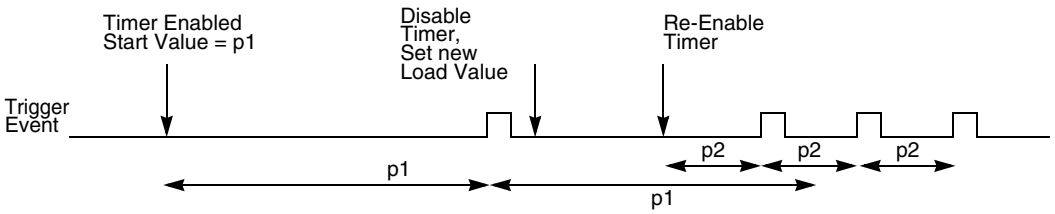


Figure 463. Modifying running timer period

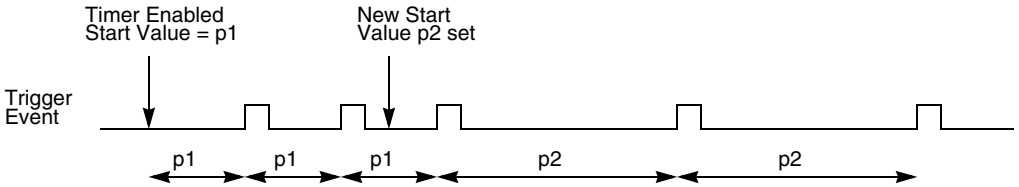


Figure 464. Dynamically setting a new load value

Debug mode

In Debug mode, the timers are frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g., timer values) and then continue the operation.

30.4.2 Interrupts

All of the timers support interrupt generation. Refer to *9, "Interrupt Controller (INTC)* 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 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

30.5 Initialization and application information

30.5.1 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 0 to the MDIS bit in the PITMCR.

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 setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// 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 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

31 System Timer Module (STM)

31.1 Overview

The System Timer Module (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).

31.2 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

31.3 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. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

31.4 External signal description

The STM does not have any external interface signals.

31.5 Memory map and registers 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 generates a bus error termination. STM registers are accessible only when the core is in supervisor mode (see *Section 15.4.3, "ECSM_reg_protection*).

31.5.1 Memory map

The STM memory map is shown in *Table 423*.

Table 423. STM memory map

Offset from STM_BASE 0xFFF3_C000	Register	Location
0x0000	STM_CR—STM Control Register	*on page 31-791*
0x0004	STM_CNT—STM Counter Value	*on page 31-792*

Table 423. STM memory map (continued)

Offset from STM_BASE 0xFFF3_C000	Register	Location
0x0008–0x000F	Reserved	
0x0010	STM_CCR0—STM Channel 0 Control Register	*on page 31-793*
0x0014	STM_CIR0—STM Channel 0 Interrupt Register	*on page 31-793*
0x0018	STM_CMP0—STM Channel 0 Compare Register	*on page 31-794*
0x001C	Reserved	
0x00200	STM_CCR1—STM Channel 1 Control Register	*on page 31-793*
0x00244	STM_CIR1—STM Channel 1 Interrupt Register	*on page 31-793*
0x00288	STM_CMP1—STM Channel 1 Compare Register	*on page 31-794*
0x002C	Reserved	
0x0030	STM_CCR2—STM Channel 2 Control Register	*on page 31-793*
0x0034	STM_CIR2—STM Channel 2 Interrupt Register	*on page 31-793*
0x0038	STM_CMP2—STM Channel 2 Compare Register	*on page 31-794*
0x003C	Reserved	
0x0040	STM_CCR3—STM Channel 3 Control Register	*on page 31-793*
0x0044	STM_CIR3—STM Channel 3 Interrupt Register	*on page 31-793*
0x0048	STM_CMP3—STM Channel 3 Compare Register	*on page 31-794*
0x004C–0x3FFF	Reserved	

31.5.2 Registers description

The following sections detail the individual registers within the STM programming model.

STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Figure 465. STM Control Register (STM_CR)

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[7:0]								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

Table 424. STM_CR field descriptions

Field	Description
CPS[7:0]	Counter Prescaler Selects the clock divide value for the prescaler (1 - 256). 0x00 Divide system clock by 1. 0x01 Divide system clock by 2. ... 0xFF 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.

STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Figure 466. STM Count Register (STM_CNT)

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	CNT															
Reset	0	0	0	0	0	0	0	0	0	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	CNT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 425. 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.

STM Channel Control Register (STM_CCR*n*)

The STM Channel Control Register (STM_CCR*n*) enables and services channel *n* of the timer.

Figure 467. STM Channel Control Register (STM_CCR*n*)

Address: Base + 0x0010 (STM_CCR0), Base + 0x0020 (STM_CCR1), Base + 0x0030 (STM_CCR2) — 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	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 426. STM_CCR*n* field descriptions

Field	Description
CEN	Channel Enable 0 The channel is disabled. 1 The channel is enabled.

STM Channel Interrupt Register (STM_CIR*n*)

The STM Channel Interrupt Register (STM_CIR*n*) enables and services channel *n* of the timer.

Figure 468. STM Channel Interrupt Register (STM_CIR*n*)

Address: Base + 0x0014 (STM_CIR0), Base + 0x0024 (STM_CIR1), Base + 0x0034 (STM_CIR2) — 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	CIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 427. STM_CIR*n* field descriptions

Field	Description
CIF	Channel 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 a match on the channel.

STM Channel Compare Register (STM_CMP*n*)

The STM Channel Compare Register (STM_CMP*n*) holds the compare value for channel *n*.

Figure 469. STM Channel Compare Register (STM_CMP*n*)

Address: Base + 0x0018 (STM_CMP0)
Base + 0x0028 (STM_CMP1)
Base + 0x0038 (STM_CMP2)

Access: User read/write

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

Table 428. 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.

31.6 Functional description

The System Timer Module (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 will set the STM_CIR[CIF] bit and generate 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.

32 Cyclic Redundancy Check (CRC)

32.1 Introduction

The Cyclic Redundancy Check (CRC) computing unit is dedicated to the computation of CRC, thus off-loading the CPU. The SPC560P40/34 CRC supports two contexts. 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 (byte/half-word/word) formats.

32.1.1 Glossary

- CRC: cyclic redundancy check
- CPU: central processing unit
- DMA: direct memory access
- CCITT: ITU-T (for Telecommunication Standardization Sector of the International Telecommunications Union)
- SW: software
- WS: wait state
- SPI: serial peripheral interface

32.2 Main features

- 2 contexts for the concurrent CRC computation
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- 2 hard-wired polynomials (CRC-16-CCITT and CRC-32 ethernet)
- Support for byte/half-word/word width of the input data stream

32.2.1 Standard features

- IPS bus interface
- CRC-16-CCITT
- CRC-32 ethernet

32.3 Block diagram

Figure 470 shows the top level diagram of the CRC unit.

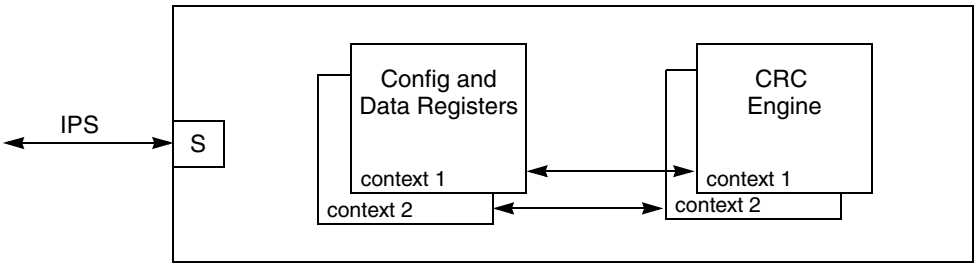


Figure 470. CRC top level diagram

32.3.1 IPS bus interface

The IPS bus interface is a slave bus used for configuration and data streaming (CRC computation) purposes via CPU or DMA. The following bus operations (contiguous byte enables) are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[31:16] or data[15:0]) data write/read operations to any registers
- Byte (8-bit, data[31:24] or data[23:16] or data[15:8] or data[7:0]) data write/read operations to any registers
- Any other operation (free byte enables or other operations) must be avoided.

The CRC generates a transfer error in the following cases:

- Any write/read access to the register addresses not mapped on the peripheral but included in the address space of the peripheral.
- Any write/read operation different from byte/hword/word (free byte enables or other operations) on each register.

The registers of the CRC module are accessible (read/write) in each access mode: user, supervisor, or test.

In terms of bus performance of the operations, following the summary:

- 0 WS (single bus cycle) for each write/read operations to the CRC_CFG and CRC_INP registers
- 0 WS (single bus cycle) for each write operation to the CRC_ CSTAT register
- Double WS (3 bus cycles) for each read operation to the CRC_ CSTAT or CRC_OUTP registers immediately following (next clock cycle) a write operation to the CRC_CSTAT, CRC_INP or CRC_CFG registers belonging to the same context. In all the other cases no WS are inserted.

32.4 Functional description

The CRC module supports the CRC computation for each context. Each context has a 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 or equal to the number of contexts.

Two standard generator polynomials are given in *Equation 35* and *Equation 36* for the CRC computation of each context.

Equation 35CRC-16-CCITT (x25 protocol)

$$X^{16} + X^{12} + X^5 + 1$$

Equation 36CRC-32 (ethernet protocol)

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

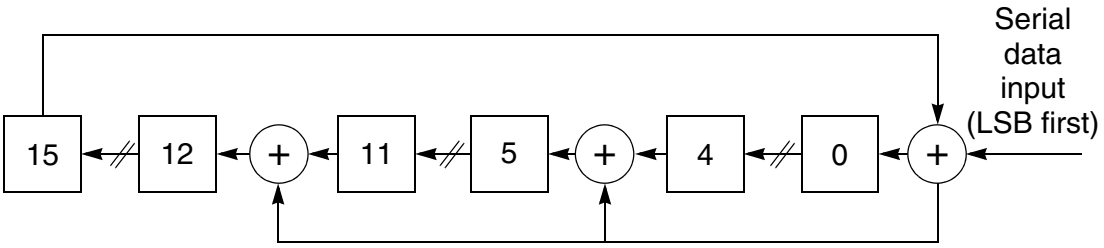


Figure 471. CRC-CCITT engine concept scheme

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 471* 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 (e.g., SPI, ..) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register) before to transmit the CRC.

The usage of the CRC is summarized in the flow-chart given in *Figure 472*.

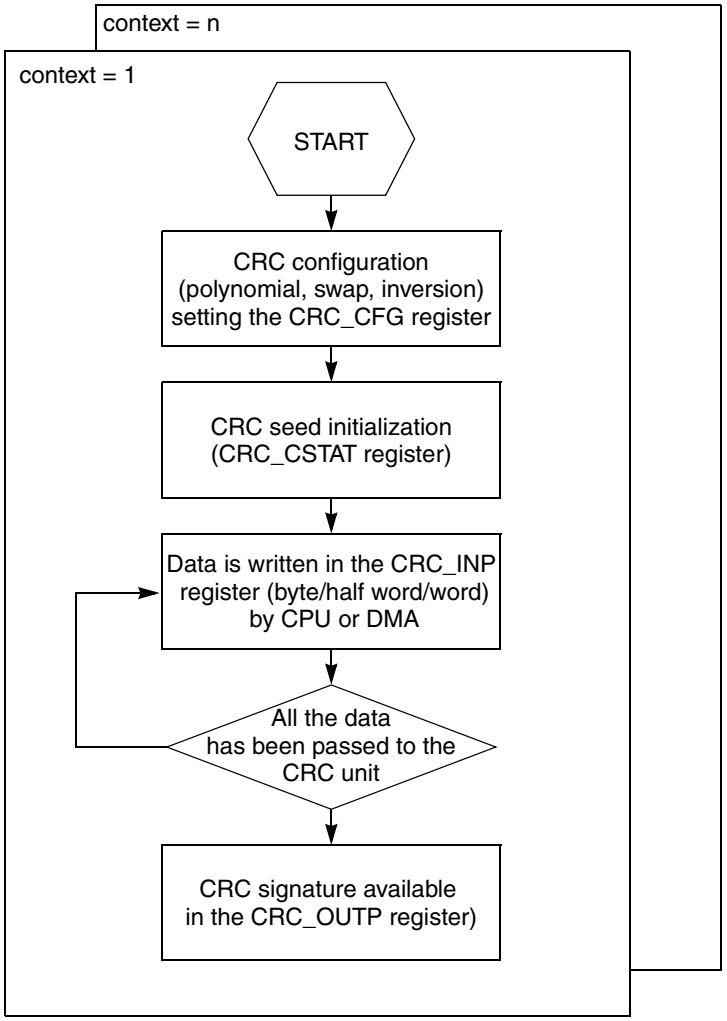


Figure 472. CRC computation flow

32.5 Memory map and registers description

Table 429 shows the CRC memory map.

Table 429. CRC memory map

Offset from CRC_BASE 0xFFE6_8000	Register	Location
0x0000	*CRC_CFG—CRC Configuration Register, Context 1*	*on page 32-800*
0x0004	*CRC_INP—CRC Input Register, Context 1*	*on page 32-801*
0x0008	*CRC_CSTAT—CRC Current Status Register, Context 1*	*on page 32-802*
0x000C	CRC_OUTP—CRC Output Register, *Context 1*	*on page 32-802*

Table 429. CRC memory map (continued)

Offset from CRC_BASE 0xFFE6_8000	Register	Location
0x0010	*CRC_CFG—CRC Configuration Register, Context 2*	*on page 32-800*
0x0014	*CRC_INP—CRC Input Register, Context 2*	*on page 32-801*
0x0018	*CRC_CSTAT—CRC Current Status Register, Context 2*	*on page 32-802*
0x001C	CRC_OUTP—CRC Output Register, *Context 2*	*on page 32-802*
0x0020–0x3FFF	Reserved	

32.5.1 CRC Configuration Register (CRC_CFG)

Figure 473. CRC Configuration Register (CRC_CFG)

Address: Context 1: Base + 0x0000
Context 2: 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	0	POLYG	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 430. CRC_CFG field descriptions

Field	Description
0:28	Reserved These are reserved bits. These bits are always read as 0 and must always be written with 0.
29	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.

Table 430. CRC_CFG field descriptions (continued)

Field	Description
30	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.
31	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.

32.5.2 CRC Input Register (CRC_INP)

Figure 474. CRC Input Register (CRC_INP)

Address: Context 1: Base + 0x0004
Context 2: Base + 0x0014

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

Table 431. CRC_INP field descriptions

Field	Description
0:31	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.

32.5.3 CRC Current Status Register (CRC_CSTAT)

Figure 475. CRC Current Status Register (CRC_CSTAT)

Address: Context 1: Base + 0x0008
Context 2: Base + 0x0018
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

Table 432. CRC_CSTAT field descriptions

Field	Description
0:31	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 significant. 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.

32.5.4 CRC Output Register (CRC_OUTP)

Figure 476. CRC Output Register (CRC_OUTP)

Address: Context 1: Base + 0x000C
Context 2: Base + 0x001C
Access: User read/write

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

Table 433. CRC_OUTP field descriptions

Field	Description
0:31	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 significant. The 16 MSB bits are tied at 0b during the computation. This register can be read by the software.

32.6 Use cases and limitations

Two main use cases shall be considered:

- Calculation of the CRC of the configuration registers during the process safety time
- 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 the following type of data transfer: mem2mem, periph2mem, mem2periph, the following sequence, as given in *Figure 477*, shall be 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 (phase1) 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 (e.g., SPI Tx fifo) (phase 3) by DMA (mem2periph data transfer, channel x)

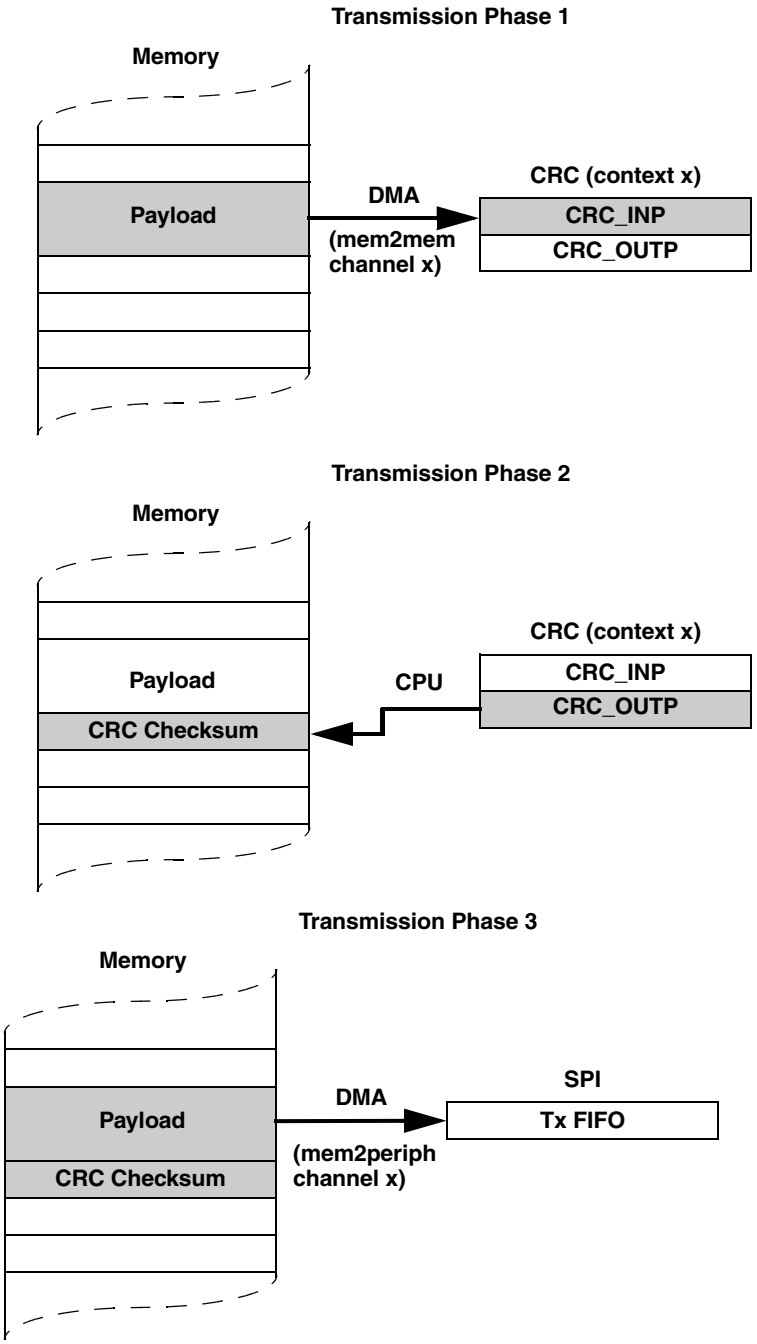


Figure 477. DMA-CRC Transmission Sequence

The following sequence, as given in *Figure 478*, shall be 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 (e.g., 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)1

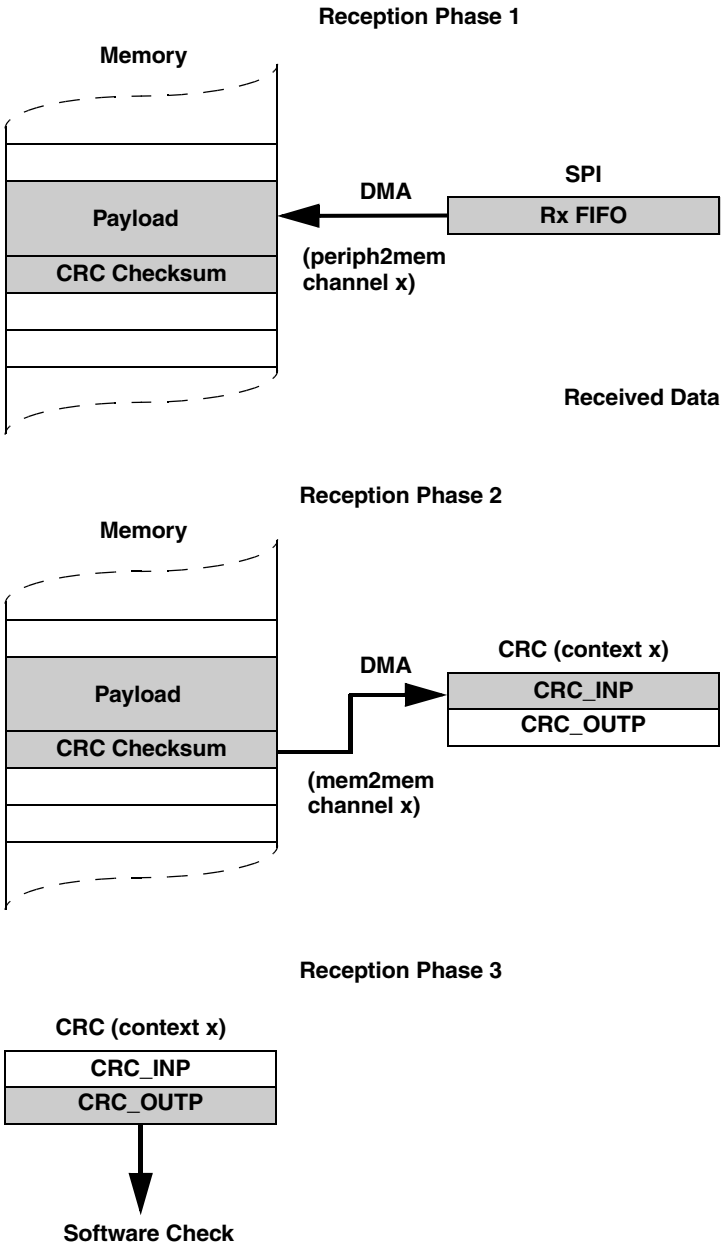


Figure 478. DMA-CRC Reception Sequence

33 Boot Assist Module (BAM)

33.1 Overview

The Boot Assist Module is a block of read-only memory containing VLE code that is executed according to the boot mode of the device.

The BAM allows downloading boot code via the FlexCAN or LINFlex interfaces into internal SRAM and then executing it.

33.2 Features

The BAM provides the following features:

- SPC560P40/34 in static 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 internal flash module

33.3 Boot modes

The SPC560P40/34 device 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 LINFlex or FlexCAN interface and then execute it.

If booting is not possible with the selected configuration (e.g., if no Boot ID is found in the selected boot location) then the device enters the static mode.

33.4 Memory map

The BAM code resides in a reserved 8 KB ROM mapped from address 0xFFFF_C000. The address space and memory used by the BAM application is shown in *Table 434*.

Table 434. BAM memory organization

Parameter	Address
BAM entry point	0xFFFF_C000
Downloaded code base address	0x4000_0100

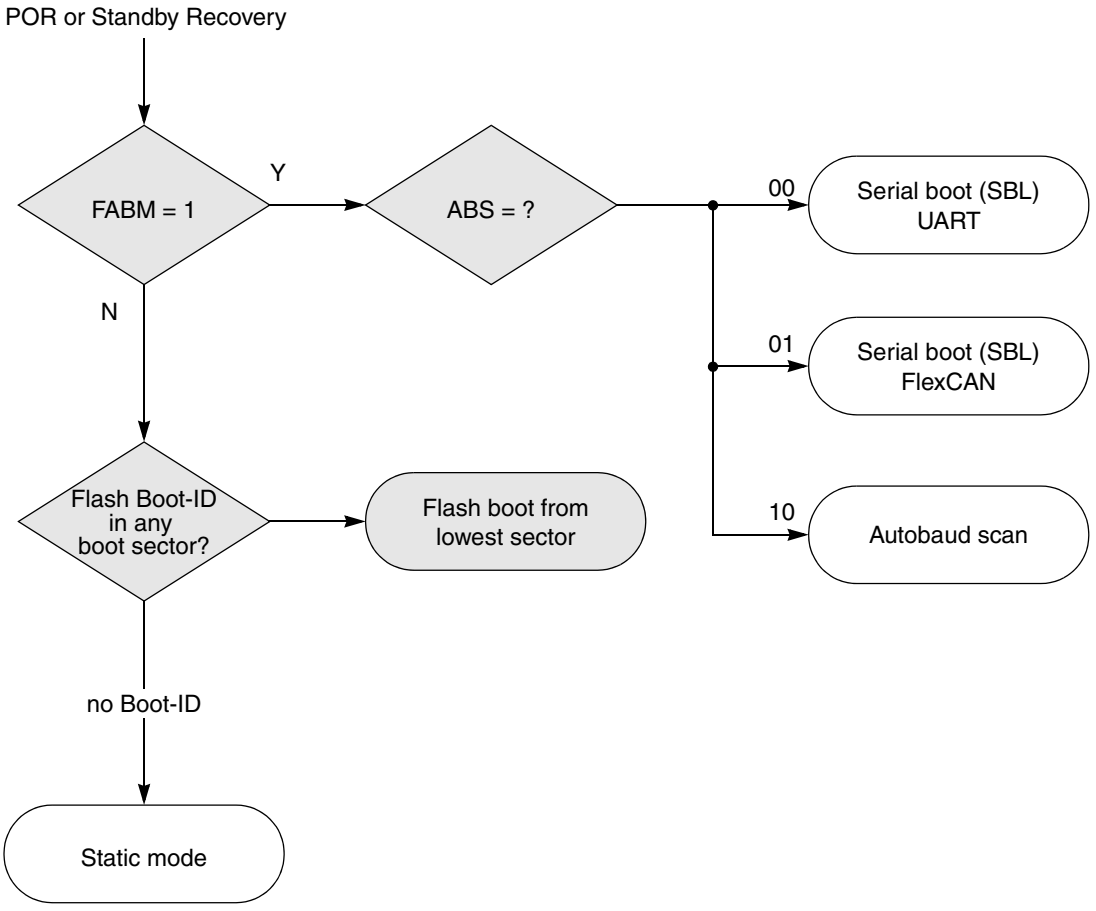
The RAM location where to download the code can be any 4-byte-aligned location starting from the address 0x4000_0100.

33.5 Functional description

33.5.1 Entering boot modes

The SPC560P40/34 detects the boot mode based on external pins and device status. The following sequence applies (see *Figure 479*):

- To boot either from FlexCAN or LINFlex, the device must be forced into an Alternate Boot Loader Mode via the FAB (Force Alternate Boot Mode), 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 435*).
- If FAB is not asserted, the device boots from the lowest Flash sector that contains a valid boot signature.
- If no Flash sector contains a valid boot signature, the device will go into static mode.



Note: The gray blocks in this figure represent hardware actions; white blocks represent software (BAM) actions.

Figure 479. Boot mode selection

Boot configuration pins are:

- PAD A[2] - ABS[0],
- PAD A[3] - ABS[1],
- PAD A[4] - FAB

Table 435. Hardware configuration to select boot mode

FAB	ABS[1:0](1)	Standby-RAM Boot Flag	Boot ID	Boot Mode
1	00	0	—	LINFlex without autobaud
1	01	0	—	FlexCAN without autobaud
1	10	0	—	Autobaud scan

1. During reset the boot configuration pins are weak pull down.

Note: PAD A[2] - ABS[0] is not bonded on SPC560P40/34 64-pin LQFP package so for this package the option 'FlexCAN without Autobaud ' is not available and the internal pull-down on PAD A[2] assures that it is at low logical value at reset.

33.5.2 SPC560P40/34 boot pins

The TX/RX pin (LINFlex_0 and FlexCAN_0) used for serial boot and configuration boot pins to select the serial boot mode are described in the *Table 436* for 64-pin and 100-pin LQFP packages.

Table 436. SPC560P40/34 boot pins

Port pin	Function	Pin	
		64-pin	100-pin
A[2](1)	ABS[0]	—	57
A[3]1	ABS[1]	41	64
A[4]1	FAB	48	75
B[0]	CAN_0 TX	49	76
B[1]	CAN_0 RX	50	77
B[2]	LIN_0 TX	51	79
B[3](2)	LIN_0 RX	—	80(2)
B[7](3)	CAN_0 RX	20(3)	29

1. Weak pull down during reset.
2. SPC560P40/34 100-pin LQFP package uses only PAD B[3] - pin 80 for boot via LINFLEX.
3. SPC560P40/34 64-pin LQFP package uses only PAD B[7] - pin 20 for boot via LINFLEX.

33.5.3 Reset Configuration Half Word (RCHW)

The SPC560P40/34 Flash is partitioned into boot sectors as shown in *Table 438*.

Each boot sector contains the Reset Configuration Half-Word (RCHW) at offset 0x00.

Figure 480. Reset Configuration Half Word (RCHW)

Address: Base + 0x0000 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	VLE	BOOT_ID[0:7]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 437. RCHW field descriptions

Field	Description
0-6	Reserved
7 VLE	VLE Indicator This bit configures the MMU for the boot block to execute as either Power Architecture techmology code or as VLE code. 0 Boot code executes as Power Architecture techmology code 1 Boot code executes as VLE code
8-15 BOOT_ID[0:7]	Is valid if its value is 0x5A, then the sector is considered bootable.

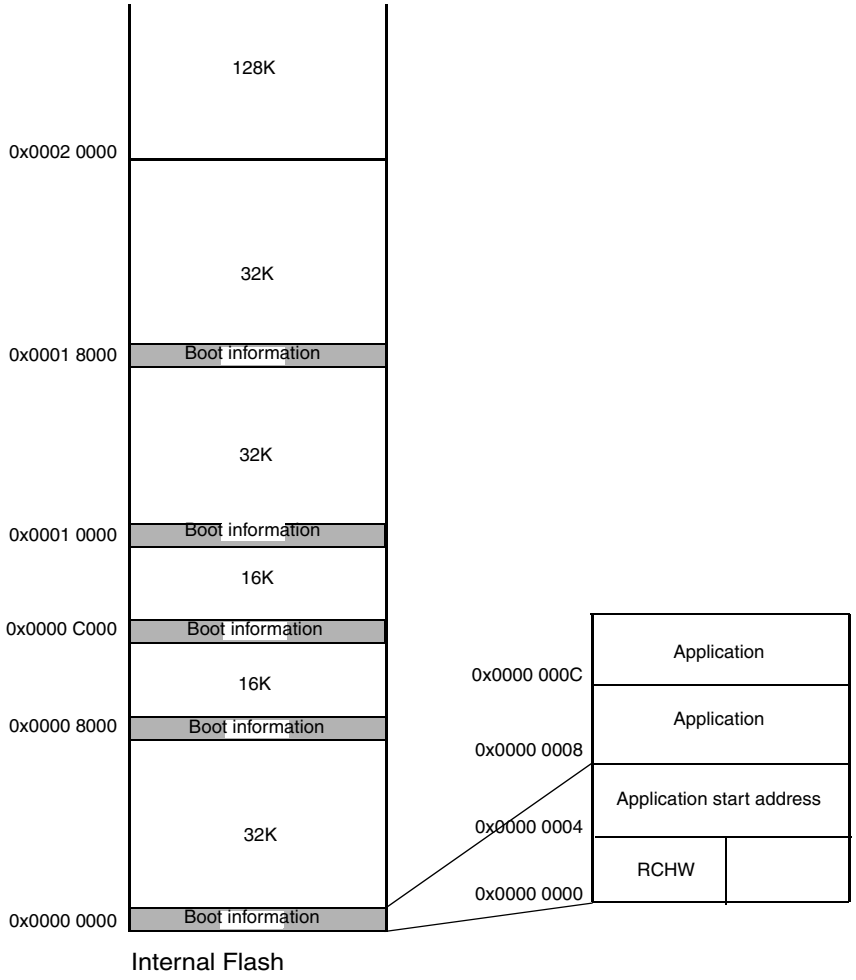


Figure 481. SPC560P40/34 Flash partitioning and RCHW search

Table 438. Flash boot sector

Block	Address
0	0x0000_0000
1	0x0000_8000
2	0x0000_C000
3	0x0001_0000
4	0x0001_8000

33.5.4 Single chip boot mode

In single chip boot mode, the hardware searches the flash boot sector 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. A user application should 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 BAM moves the SPC560P40/34 into static mode.

Boot and alternate boot

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

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

This ensures that even if one boot sector is erased still there will always be another active boot sector:

- Sector shall be activated (i.e., program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector shall be deactivated writing to 0 some of the bits BOOT_ID bit field (bit1 and/or bit3, and/or bit4, and/or bit6).

33.5.5 Boot through BAM

Executing BAM

Single chip boot mode is managed by hardware and BAM does not participate in it.

BAM is executed only on these two following cases:

- Serial boot mode has been selected by FAB pin
- Hardware has not found a valid Boot-ID in any Flash boot locations

If one of these conditions is true, the device fetches code at location 0xFFFF_C000 and BAM application starts.

BAM software flow

Figure 482 illustrates the BAM logic flow.

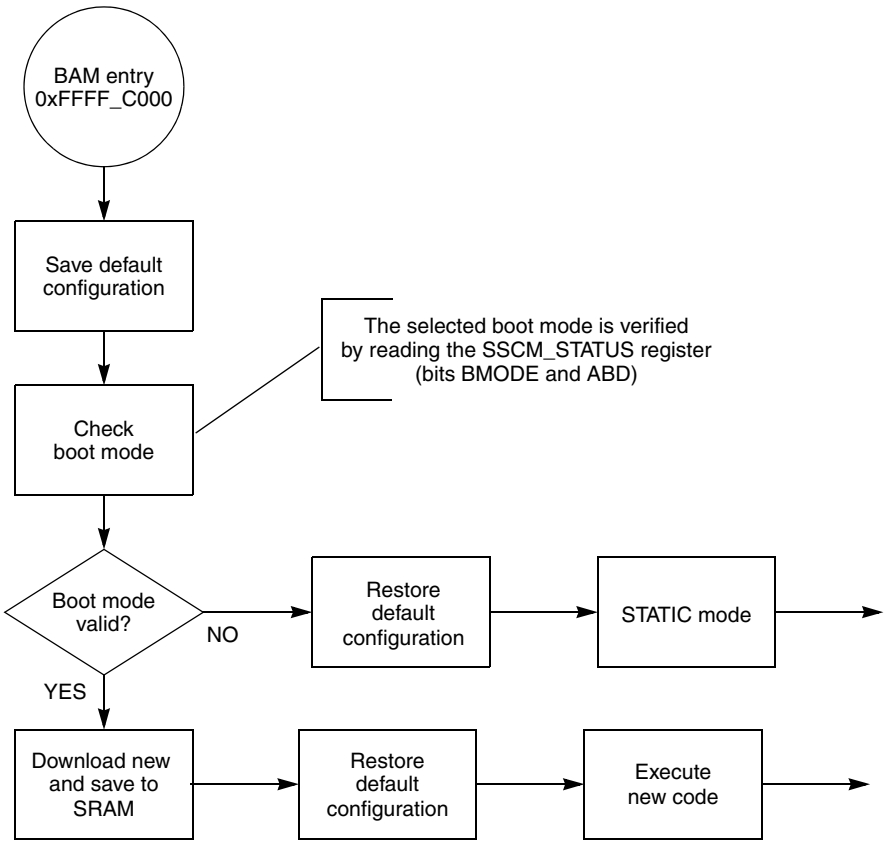


Figure 482. BAM logic flow

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 was just coming out of reset.

The BMODE and ABD fields of SSCM_STATUS register (see *Section , "System Status register (STATUS)*) indicate which boot has to be executed (see *Table 439*).

If the BMODE field shows either a single chip value (011) or a reserved value, the boot mode is not considered valid and the BAM pushes the device into static mode.

In all other cases the code of the relative boot is called. Data is downloaded and saved into proper SRAM location.

Table 439. Fields of SSCM STATUS register used by BAM

Field	Description
BMODE [2:0]	Device Boot Mode 000 Test Flash/autobaud_scan 001 CAN Serial Boot Loader 010 SCI Serial Boot Loader 011 Single Chip 100–111Reserved This field is updated only during reset.

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 an error occurs, (e.g., communication error, wrong boot selected, etc.), the BAM restores the default configuration and puts the device into static mode. Static mode means the device enters the low power mode SAFE and the processor executes a wait instruction. This 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 ME Module is “DRUN”.

BAM resources

BAM uses/initializes the following MCU resources:

- ME and CGM modules to initialize mode and clock sources
- CAN_0, LINFlex_0, and their pads when performing serial boot mode
- SSCM to check the boot mode and during password check (see *Table 439* and *Figure 483*)
- External oscillator

The following hardware resources are used only when autobaud feature is selected:

- STM to measure the baud rate
- CMU to measure the external clock frequency related to the internal RC clock source
- FMPLL to work with system clock near the maximum allowed frequency (this to have higher resolution during baud rate measurement).

As already mentioned, 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 440*).

Table 440. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
f_{extal}	f_{extal} / 833	f_{extal} / 40
8	9600	200 K
12	14400	300 K

Table 440. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
16	19200	400 K
20	24000	500 K
40	48000	1 M

Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send message and receive acknowledge message for autobaud or autobit rate selection. (optional step).
2. Send 64-bit password.
3. Send start address, size of downloaded code in bytes, and VLE bit[d].
4. Download data.
5. Execute code from start address.

Each step must be complete before the next step starts.

The step from 2 to 5 are correct if autobaud is disabled. Otherwise, to measure the baud rate, some data is sent from the host to the MCU before step 2 (see *Section 33.6.1, "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 start waiting.
2. MCU echoes to host the data received.
3. MCU verifies if echo is correct.
   - If data is correct, the host can continue to send data.
   - If data is not correct, the host stops transmitting and the MCU needs to be reset.

All multi-byte data structures are sent MSB first.

A more detailed description of these steps follows.

Download 64-bit password and password check

The first 64 received bits represent the password. This password is sent to the Password Check procedure for verification.

Password check data flow is shown in *Figure 483* where:

- SSCM_STATUS[SEC] = 1 means flash secured
- SSCM_STATUS[PUB] = 1 means flash with public access.

In case of flash with public access, the received password is compared with the public password 0xFEED_FACE_CAFE_BEEF.

d. Since the device supports only VLE code and does not support Book E code, this flag is used only for backward compatibility.

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 uncensored devices, it is possible to download code via LINFlex or FlexCAN (serial boot mode) into internal SRAM with any 64-bit private password stored in the flash and provided during the boot sequence.

In the previous cases, comparison is done by the BAM application. If it goes wrong, BAM pushes the device into static mode.

In case of public access not allowed and flash secured, the password is written into SSCM[PWCMPH/L] registers.

In case of flash secured with public access not allowed (user password configured in the NVPWD0 and NVPWD1 registers), the user must set the swapped password: NVPWD1 and NVPWD0 to access the device (refer to following examples).

Example 15

In devices with flash secured, registers are programmed:

NVPWD0 = 0x87654321

NVPWD1 = 0x12345678

NVSCI0 = 0x55AA1111

NVSCI1 = 0x55AA1111

To download the code via SLB, the provided password is 0x1234_5678_8765_4321 (swapped WITH respect to NVPWD0/NVPWD1 —> NVPWD1/NVPWD0).

Example 16

In devices with flash NOT secured, registers are programmed:

NVPWD0 = 0x87654321

NVPWD1 = 0x12345678

NVSCI0 = 0x55AA55AA

NVSCI1 = 0x55AA55AA

To download the code via SLB the provided password is 0x8765_4321_1234_5678 (as expected from NVPWD0/NVPWD1).

After a fixed time waiting, comparison is done by hardware. Then BAM again verifies the SEC flag in SSCM_STATUS:

- SEC = 0, flash is now unsecured and BAM continues its task
- SEC = 1, flash is still secured because password was wrong; BAM puts the device into static mode.

This fixed time depends on external crystal oscillator frequency (XOSC). With XOSC of 12 MHz, fixed time is 350 ms.

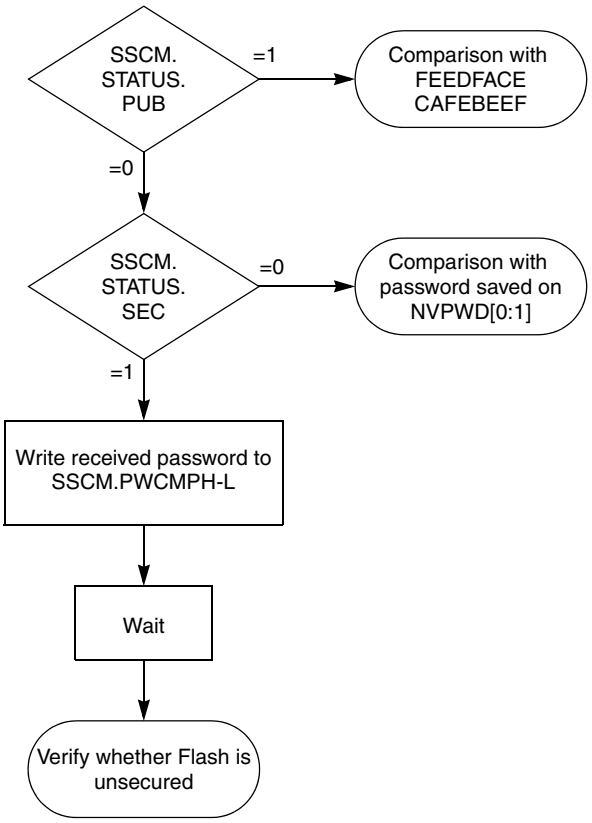


Figure 483. Password check flow

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 484*.

The VLE bit (Variable Length Instruction) indicates which instruction set for which the code has been compiled. This device family supports only VLE = 1. The bit is used for backward compatibility.

The Start Address defines where the received data will be stored and where the MCU will branch 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 have to be loaded.

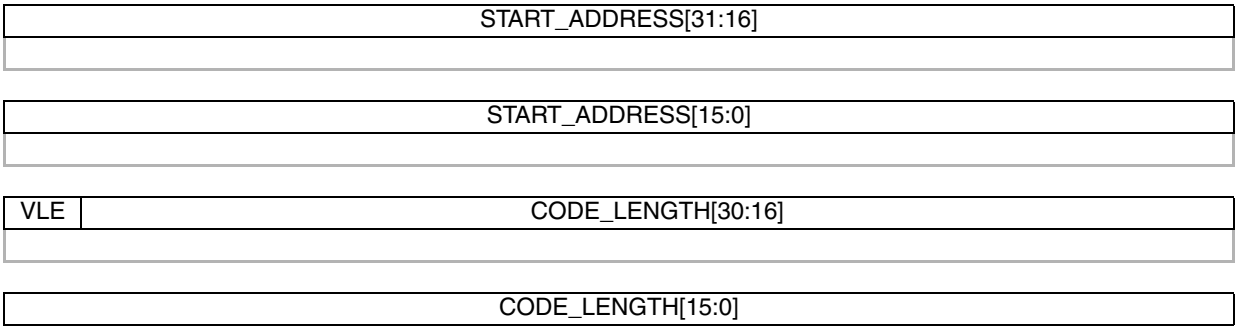


Figure 484. Start address, VLE bit and download size in bytes

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.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), BAM always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, BAM fills it with 0 bytes.

Then a "dummy" word (0x0000_0000) is written to avoid ECC error during core prefetch.

Execute code

The BAM program waits for the last echo message transmission being completed.

Then it restores the initial MCU configuration and jumps to the loaded code at Start Address that was received in step 3 of the protocol.

At this point BAM has finished its tasks and MCU is controlled by new code executing from SRAM.

33.5.6 Boot from UART—autobaud disabled

Configuration

Boot from UART protocol is implemented by LINFlex_0 module. Pins used are:

- LINFlex_TX corresponds to pin B[2]
- LINFlex_RX corresponds to pin B[3].

When autobaud feature is disabled, the system clock is driven by external oscillator.

LINFlex controller is configured to operate at a baud rate = system clock frequency/833 (see *Table 440* for baud rate example), using 8-bit data frame without parity bit and 1 stop bit.

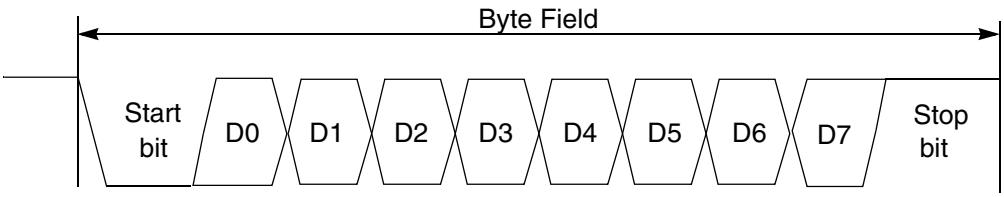


Figure 485. LINFlex bit timing in UART mode

UART boot mode download protocol

Table 441 summarizes the download protocol and BAM action during the UART boot mode.

Table 441. 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 is stored for future use. Verify if VLE bit is set to 1.
4	8 bits of raw binary data	8 bits of raw binary data	8-bit data are packed into 32-bit word. This word is 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.
5	none	none	Branch to downloaded code.

33.5.7 Bootstrap with FlexCAN—autobaud disabled

Configuration

Boot from FlexCAN protocol is implemented by the FlexCAN_0 module. Pins used are:

- CAN_TX corresponds to pin B[0]
- CAN_RX corresponds to pin B[1].

Boot from FlexCAN with autobaud disabled uses system clock driven by the external oscillator.

The FlexCAN controller is configured to operate at a baud rate equal to the system clock frequency/40 (see *Table 440* 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 quanta, and the sample point is 2 time quanta before the end, as shown in *Figure 486*.

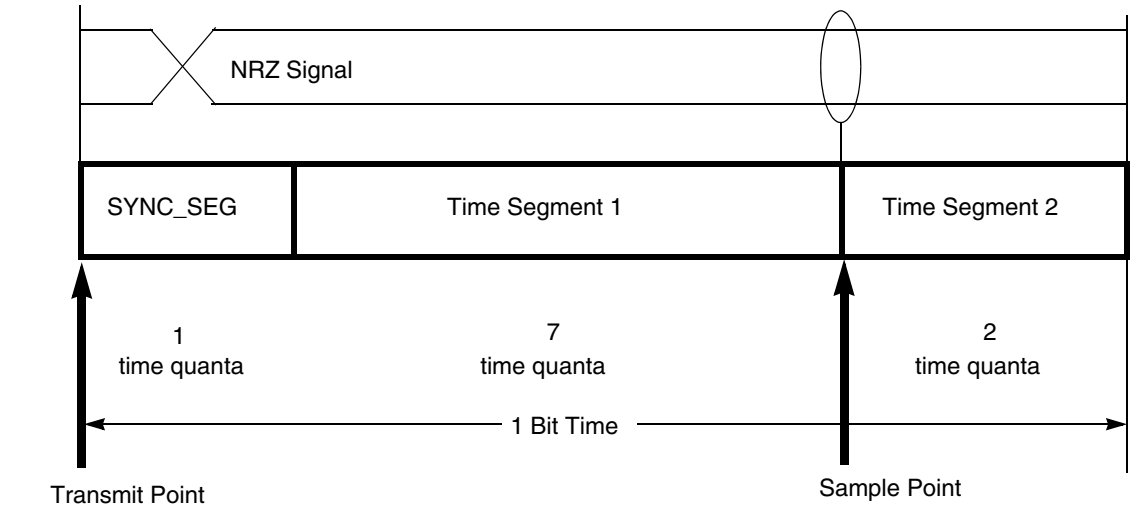


Figure 486. FlexCAN bit timing

33.6 FlexCAN boot mode download protocol

Table 442 summarizes the download protocol and BAM action during the FlexCAN boot mode. All data are transmitted bytewise.

Table 442. 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 is 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-bit 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 downloaded code.

33.6.1 Autobaud feature

The autobaud feature allows boot operation with a wide range of baud rates independent of the external oscillator frequency.

Configuration

SPC560P40/34 devices implement the autobaud feature via FlexCAN or LINFlex selecting the active serial communication peripheral by means of an autoscan routine.

When autobaud configuration is selected by ABS and FAB pins, the autoscan routine starts and listens to the active bus protocol. Initially the LinFlex_0 RX pin and FlexCAN_0 RX pin are configured as GPIO inputs:

- for 100-pin LQFP package internal weak pull-up enabled for both RX pins
- for 64-pin LQFP package internal weak pull-up enabled only for FlexCAN_0 RX pin

The autoscan routine waits in polling for the first LOW level to select which routine will be executed:

- FlexCAN Autobaud routine
- LinFlex Autobaud routine

Then the measurement baud rate is computed to configure the serial communication at the right rate. In the end of baud rate measurement, LinFlex_0 RX pin and FlexCAN_0 RX pin switches to work as dedicated pin.

Baud rate measurement is using the System Timer Module (STM) which is driven by the system clock. Measurement itself is performed by software polling the related inputs as general purpose IO's, resulting in a detection granularity that is directly related to the execution speed of the software.

One main difference of the autobaud feature is that the system clock is not driven directly by the external oscillator, but it is driven by the FMPLL output. The reason is that to have an optimum resolution for baud rate measurement, the system clock needs to be nearer to the maximum allowed device's frequency.

This is achieved with the following two steps:

1. using the Clock Monitor Unit (CMU) and the internal RC oscillator (IRC), the external frequency is measured using the IRC as reference to determine this frequency.
2. Based on the result of this measurement, the FMPLL is programmed to generate a system clock that is configured to be near, but lower, to the maximum allowed frequency.

The relation between system clock frequency and external clock frequency with FMPLL configuration value is shown in *Table 443*.

Table 443. System clock frequency related to external clock frequency

f_{osc} [MHz]	f_{rc}/f_{osc} (1)	f_{sys} [MHz]
4–8	4–2	16–32
8–12	2–4/3	32–48
12–16	4/3–1	36–48
16–24	1–2/3	32–48
> 24	< 2/3	> 24

1. These values and consequently the f_{sys} suffer from the precision of RC internal oscillator used to measure f_{osc} through CMU module.

After setting up the system clock, the BAM autoscan code configures the FlexCAN RX pin (B[1] on all packages) and LINFlex RX pin (B[3] on LQFP100 or B[7] on LQFP64) as GPIO inputs and searches for FlexCAN RX pin level to verify if CAN is connected or not.

Then continuously waits in polling on change of RX pins level.The FlexCAN RX pin level takes precedence. First signal found at low level selects the serial boot routine that will be executed.

In case a low level is detected on any input, the corresponding autobaud measurement functionality is started:

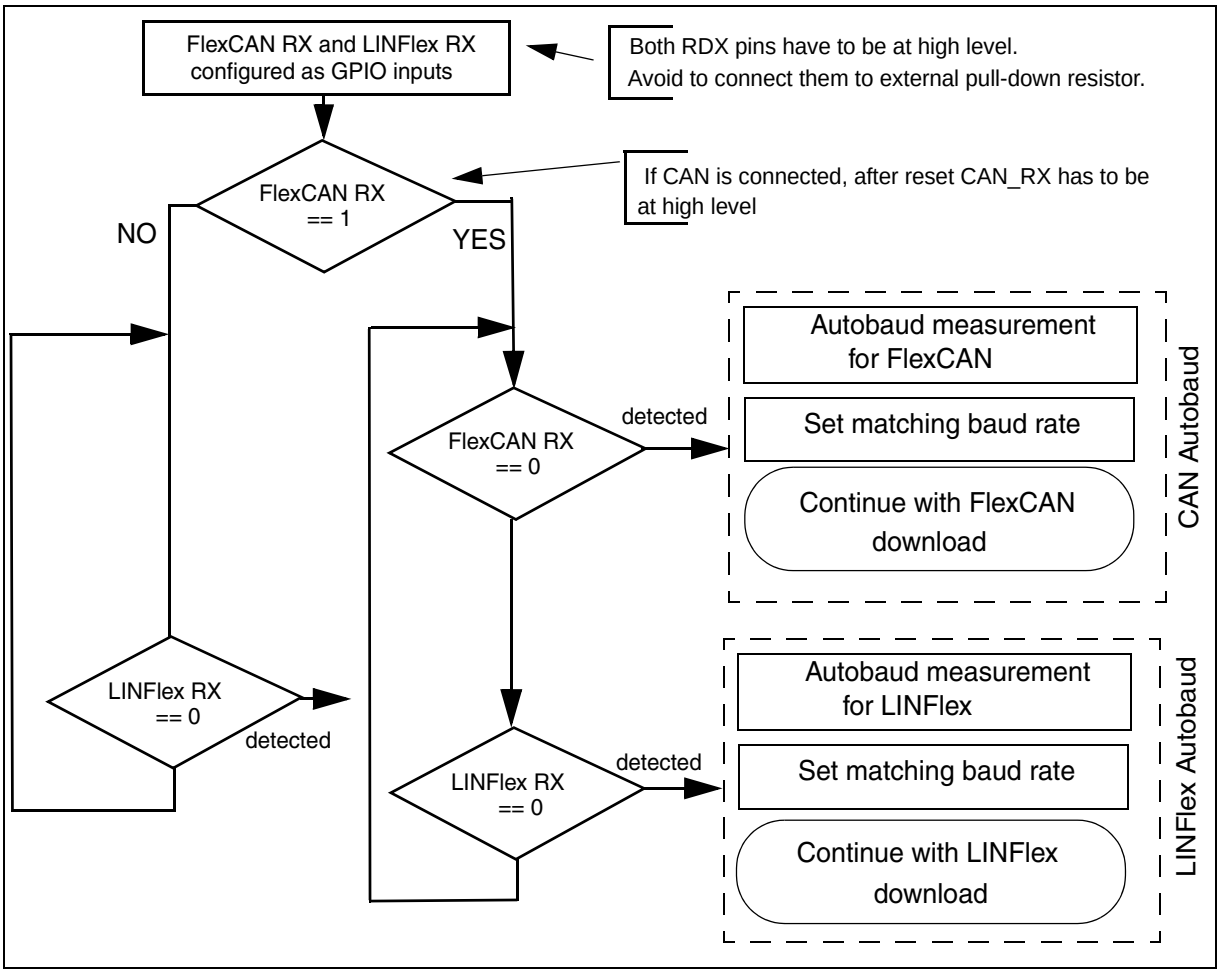
- when FlexCAN RX (corresponds to pin B[1]) level is low, the CAN autobaud measurement starts and then sets up the FlexCAN baud rate accordingly;
- when UART RX (corresponds to pin B[3] on LQFP100 or B[7] on LQFP64) level is low, the UART autobaud measurement starts and then sets up the LINFlex baud rate accordingly.

After performing the autobaud measurement and setting up the baud rate, the corresponding RX input is reconfigured and the related standard download process is started; in case of a detected CAN transmission a download using the CAN protocol as described in section *Section 33.5.7, "Bootstrap with FlexCAN—autobaud disabled*, and in case of a detected UART transmission a download using the UART protocol as described in *Section 33.5.6, "Boot from UART—autobaud disabled*.

The following *Figure 487* identifies the corresponding flow and steps.

Note: When autobaud scan is selected, initially both LINFlex_0 RX pin and FlexCAN_0 RX pin should be at high level. No external circuity should pull-down them to allow right autoscan.

Figure 487. BAM Autoscan code flow



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.

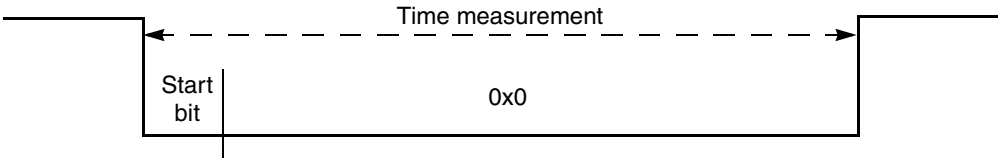


Figure 488. Baud measurement on UART boot

Initially the UART RX pin is configured as GPIO input and it waits in polling for the first falling edge, then STM starts. UART RX pin waits again for the first rising. Then STM stops and from its measurement baud rate is computed.

The LINFlex 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 489*).

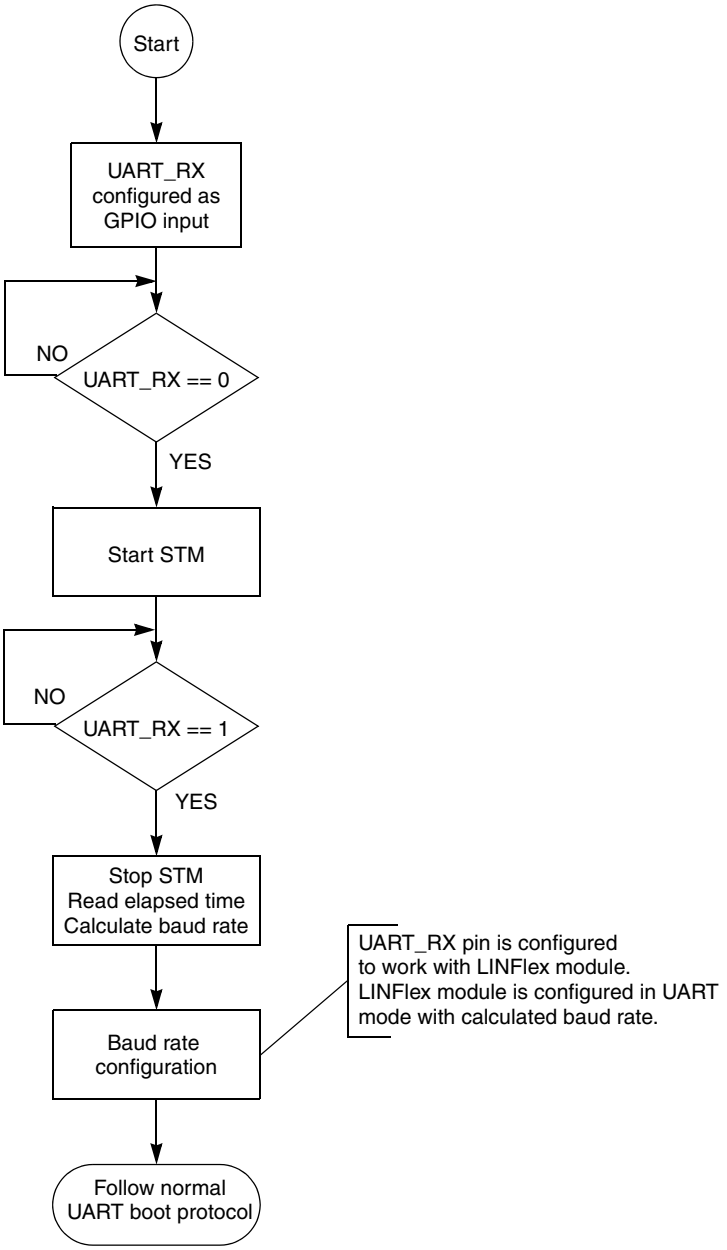


Figure 489. BAM rate measurement flow during UART boot

Choosing the host baud rate

The calculation of the UART baud rate from the length of the first 0 byte that is received, allows the operation of the boot loader with a wide range of baud rates. However, to ensure proper data transfer, the upper and lower limits have to be kept.

SCI autobaud rate feature operates by polling the LINFlex_RX pin for a low signal. The length of time until the next low to high transition is measured using the System Timer Module (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.

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:

Equation 37

$$LDIV = \frac{F_{cpu}}{16 \cdot baudrate}$$

LDIV is an unsigned fixed point number and its mantissa is coded into 13 bits of the LINFlex's register LINIBRR.

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:

Equation 38

$$LINIBRR = \frac{timebase}{144}$$

To minimize errors in baud rate, a large external oscillator frequency value and low baud rate signal are preferred.

Example 17 Baud rate calculation for 24 kBaud signal

Considering a 24 kbaud signal and the device operating with 20 MHz external frequency.

Over 9 bits the STM will measure: (9 × 20 MHz)/24 kbaud = 7497 cycles.

Error expected to be approximately ± 6 cycles due to polling.

Thus, LINIBB will be 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 444*.

Table 444. Maximum and minimum recommended baud rates

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 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 444. Maximum and minimum recommended baud rates (continued)

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 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 the user will be required to ensure they fall within an acceptable error range. This is illustrated in *Figure 490*, which shows the effect of quantization error on the baud rate selection.

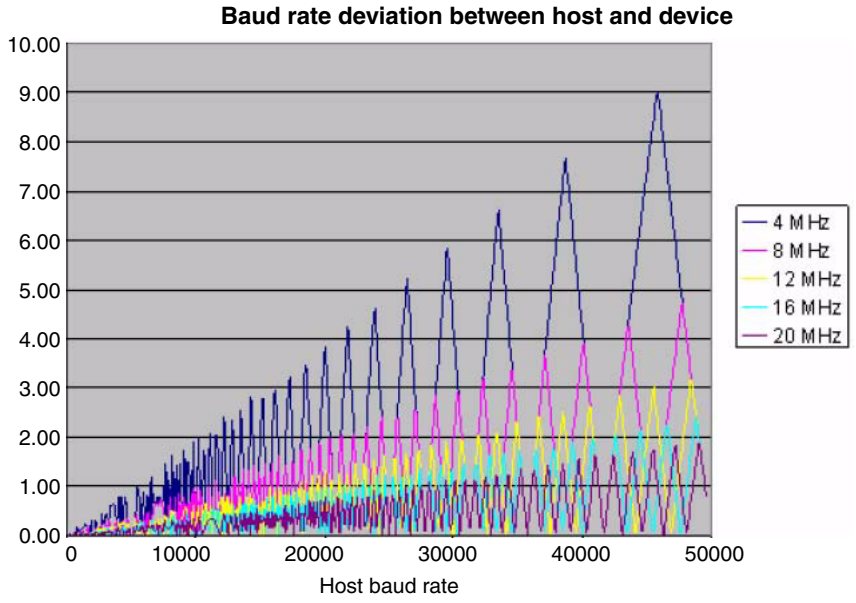


Figure 490. Baud rate deviation between host and SPC560P40/34

Example 18 Baud rate calculation for 250 kBaud signal

Considering reception of a 250kBaud signal from the host and SPC560P40/34 operating with a 4 MHz oscillator. Over 9 bits the time base will measure: (9 × 4e6)/250e3 = 144 cycles.

Thus, LINIBB is set to 144/144 = 1. This results in a baud rate of exactly 250 kBd.

However, a slower 225 kBd signal operating with 4 MHz XTAL would again result in LINIBB = 1, but this time with an 11.1% deviation.

Boot from FlexCAN with autobaud enabled

The only difference between booting from FlexCAN with autobaud enabled and booting from FlexCAN with autobaud disabled is that the following initialization FlexCAN frame is sent for baud measurement purposes from the host to the MCU when autobaud is enabled:

- 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 491*).

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.

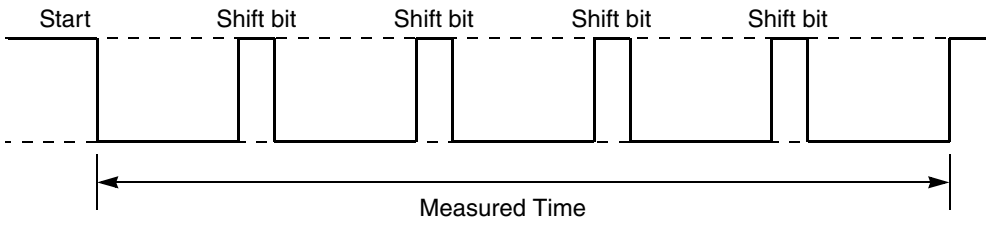


Figure 491. Bit time measure

In FlexCAN 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 the FlexCAN module.

The baud rate measurement flow is detailed in *Figure 492*.

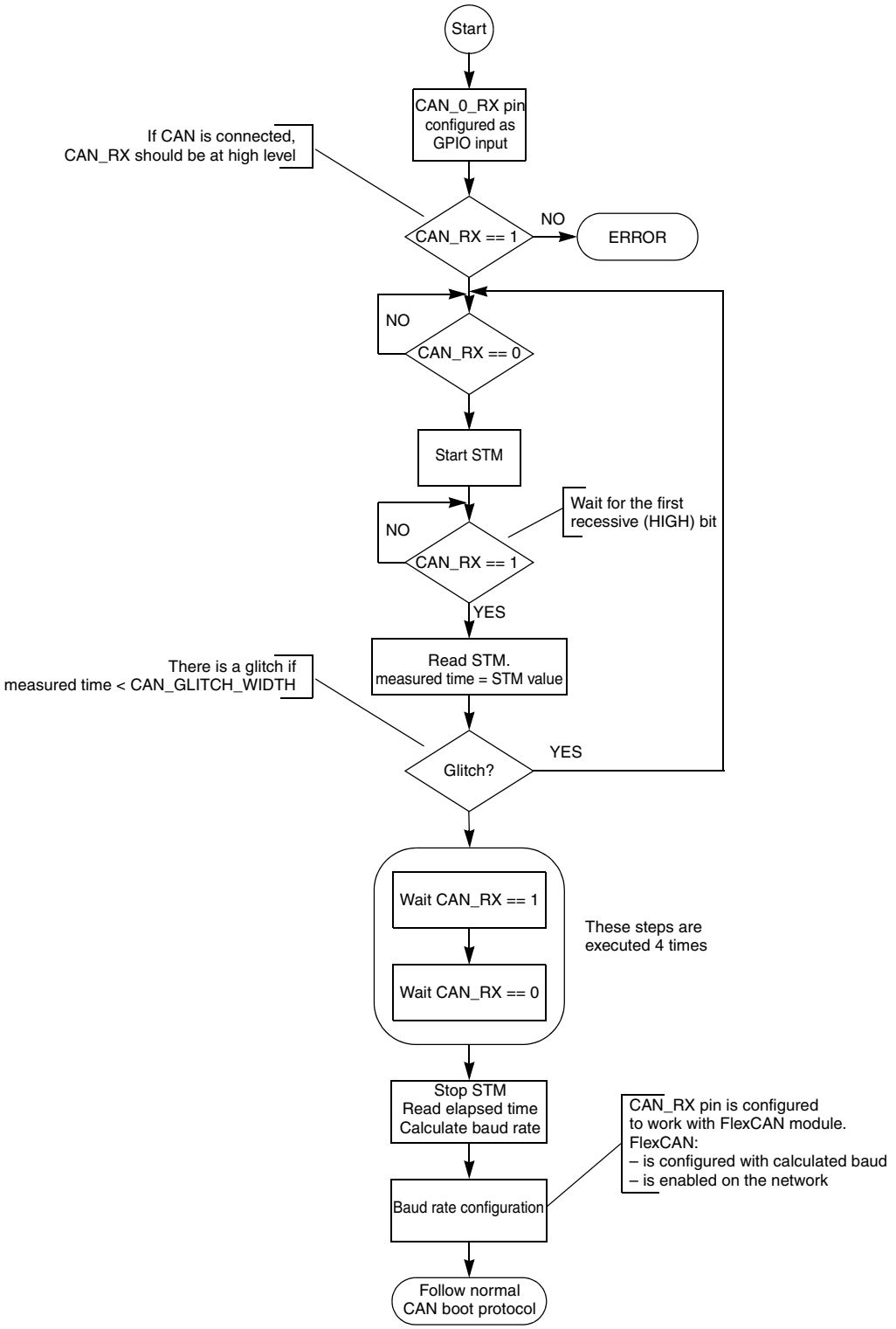


Figure 492. BAM rate measurement flow during FlexCAN boot

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, to ensure proper data transfer, the upper and lower limits have to be kept.

Pins are measured until reception of the 5th recessive bit. Thus a total of 29 bit-times are measured (see *Figure 491*).

When calculating bit times and prescalers, to minimize any errors, ideally operate with the minimum system clock prescaler divider (CAN_CR[PRESDIV]) and maximum number of time quanta possible.

After measuring the 29 bit times, the results stored in the STM time base are 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)

FlexCAN protocol specifies that the FlexCAN bit timing should comprise a minimum of 8 time quanta and a maximum of 25 time quanta. Therefore, the 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 lie, accounting for PRESDIV, is:

$(232 \times (1 + PRESDIV)) \leq time\ base \leq (725 \times (1 + PRESDIV))$

Therefore, the available values of the time base can be divided into windows of 725 counts.

Table 445. 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 will be 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 can baud rates. See *Table 446*.

Table 446. FlexCAN standard compliant bit timing segment settings

Time Segment 1	Time Segment 2	RJW
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

Timing segment 2 is kept as large as possible to keep sample time within bit time.

Table 447. 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 448. 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 449. 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
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

Worked examples showing FlexCAN Autobaud rate:

Example 198 MHz crystal

Consider 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:
– 1MB = 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 give 0% error.

Example 2020 MHz crystal

Consider case where using a 20 MHz crystal, user attempts to send 62.5 Kb/s FlexCAN message.

- Time base, clocking at crystal frequency, would measure:
- 62.5 Kb/s = 320 clocks/bit => 29 * 320 = 9280 clocks
- To calculate PRESDIV = 9280/725 = 12r580 => PRESDIV = 12
- To calculate time quanta requirement:
- Time base result = 29 x (Presdiv+1) x (SYNCSEG + TSEG1 + TSEG2)
- 9280 = 29 x 13 x (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 give a baud rate of 61.538k Baud
- This equates to an error of: ~1.6%

This excludes cycle count error introduced due to software polling likely to be ~6 system clocks.

33.6.2 Interrupt

No interrupts are generated by or are enabled by the BAM.

33.7 Censorship

Censorship can be enabled to protect the contents of the flash memory from being read or modified. In order to achieve this, the censorship mechanism controls access to the:

- JTAG / Nexus debug interface
- Serial boot mode (which could otherwise be used to download and execute code to query or modify the flash memory)

To re-gain access to the flash memory via JTAG or serial boot, a 64-bit password must be correctly entered.

Caution:

When censorship has been enabled, the only way to regain access is with the password. If this is forgotten or not correctly configured, then there is no way back into the device.

There are two 64-bit values stored in the shadow flash which control the censorship (see *Table 145* for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCI0 and NVSCI1

Censorship password registers (NVPWD0 and NVPWD1)

The two private password registers combine to form a 64-bit password that should be programmed to a value known only by you. After factory test these registers are programmed as shown below:

- NVPWD0 = 0xFEED_FACE
- NVPWD1 = 0xCAFE_BEEF

This means that even if censorship was inadvertently enabled by writing to the censorship control registers, there is an opportunity to get back into the microcontroller using the default private password of 0xFEED_FACE_CAFE_BEEF.

When configuring the private password, each half word (16-bit) must contain at least one "1" and one "0". Some examples of legal and illegal passwords are shown in *Table 450*:

Table 450. Examples of legal and illegal passwords

Legal (valid) passwords	Illegal (invalid) passwords
0x0001_0001_0001_0001 0xFFFE_FFFE_FFFE_FFFE 0x1XXX_X2XX_XX4X_XXX8	0x0000_XXXX_XXXX_XXXX 0xFFFF_XXXX_XXXX_XXXX

In uncensored devices it is possible to download code via LINFlex or FlexCAN (Serial Boot Mode) into internal SRAM even if the 64-bit private password stored in the flash and provided during the boot sequence is a password that does not conform to the password rules.

Nonvolatile System Censorship Control registers (NVSCI0 and NVSCI1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below which disables censorship:

- NVSCI0 = 0x55AA_55AA
- NVSCI1 = 0x55AA_55AA

Each 32-bit register is split into an upper and lower 16-bit field. The upper 16 bits (the SC field) are used to control serial boot mode censorship. The lower 16 bits (the CW field) are used to control flash memory boot censorship.

Caution:

If the contents of the shadow flash memory are erased and the NVSCI0,1 registers are not re-programmed to a valid value, the microcontroller will be permanently censored with no way for you to regain access. A microcontroller in this state cannot be debugged or re-flashed.

Censorship configuration

The steps to configuring censorship are:

1. Define a valid 64-bit password that conforms to the password rules.
2. Using the table and flow charts below, decide what level of censorship you require and configure the NVSCI0,1 values.
3. Re-program the shadow flash memory and NVPWD0,1 and NVSCI0,1 registers with your new values. A POR is required before these will take effect.

Caution:

If
(NVSCI0 and NVSCI1 do not match)
or
(Either NVSCI0 or NVSCI1 is not set to 0x55AA)
then the microcontroller will be permanently censored with no way to get back in.

Table 451 shows all the possible modes of censorship. The red shaded areas are to be avoided as these show the configuration for a device that is permanently locked out. If you wish to enable censorship with a private password there is only one valid configuration — to modify the CW field in both NVSCI0,1 registers so they match but do not equal 0x55AA. This will allow you to enter the private password in both serial and flash boot modes.

Table 451. Censorship configuration and truth table

<table>
<tr><th colspan="2">Boot configuration</th><th rowspan="2">Serial censorship control word (NVSCIn[SC])</th><th rowspan="2">Censorship control word (NVSCIn[CW])</th><th rowspan="2">Internal flash memory state</th><th rowspan="2">Nexus state</th><th rowspan="2">Serial password</th><th rowspan="2">JTAG password</th></tr>
<tr><th>FAB pin state</th><th>Control options</th></tr>
<tr><td rowspan="4">0 (flash memory boot)</td><td>Uncensored</td><td>0xXXXX AND NVSCI0 == NVSCI1</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled</td><td rowspan="4"></td><td>N/A</td></tr>
<tr><td>Private flash memory password and censored</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>!0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled with password</td><td>NVPWD1,0 (SSCM reads flash memory(1))</td></tr>
<tr><td rowspan="2">Censored with no password access (lockout)</td><td>!0x55AA</td><td>!0X55AA</td><td rowspan="2">Enabled</td><td rowspan="2">Disabled</td><td rowspan="2">N/A</td></tr>
<tr><td colspan="2">OR NVSCI0 != NVSCI1</td></tr>
<tr><td rowspan="5">1 (serial boot)</td><td>Private flash memory password and uncensored</td><td colspan="2">0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Enabled</td><td>NVPWD0,1 (BAM reads flash memory(1))</td><td rowspan="4"></td></tr>
<tr><td>Private flash memory password and censored</td><td>0x55AA AND NVSCI0 == NVSCI1</td><td>!0x55AA AND NVSCI0 == NVSCI1</td><td>Enabled</td><td>Disabled</td><td>NVPWD1,0 (SSCM reads flash memory(1))</td></tr>
<tr><td>Public password and uncensored</td><td>!0x55AA AND NVSCI0 != NVSCI1</td><td>0X55AA AND NVSCI0 != NVSCI1</td><td>Enabled</td><td>Enabled</td><td>Public (0xFEED_FACE_CAFE_BEEF)</td></tr>
<tr><td rowspan="2">Public password and censored (lockout)</td><td colspan="2">!0x55AA</td><td rowspan="2">Disabled</td><td rowspan="2">Disabled</td><td rowspan="2">Public (0xFEED_FACE_CAFE_BEEF)</td></tr>
<tr><td colspan="2">OR NVSCI0 != NVSCI1</td></tr>
</table>

= Microcontroller permanently locked out

= Not applicable

1. When the SSCM reads the passwords from flash memory, the NVPWD0 and NVPWD1 password order is swapped, so you have to submit the 64-bit password as {NVPWD1, NVPWD0}.

The flow charts in *Figure 493* and *Figure 494* provide a way to quickly check what will happen with different configurations of the NVSCI0,1 registers as well as detailing the correct way to enter the serial password. In the password examples, assume the 64-bit

password has been programmed into the shadow flash memory in the order {NVPWD0, NWPWD1} and has a value of 0x01234567_89ABCDEF.

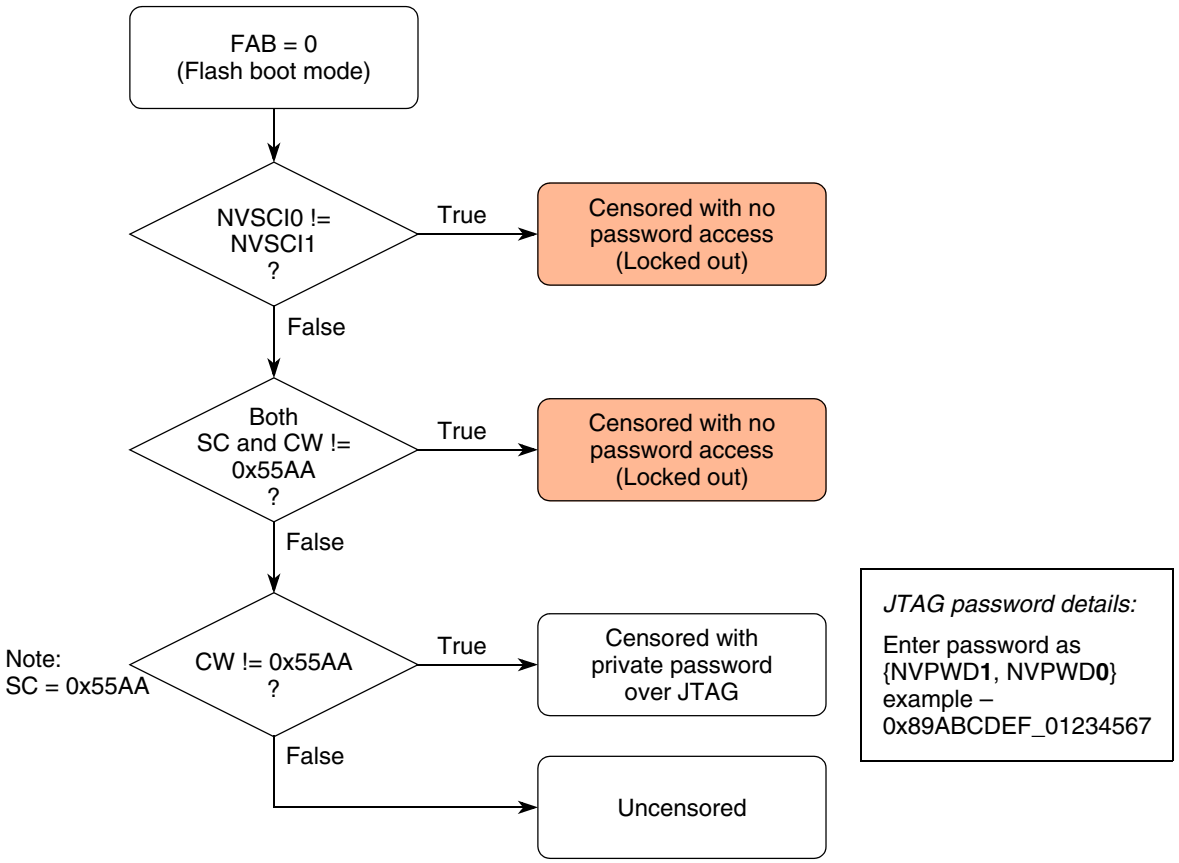


Figure 493. Censorship control in flash memory boot mode

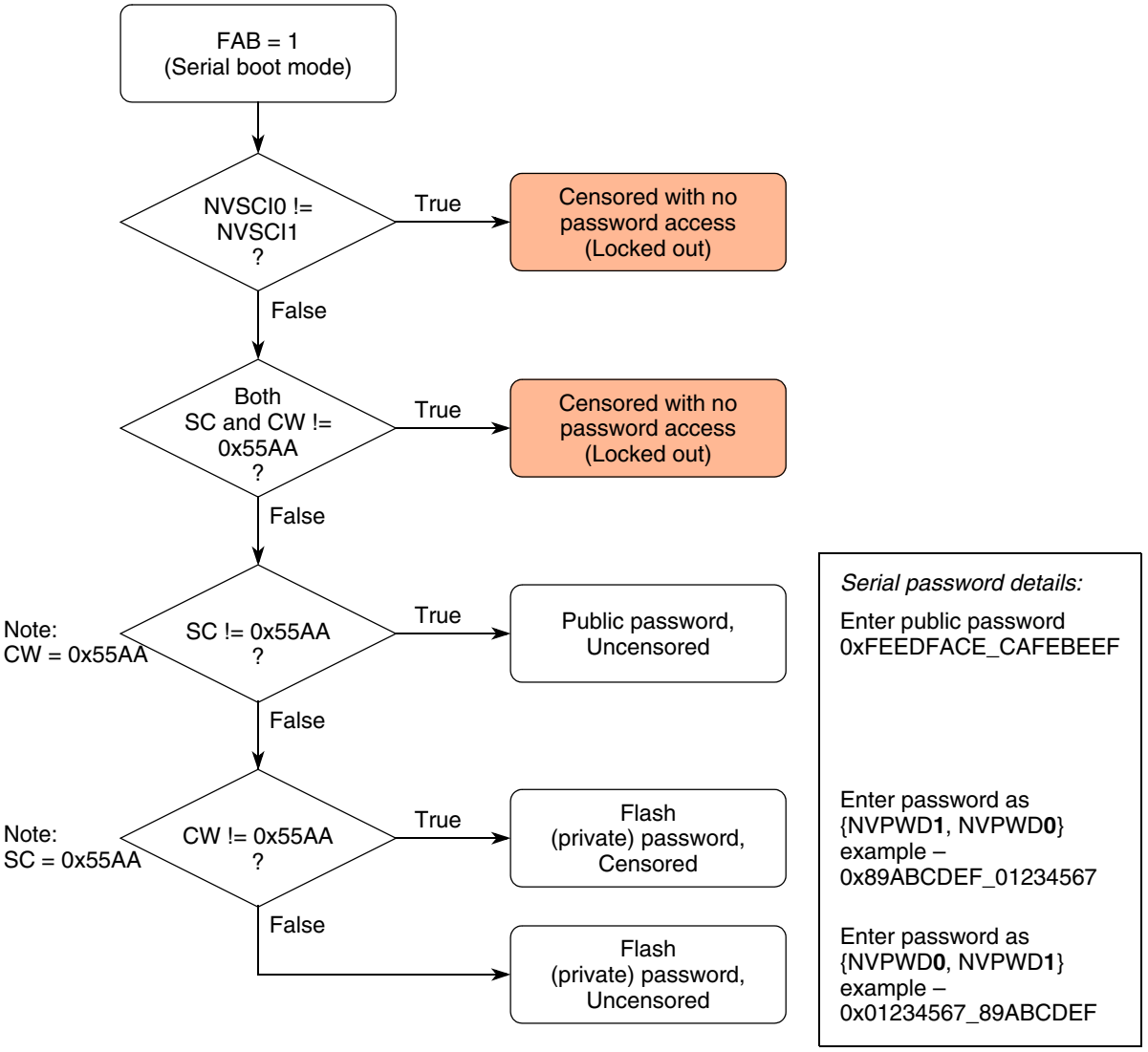


Figure 494. Censorship control in serial boot mode

34 Voltage Regulators and Power Supplies

34.1 Voltage regulator

The power blocks are used for providing 1.2 V digital supply to the internal logic of the device. The main/input supply is 3.3 V to 5.0 V ±10% and the digital/regulated output supply has a trim target voltage of 1.28 V. The voltage regulator used in SPC560P40/34 is the high power or main regulator (HPREG).

The internal voltage regulator requires an external ballast transistor and (external) capacitance (CREG) to be connected to the device in order to provide a stable low voltage digital supply to the device. Capacitances should be placed on the board as near as possible from the associated pins.The regulator has a digital domain called the High Power domain that has a low voltage detector for the 1.2 V output voltage. Additionally, there are two low voltage detectors for the main/input supply with different threshold, one at 3.3 V level and the other one at 5 V level.

34.1.1 High Power or Main Regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. The nominal target output is 1.28 V. Due to all variations, the actual output will be in range of 1.08 V to 1.32 V in the full current load range (0–250 mA) after trimming.

The stabilization for HPREG is achieved using an external capacitance. The minimum recommended value is 3 × 10 µF with low ESR (refer to datasheet for details).

Note: In general an offset voltage must be avoided to pre-charge $V_{DD\_HV\_REG}$ through parasitic paths to allow a correct power up sequence.

The MCU supply must power on from GND to power supply with a monotonic ramp, minimum and maximum values as described in the data sheet (TV_{DD}).

34.1.2 Low Voltage Detectors (LVD) and Power On Reset (POR)

Five types of low voltage detectors are provided on the device:

- $V_{REGLVDMOK\_L}$ monitors the 3.3 V regulator supply
- $V_{FLLVDMOK\_L}$ monitors the 3.3 V flash supply
- $V_{IOLVDMOK\_L}$ monitors the 3.3 V I/O supply
- $V_{IOLVDM5OK\_L}$ monitors the 5 V I/O supply
- $V_{MLVDDOK\_L}$ monitors the 1.2 V digital logic supply

LVD_MAIN is the main voltage LVD with a threshold set around 2.7 V. LVD_MAIN5 is the main voltage LVD with a threshold set around 4 V. The LVD_MAIN and LVD_MAIN5 sense the 3.3 V–5 V power supply for CORE, shared with IO ring supply and indicate when the 3.3 V–5 V supply is stabilized. The threshold levels of LVD_MAIN5 are trimmable with the help of LVDM5[0:3] trim bits.

The LVD_MAIN and LVD_MAIN5 detectors sense the V_{DDIO} supply and provide $V_{IOLVDMOK\_H}$/$V_{IOLVDM5OK\_H}$ and $V_{IOLVDMOK\_L}$/$V_{IOLVDM5OK\_L}$ as active high signals at 3.3 V and 1.2 V supply levels, respectively.

Two more LVD_MAIN detectors are also used for sensing VDDREG and VDDFLASH.

An LVD_DIG in the regulator senses the HPREG output. It provides $V_{MLVDDOK\_H}$ and V_{MLVDOK_L} as active high signals.

The reference voltage used for all LVDs is trimmed for LVD_DIG using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD_DIG exhibits higher thresholds, whereas post trimming, the thresholds come in the desired range. The trimming bits are provided by SSCM device option bits, which are updated during the reset phase (RGM reset phase 2) only. Power-down pins are provided for LVDs. When LVDs are powered down, their outputs are pulled high. LVDs are not controllable by the user. The only option is the possibility to mask LVD_MAIN5 using the mask bit (5V_LVD_MASK) in the VREG_CTL register.

POR is required to initialize the device during supply rise. POR works only on the rising edge of main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of the LVD_MAIN block when main supply reaches below the lower voltage threshold of the LVD_MAIN.

POR is asserted on power-up when V_{DD} supply is above V_{PORUP} minimum (refer to datasheet for details). It is released only after V_{DD} supply is above V_{PORH} (refer to datasheet for details). V_{DD} above V_{PORH} ensures power management module including internal LVDs modules are fully functional.

34.1.3 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and at exit from low-power modes. A signal, indicating that Low Power domain is powered, is used at power-up to release reset to temporization counter. On completion of the delay counter, a end-of-count signal is released, it is gated with an other signal indicating main domain voltage fine in order to release the VREGOK signal. This is used by RGM to release the reset to the device.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

34.1.4 Registers Description

Voltage Regulator Control Register (VREG_CTL)

Figure 495. Voltage Regulator Control register (VREG_CTL)

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	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	1	0	0	0	0	0	0	0	5V_LVD_MASK
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Table 452. VREG_CTL field descriptions

Field	Description
5V_LVD_MASK	Mask bit for 5 V LVD from regulator This is a read/write bit and must be unmasked by writing a 1 by software to generate LVD functional reset request to RGM for 5V trip. 05 V LVD not masked 15 V LVD masked

Voltage Regulator Status register (VREG_STATUS)

Figure 496. Voltage Regulator Status register (VREG_STATUS)

Address: Base + 0x0084 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	1	0	0	0	0	0	0	0	5V_LVD_STATUS
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Table 453. VREG_STATUS field descriptions

Field	Description
5V_LVD_STATUS	Status bit for 5 V LVD from regulator 05 V LVD not OK 15 V LVD OK

34.2 Power supply strategy

The SPC560P40/34 provides three dedicated supply domains at the package level:

- HV—High voltage external power supply for I/Os, voltage regulator module, and most analog modules

  This must be provided externally through V_{DD_HV}/V_{SS_HV} power pins. Voltage values should be aligned with V_{DD}/V_{SS}. Refer to the device datasheet for details.
- ADC—High voltage external power supply for ADC module

  This must be provided externally through $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$ power pins. Voltage values should be aligned with $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$. Refer to the device datasheet for details.
- LV—Low voltage internal power supply for core, PLL, and flash digital logic

  This is provided to the core, PLL and flash. Five V_{DD_LV}/V_{SS_LV} pins pairs are provided to connect the low voltage power supply. Refer to the device datasheet for details.

The three dedicated supply domains are further divided within the package in order to reduce EMI and noise as much as possible:

- HV_REG—High voltage regulator supply
- HV_IO*n*—High voltage PAD supply
- HV_OSC[e]—High voltage external oscillator and regulator supply
- HV_AD0—High voltage supply and reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- LV_COR*n*—Low voltage supply for the core. It is also used to provide supply for PLL and Flsah memory through double bonding.

e.Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing.

35 IEEE 1149.1 Test Access Port Controller (JTAGC)

35.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test mode select (TMS), test clock input (TCK) and test data output (TDO). TDI, TMS, TCK and TDO are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

35.2 Block diagram

Figure 497 is a block diagram of the JTAG Controller (JTAGC).

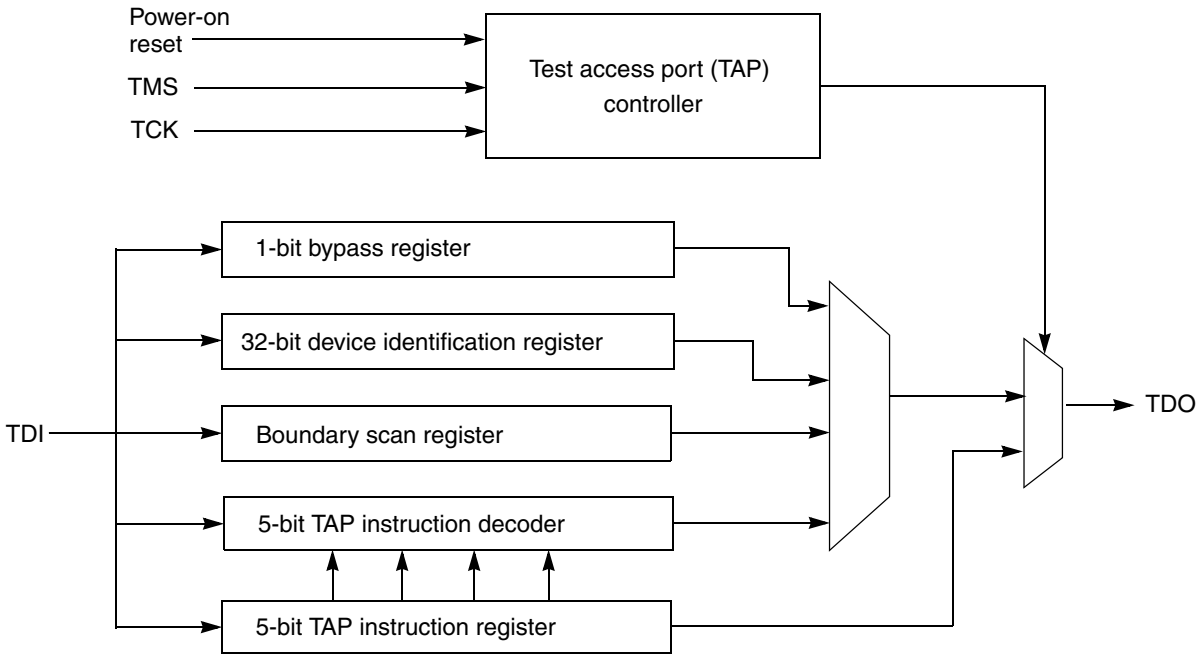


Figure 497. JTAG controller block diagram

35.3 Overview

The JTAGC 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. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

35.4 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- Four pins (TDI, TMS, TCK, and TDO)—see *Section 35.6, "External signal description.*
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific 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.

35.5 Modes of operation

The JTAGC uses 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.

35.5.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset 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.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

35.5.2 IEEE 1149.1-2001 defined test modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The 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 registers that can 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 35.8.4, "JTAGC instructions.*

Bypass mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC 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.

TAP sharing mode

There are four selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200 OnCE, and eDMA Nexus. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_ONCE and ACCESS_AUX_TAP_NPC. Instruction opcodes for each instruction are shown in *Table 456*.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is 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.

For more information on the TAP controllers refer to *36, "Nexus Development Interface (NDI)."*

35.6 External signal description

The JTAGC consists of four signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 454*.

Table 454. JTAG signal properties(1)

Name	I/O	Function	Reset state	Pull(2)
TCK	I	Test clock	—	Down
TDI	I	Test data in	—	Up
TDO	O	Test data out	High Z(3)	Down(3)
TMS	I	Test mode select	—	Up

1. Test clock frequency must always be less than one fourth of system clock frequency.
2. The pull is not implemented in this module. Pullup/down devices are implemented in the pads.
3. TDO output buffer enable is negated when JTAGC is not in the Shift-IR or Shift-DR states. A weak pulldown can be implemented on TDO.

35.7 Memory map and registers description

This section provides a detailed description of the JTAGC 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.

35.7.1 Instruction register

The JTAGC uses a 5-bit instruction register as shown in *Figure 498*. The instruction register allows instructions to be loaded into the module 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 498. 5-bit Instruction register

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

35.7.2 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.

35.7.3 Device identification register

The device identification register, shown in *Figure 499*, allows the part revision number, design center, part identification number, and manufacturer identity code 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.

Figure 499. Device identification register

IR[4:0]: 0_0001 (IDCODE) Access: User read-only

	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	PRN	DC		PIN		MIC		ID
W								
Reset	0 0 0 0	1 0 1 0	1 1 1 0	0 0 1 0	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 1

Table 455. Device identification register field descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. For the SPC560P40/34 this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC560P40/34, this value is 0x222.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for STMicroelectronics, 0x20.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

35.7.4 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 35.8.5, "Boundary scan.* The size of the boundary scan register is 464 bits.

35.8 Functional description

35.8.1 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.

35.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port (TAP)

The JTAGC uses the IEEE 1149.1-2001 Test Access Port (TAP) for accessing registers. This port can be shared with other TAP controllers on the MCU. For more detail on TAP sharing via JTAGC instructions refer to *Section , "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 500*. This applies for the instruction register, test data registers, and the bypass register.

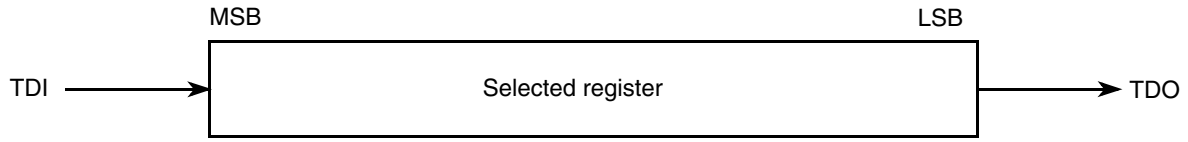
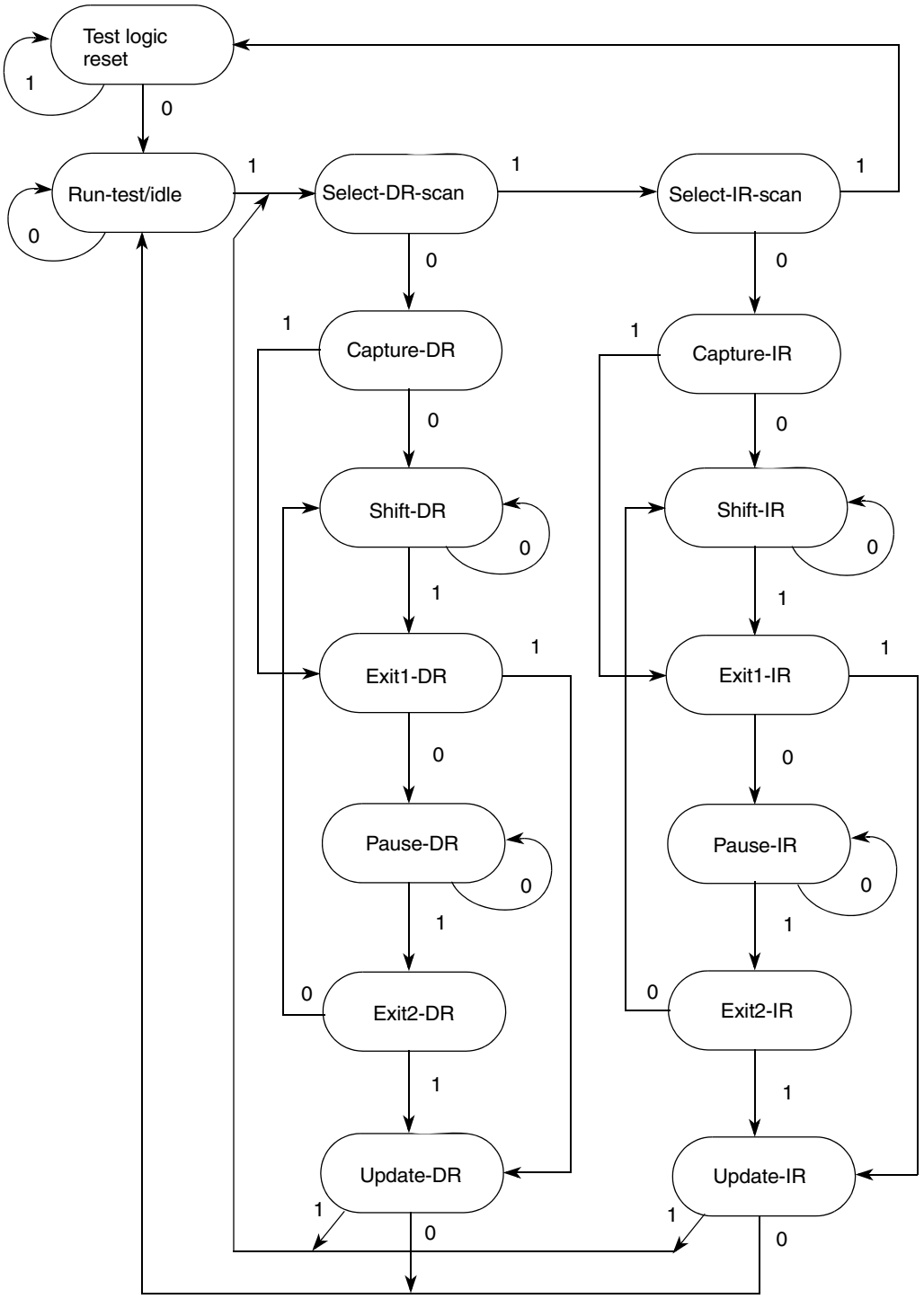


Figure 500. Shifting data through a register

35.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 501* 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 501* 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 501. IEEE 1149.1-2001 TAP controller finite state machine

Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC 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 can be terminated after fetching the required number of bits.

35.8.4 JTAGC instructions

This section gives an overview of each instruction. Refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in *Table 456*.

Table 456. 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
HIGHZ	01001	Selects bypass register while tristating 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_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_CORE0	10001	Enables access to Core_0 TAP controller
ACCESS_AUX_TAP_CORE1	11001	Enables access to Core_1 TAP controller
ACCESS_AUX_TAP_NASPS_0	10111	Selects the ACCESS_AUX_NASPS_0 configuration that connects the auxiliary TAP interface to the Nexus SRAM Port sniffer connected between the XBAR_0 and the SRAMC_0
ACCESS_AUX_TAP_NASPS_1	11000	Selects the ACCESS_AUX_NASPS_1 configuration that connects the auxiliary TAP interface to the Nexus SRAM Port sniffer connected between the XBAR_0 and the SRAMC_1
BYPASS	11111	Selects bypass register for data operations

Table 456. JTAG instructions (continued)

Instruction	Code[4:0]	Instruction summary
Factory Debug Reserved[1]	00101 00110 01010	Intended for factory debug only
Reserved	All Other Codes	Decoded to select bypass register

1. Intended for factory debug, and not customer use.

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.

ACCESS_AUX_TAP_*x* instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary 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.

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.

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.

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.

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 is reset.

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.

SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on 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.
- The PRELOAD part 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.

35.8.5 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.

35.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug. A complete discussion of the e200z0 OnCE debug features is available in the core reference manual.

35.9.1 e200z0 OnCE controller block diagram

Figure 502 is a block diagram of the e200z0 OnCE block.

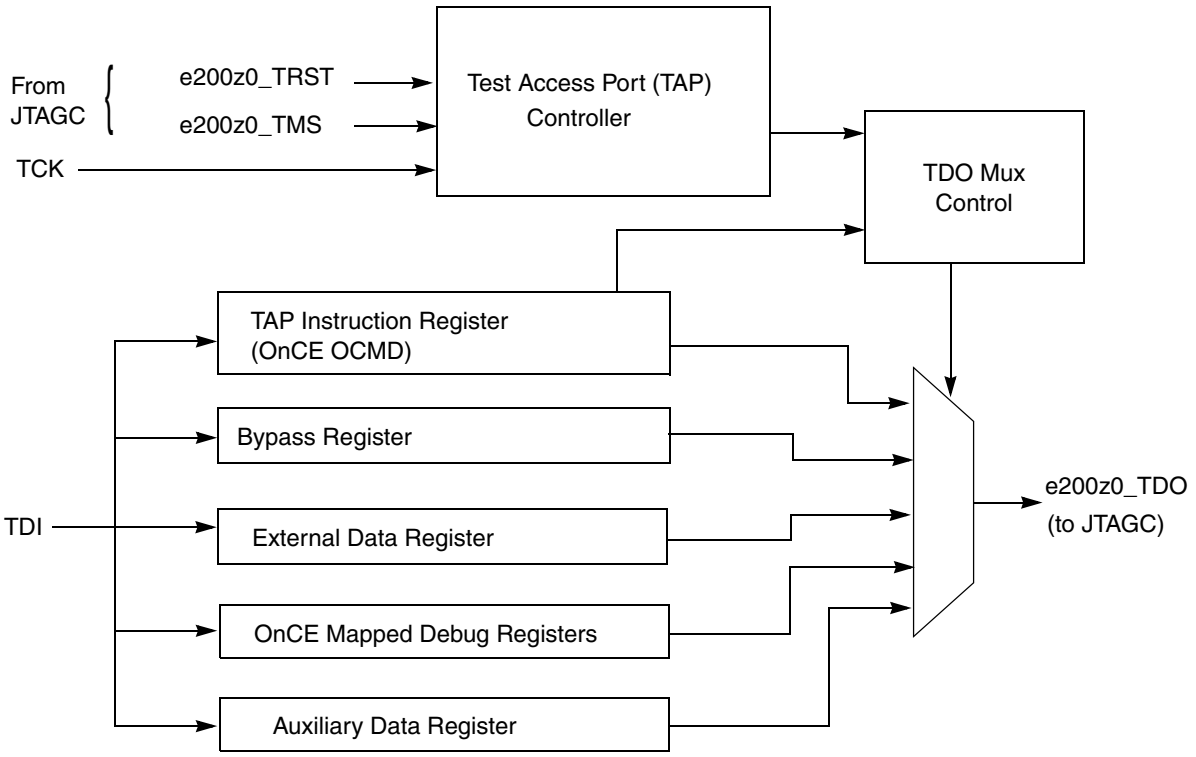


Figure 502. e200z0 OnCE block diagram

35.9.2 e200z0 OnCE controller functional description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described as follows.

Enabling the TAP controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in *Section , "TAP sharing mode.* The e200z0 OnCE TAP controller may either be accessed independently or chained with the e200z1 OnCE TAP controller, such that the TDO output of the e200z1 TAP controller is fed into the TDI input of the e200z0 TAP controller. The chained configuration allows commands to be loaded into both core's OnCE registers in one shift operation, so that both cores can be sent a GO command at the same time for example.

35.9.3 e200z0 OnCE controller registers description

Most e200z0 OnCE debug registers are fully documented in the core reference manual.

OnCE Command register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in *Figure 503*. The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for

controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

Figure 503. OnCE Command register (OCMD)

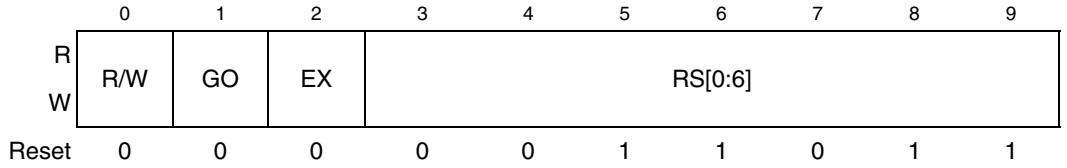


Table 457. e200z0 OnCE register addressing

RS[0:6]	Register selected
000 0000 000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)
010 0111	Data Value Compare 2 (DVC2)
010 1000 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100 – 101 1111	Reserved (do not access)

Table 457. e200z0 OnCE register addressing (continued)

RS[0:6]	Register selected
110 1111	Shared Nexus Control Register (SNC) (only available on the e200z0 core)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010 – 111 1011	Reserved
111 1100	Reserved
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

35.10 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 module and enable access to registers, the following sequence is required:

1. Place the JTAGC in reset through TAP controller state machine transitions controlled by TMS
   – Load the appropriate instruction for the test or action to be performed.

36 Nexus Development Interface (NDI)

36.1 Introduction

The Nexus Development Interface (NDI) block provides development support capabilities for the SPC560P40/34 MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility.

The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for SPC560P40/34.

The NDI block interfaces to the e200z0, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes the MCU's internal memory map, and access to the e200z0 internal registers, via the JTAG port.

36.2 Information specific to this device

This section presents device-specific information not specifically referenced in the remainder of this chapter.

36.2.1 Features not supported

The following features are mentioned in this chapter but are not supported by the device:

- MMU (memory management unit)
- Nexus2 module
- Nexus3 module

36.3 Block diagram

Figure 504 shows a functional block diagram of the NDI.

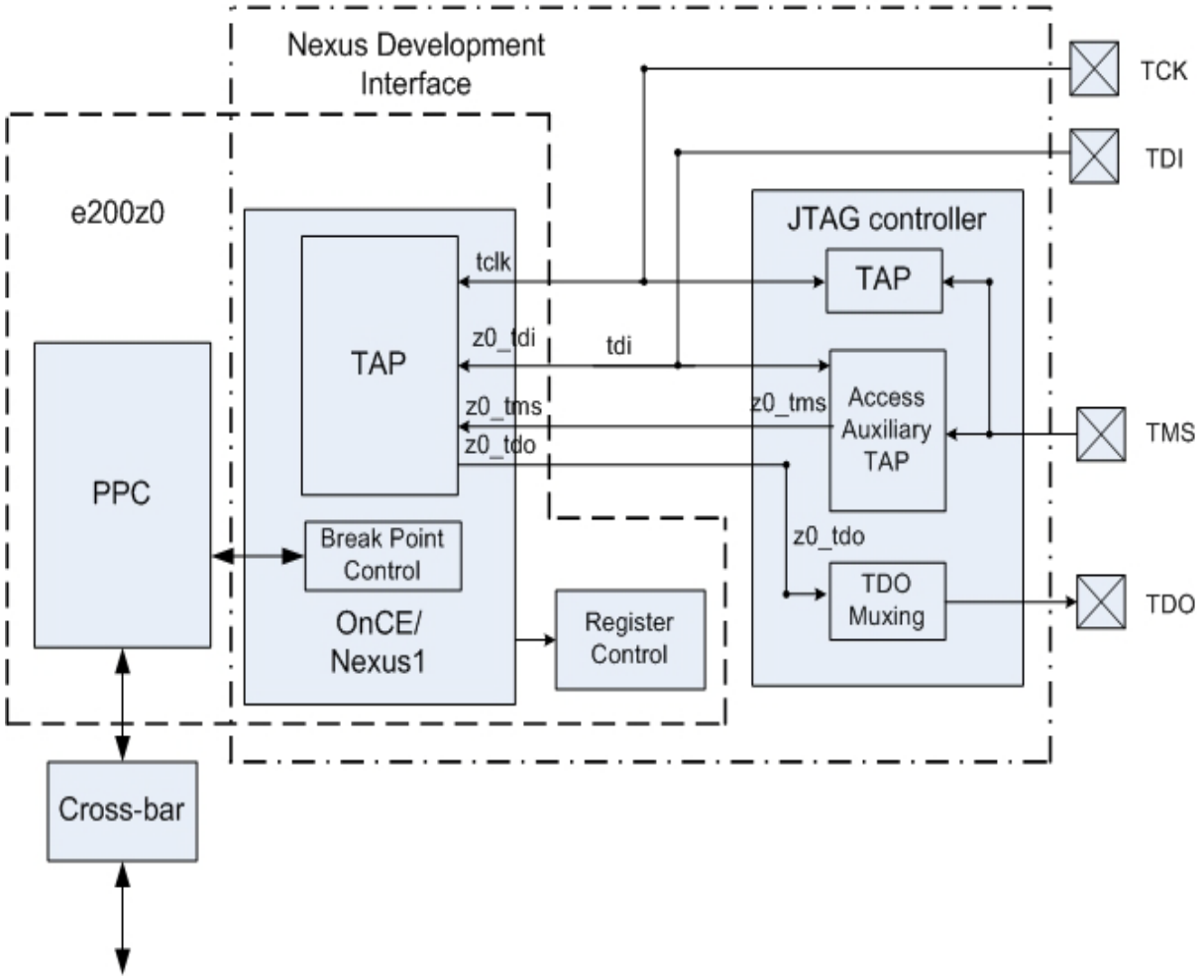


Figure 504. NDI functional block diagram

36.4 Features

The NDI module of the SPC560P40/34 is compliant with Class 1 of the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- 4-pin JTAG port (TDI, TDO, TMS, and TCK)
- All features controllable and configurable via the JTAG port
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- Support for internal censorship mode to prevent external access to flash memory contents when censorship is enabled

Note: If the e200z0 core has executed a wait instruction, then the Nexus1 controller clocks are gated off. While the core is in this state, it is not possible to perform Nexus read/write operations.

36.5 Modes of operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal or through state machine transitions controlled by TMS. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block.

The NPC transitions out of the reset state immediately following negation of power-on reset.

36.5.1 Nexus reset

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to tristate the output pins or use them for another function.

36.5.2 NDI modes

Censored mode

The NDI supports internal flash censorship mode by preventing the transmission of trace messages and Nexus access to memory-mapped resources when censorship is enabled.

Stop mode

Stop mode logic is implemented in the Nexus port controller (NPC). When a request is made to enter stop mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the stop request. After the acknowledgment, the system clock input are shut off by the clock driver on the device. While the clocks are shut off, the development tool cannot access NDI registers via the JTAG port.

36.6 External signal description

All signals are shared by all the individual blocks that make up the NDI block. The Nexus port controller (NPC) block controls the signal sharing.

Refer to *3, "Signal Description* for detailed signal descriptions.

36.7 Memory map and registers description

The NDI block contains no memory-mapped registers. Nexus registers are accessed by a development tool via the JTAG port using a client-select value and a register index. OnCE registers are accessed by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD) via the JTAG port.

36.8 Interrupts and Exceptions

The Power Architecture technology defines the mechanisms by which the e200z0h core implements interrupts and exceptions. The document uses the terminology *Interrupt* as the action in which the processor saves its old context and begins execution at a predetermined interrupt handler address. *Exceptions* are referred to as events, which when enabled, will cause the processor to take an interrupt. This section uses the same terminology.

The Power Architecture exception mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions, and from external signals, bus errors, or various internal conditions. When interrupts occur, information about the state of the processor is saved to machine state save/restore registers (SRR0/SRR1, CSRR0/CSRR1, or DSRR0/DSRR1) and the processor begins execution at an address (interrupt vector) determined by the Interrupt Vector Prefix register (IVPR), and one of the hardwired Interrupt Vector Offset values. Processing of instructions within the interrupt handler begins in supervisor mode.

Multiple exception conditions can map to a single interrupt vector, and may be distinguished by examining registers associated with the interrupt. The Exception Syndrome register (ESR) is updated with information specific to the exception type when an interrupt occurs.

To prevent loss of state information, interrupt handlers must save the information stored in the machine state save/restore registers, soon after the interrupt has been taken. Three sets of these registers are implemented; SRR0 and SRR1 for non-critical interrupts, CSRR0 and CSRR1 for critical interrupts, and DSRR0 and DSRR1 for debug interrupts (when the Debug APU is enabled). Hardware supports nesting of critical interrupts within non-critical interrupts, and debug interrupts within both critical and non-critical interrupts. It is up to the interrupt handler to save necessary state information if interrupts of a given class are re-enabled within the handler.

The following terms are used to describe the stages of exception processing:

- Recognition—Exception recognition occurs when the condition that can cause an exception is identified by the processor. This is also referred to as an exception event.
- Taken—An interrupt is said to be taken when control of instruction execution is passed to the interrupt handler; that is, the context is saved and the instruction at the appropriate vector offset is fetched and the interrupt handler routine begins.
- Handling—Interrupt handling is performed by the software linked to the appropriate vector offset. Interrupt handling is begun in supervisor mode.

Returning from an interrupt is performed by executing an **se_rfi**, **se_rfci**, or **se_rfdi** instruction to restore state information from the respective machine state save/restore register pair.

36.9 Debug support overview

Internal debug support in the e200z0h core allows for software and hardware debug by providing debug functions, such as instruction and data breakpoints and program trace modes. For software based debugging, debug facilities consisting of a set of software accessible debug registers and interrupt mechanisms are provided. These facilities are also available to a hardware based debugger which communicates using a modified IEEE 1149.1 Test Access Port (TAP) controller and pin interface. When hardware debug is enabled, the debug facilities controlled by hardware are protected from software modification.

Software debug facilities are built on Power Architecture technology. e200z0h supports a subset of these defined facilities. In addition to the facilities built on Power Architecture technology, e200z0h provides additional flexibility and functionality in the form of linked instruction and data breakpoints, and sequential debug event detection. These features are also available to a hardware-based debugger.

The e200z0h core provides support for a Nexus real-time debug module. Real-time debugging in an e200z0h-based system is supported by a Nexus class 2, 3, or 4 module.

36.9.1 Software Debug Facilities

e200z0h provides debug facilities to enable hardware and software debug functions, such as instruction and data breakpoints and program single stepping. The debug facilities consist of a set of debug control registers (DBCR0–2, DBCR4, DBERC0), a set of address compare registers (IAC1, IAC2, IAC3, IAC4, DAC1, and DAC2), a set of data value compare registers (DVC1, DVC2), a Debug Status Register (DBSR) for enabling and recording various kinds of debug events, and a special Debug interrupt type built into the interrupt mechanism. The debug facilities also provide a mechanism for software-controlled processor reset in a debug environment.

Software debug facilities are enabled by setting the internal debug mode bit in Debug Control register 0 ($DBCR0_{IDM}$). When internal debug mode is enabled, debug events can occur, and can be enabled to record exceptions in the Debug Status register (DBSR). If enabled by MSR_{DE}, these recorded exceptions cause Debug interrupts to occur. When $DBCR0_{IDM}$ is cleared, (and $DBCR0_{EDM}$ is cleared as well), no debug events occur, and no status flags are set in DBSR unless already set. In addition, when $DBCR0_{IDM}$ is cleared (or is overridden by $DBCR0_{EDM}$ being set and DBERC0 indicating no resource is “owned” by software) no Debug interrupts will occur, regardless of the contents of DBSR. A software Debug interrupt handler may access all system resources and perform necessary functions appropriate for system debug.

Power Architecture technology compatibility

The e200z0h core implements a subset of the Power Architecture solutions internal debug features. The following restrictions on functionality are present:

- Instruction address compares do not support compare on physical (real) addresses.
- Data address compares do not support compare on physical (real) addresses.

36.9.2 Additional Debug Facilities

In addition to the debug functionality built on Power Architecture technology, e200z0h provides capability to link instruction and data breakpoints, and also provides a sequential breakpoint control mechanism.

e200z0h also defines two new debug events (CIRPT, CRET) for debugging around critical interrupts.

In addition, e200z0h implements the Debug APU, which when enabled allows Debug Interrupts to utilize a dedicated set of save/restore registers (DSRR0, DSRR1) for saving state information when a Debug Interrupt occurs, and for restoring this state information at the end of a debug interrupt handler by means of the **se_rfdi** instruction.

The e200z0h also provides the capability of sharing resources between hardware and software debuggers. See *Section 36.9.4, "Sharing Debug Resources by Software/Hardware*.

36.9.3 Hardware Debug Facilities

The e200z0h core contains facilities that allow for external test and debugging. A modified IEEE 1149.1 control interface is used to communicate with the core resources. This interface is implemented through a standard 1149.1 TAP (test access port) controller.

By using public instructions, the external debugger can freeze or halt the e200z0h core, read and write internal state and debug facilities, single-step instructions, and resume normal execution.

Hardware Debug is enabled by setting the External Debug Mode enable bit in Debug Control register 0 ($DBCR0_{EDM}$). Setting $DBCR0_{EDM}$ overrides the Internal Debug Mode enable bit $DBCR0_{IDM}$ unless resources are provided back to software via the settings in DBERC0. When the Hardware Debug facility is enabled, software is blocked from modifying the "hardware-owned" debug facilities. In addition, since resources are "owned" by the hardware debugger, inconsistent values may be present if software attempts to read "hardware-owned" debug-related resources.

When hardware debug is enabled by setting $DBCR0_{EDM}$=1, the registers and resources described in *Section 36.11, "Debug Registers* are reserved for use by the external debugger. The same events described in *Section 36.10, "Software Debug Events and Exceptions* are also used for external debugging, but exceptions are not generated to running software. Debug events enabled in the respective DBCR[0-2] registers are recorded in the DBSR regardless of MSR_{DE}, and no debug interrupts are generated unless a resource is granted back to software via DBERC0 settings. Instead, the CPU will enter debug mode when an enabled event causes a DBSR bit to become set. $DBCR0_{EDM}$ and DBERC0 may only be written through the OnCE port.

Access to most debug resources (registers) requires that the CPU clock (**m_clk**) be running in order to perform write accesses from the external hardware debugger.

36.9.4 Sharing Debug Resources by Software/Hardware

Debug resources may be shared by a hardware debugger and software debug based on the settings of debug control register DBERC0. When $DBCR0_{EDM}$ is set, DBERC0 settings determine which debug resources are allocated to software and which resources remain under exclusive hardware control. Software-owned resources which set DBSR bits when $DBCR0_{IDM}$=1 will cause a debug interrupt to occur when enabled with MSR_{DE}. Hardware-owned resources which set DBSR bits when $DBCR0_{EDM}$=1 will cause an entry into debug mode. DBERC0 is read-only by software. When resource sharing is enabled, ($DBCR0_{EDM}$=1 and $DBERC0_{IDM}$=1), only software-owned resources may be modified by software, and all status bits associated with hardware-owned resources will be forced to '0' in DBSR when read by software via a **mfspr** instruction. Hardware always has full access to

all registers and all register fields through the OnCE register access mechanism, and it is up to the debug firmware to properly implement modifications to these registers with read-modify-write operations to implement any control sharing with software. Settings in DBERC0 should be considered by the debug firmware in order to preserve software settings of control and status registers as appropriate when hardware modifications to the debug registers is performed.

Simultaneous Hardware and Software Debug Event Handing

Since it is possible that a hardware “owned” resource can produce a debug event in conjunction with a software-owned resource producing a different debug event simultaneously, a priority ordering mechanism is implemented which guarantees that the hardware event is handled as soon as possible, while preserving the recognition of the software event. The CPU will give highest priority to the software event initially in order to reach a recoverable boundary, and then will give highest priority to the hardware event in order to enter debug mode as near the point of event occurrence as possible. This is implemented by allowing software exception handing to begin internal to the CPU and to reach the point where the current program counter and MSR values have been saved into DSRR0/1, and the new PC pointing to the debug interrupt handler, along with the new MSR updates. At this point, hardware priority takes over, and the CPU enters debug mode.

Figure 505 shows the e200z0h debug resources.

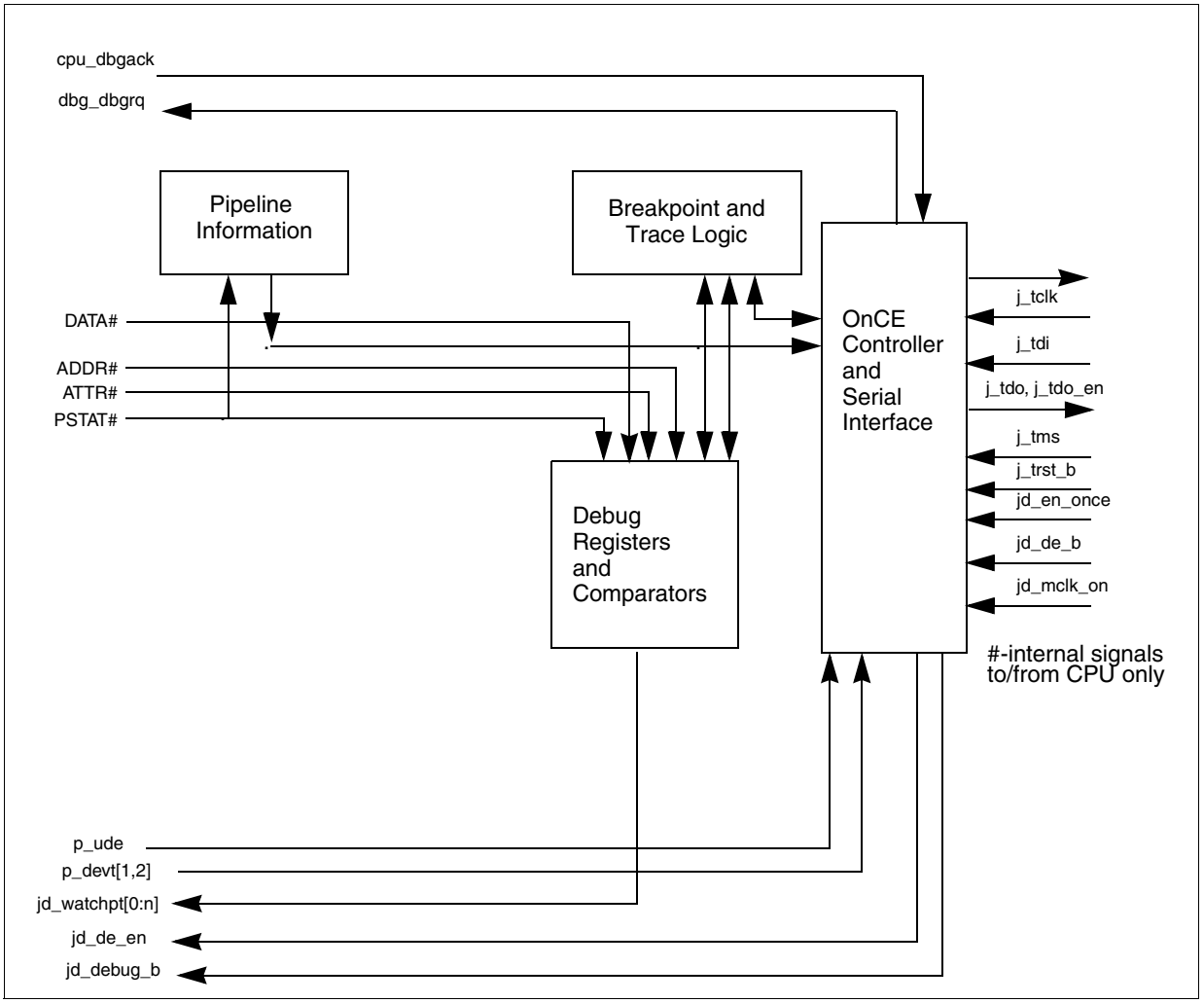


Figure 505. e200z0h Debug Resources

36.10 Software Debug Events and Exceptions

Software debug events and exceptions are available when internal debug mode is enabled ($DBCR0_{IDM}$=1) and not overridden by external debug mode ($DBCR0_{EDM}$ must either be cleared or corresponding resources must be allocated to software debug by the settings in DBERC0). When enabled, debug events cause debug exceptions to be recorded in the Debug Status Register. Specific event types are enabled by the Debug Control Registers (DBCR0-2). The Unconditional Debug Event (UDE) is an exception to this rule; it is always enabled. Once a Debug Status Register (DBSR) bit is set corresponding to a debug resource which is “owned” by software (other than MRR), if Debug interrupts are enabled by MSR_{DE}, a Debug interrupt will be generated. The debug interrupt handler is responsible for ensuring that multiple repeated debug interrupts do not occur by clearing the DBSR as appropriate.

Certain debug events are not allowed to occur when MSR_{DE}=0 and $DBCR0_{EDM}$=0. In such situations, no debug exception occurs and thus no DBSR bit is set. Other debug events may

cause debug exceptions and set DBSR bits regardless of the state of MSR_{DE}. A Debug interrupt will be delayed until MSR_{DE} is later set to '1'.

When a Debug Status Register bit is set while MSR_{DE}=0, and $DBCR0_{EDM}$=0 or $DBCR0_{EDM}$=1 and the corresponding resource is owned (via DBERC0) by software debug, an Imprecise Debug Event flag ($DBSR_{IDE}$) will also be set to indicate that an exception bit in the Debug Status Register was set while Debug interrupts were disabled. Debug interrupt handler software can use this bit to determine whether the address recorded in Debug Save/Restore Register 0 is an address associated with the instruction causing the debug exception, or the address of the instruction which enabled a delayed Debug interrupt by setting the MSR_{DE} bit. A **mtmsr** or **mtdbcr0** which causes both MSR_{DE} and $DBCR0_{IDM}$ to become set, enabling precise debug mode, may cause an Imprecise (Delayed) Debug exception to be generated due to an earlier recorded event in the Debug Status register.

There are eight types of debug events defined by Power Architecture technology:

2. Instruction Address Compare debug events
3. Data Address Compare debug events
4. Trap debug events
5. Branch Taken debug events
6. Instruction Complete debug events
7. Interrupt Taken debug events
8. Return debug events
9. Unconditional debug events

In addition, e200z0h defines additional debug events:

- The External debug events DEVT1 and DEVT2 which are described in *Section 36.10.11, "External Debug Event*.
- The Critical Interrupt Taken debug event CIRPT which is described in *Section 36.10.8, "Critical Interrupt Taken Debug Event*.
- The Critical Return debug event CRET which is described in *Section 36.10.10, "Critical Return Debug Event*.

The e200z0h debug configuration supports most of these event types. Unsupported Power Architecture technology functionality is as follows:

- Instruction Address Compare and Data Address Compare *Real address* mode is not supported

A brief description of each of the event types follows. In these descriptions, DSRR0 and DSRR1 are used, assuming that the Debug APU is enabled. If it is disabled, use CSRR0 and CSRR1 respectively.

36.10.1 Instruction Address Compare Event

Instruction Address Compare debug events occur when enabled and execution is attempted of an instruction at an address that meets the criteria specified in the DBCR0, DBCR1, IAC1, IAC2, IAC3, and IAC4 Registers. Instruction Address compares may specify user/supervisor mode and instruction space (MSR_{IS}), along with an effective address, masked effective address, or range of effective addresses for comparison. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. IAC events will not occur when an instruction would not have normally begun execution due to a higher priority exception at an instruction boundary.

IAC compares perform a 31-bit compare for VLE instructions. Each halfword fetched by the instruction fetch unit will be marked with a set of bits indicating whether an Instruction Address Compare occurred on that halfword. Debug exceptions will occur if enabled and a 16-bit instruction, or the first halfword of a 32-bit instruction, is tagged with an IAC hit.

36.10.2 Data Address Compare Event

Data Address Compare debug events occur when enabled and execution of a load or store class instruction results in a data access that meets the criteria specified in the DBCR0, DBCR2, DBCR4, DAC1, DAC2, DVC1, and DVC2 Registers. Data address compares may specify user/supervisor mode and data space (MSR_{DS}), along with an effective address, masked effective address, or range of effective addresses for comparison. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. Two address compare values (DAC1, DAC2) are provided.

Note: In contrast to the Power Architecture technology, Data Address Compare events on e200z0h do not prevent the load or store class instruction from completing. If a load or store class instruction completes successfully without a Data TLB or Data Storage interrupt, Data Address Compare exceptions are reported at the completion of the instruction. If the exception results in a precise Debug interrupt, the address value saved in DSRR0 (or CSRR0 if the Debug APU is disabled) is the address of the instruction following the load or store class instruction. For DVC DAC events, the exception can be imprecisely reported even further past the load or store class instruction generating the event (without necessarily affecting $DBSR_{IDE}$) and the saved address value can point to a subsequent instruction past the next instruction. This occurrence is indicated in the $DBSR_{DAC\_OFST}$ field.

If a load or store class instruction does not complete successfully due to a Data Storage exception, and a Data Address Compare debug exception also occurs, the result is an imprecise Debug interrupt, the address value saved in DSRR0 (or CSRR0 if the Debug APU is disabled) is the address of the load or store class instruction, and the $DBSR_{IDE}$ bit will be set. In addition to occurring when $DBCR0_{IDM}$=1, this circumstance can also occur when $DBCR0_{EDM}$=1.

*Note: DAC events will not be recorded or counted if a **lmw** or **stmw** instruction is interrupted prior to completion by a critical input or external input interrupt.*

Note: DAC events are not signaled on the second portion of a misaligned load or store that is broken up into two separate accesses.

*Note: DAC[1,2] events are not signaled if DVC[1,2]M is non-zero and a DSI or DTLB exception occurs on the load or store, since the load or store access is not performed. For a **lmw** or **stmw** transfer however, if a DVC successfully occurs on a transfer and a later transfer encounters a DSI or DTLB exception, the DAC event will be reported, since a successful data value compare took place(1).*

Data Address Compare Event Status Updates

Data Address Compare debug events with Data Value compares can be reported ambiguously in several circumstances involving back-to-back load or store class instructions. If the first load or store class instruction generates a DVC DAC, a second load or store class instruction may also generate a DAC or DVC DAC event, or may generate A DTLB or DSI exception with or without a simultaneous DAC event. *Table 459* outlines the

settings of the DBSR, DSRR0 saved value, and potential updating of the ESR and MMU MASx registers for various exception cases on back-to-back load/store class instructions.

Table 458. DAC events and Resultant Updates

1st load/store class instruction	2nd load/store class instruction	Result
DTLB Error, no DAC	—	Take DTLB exception, no DBSR update, update MASx registers for 1st load/store class instruction. Update ESR.
DSI, no DAC	—	Take DSI exception, no DBSR update, no MASx register update. Update ESR.
DTLB Error, with DACx(1)	—	Take Debug exception, DBSR update setting DACx and IDE, DAC_OFST not set. No MASx register update for 1st load/store class instruction. DSRR0 points to 1st load/store class instruction. No ESR update.
DSI, with DACx(1)	—	Take Debug exception, DBSR update setting DACx and IDE, DAC_OFST not set. DSRR0 points to 1st load/store class instruction. No MASx register update. No ESR update.
DACx	—	Take Debug exception, DBSR update setting DACx, DAC_OFST not set. DSRR0 points to 2nd load/store class instruction. No MASx register update. No ESR update.
DVC DACx	No exceptions	Take Debug exception, DBSR update setting DACx, DAC_OFST set to 2'b01. DSRR0 points to instruction after 2nd load/store class instruction. No MASx register update. No ESR update.
DVC DACx	DTLB Error, no DAC	Take Debug exception, DBSR update setting DACx, DAC_OFST not set. DSRR0 points to 2nd load/store class instruction. No MASx register update. No ESR update. No debug counter updates for 2nd ld/st instruction. Note: in this case the 2nd ld/st exception is masked. This behavior is implementation dependent and may differ on other CPUs.
DVC DACx	DSI, no DAC	Take Debug exception, DBSR update setting DACx, DAC_OFST not set. DSRR0 points to 2nd load/store class instruction. No MASx register update. No ESR update. No debug counter updates for 2nd ld/st instruction. Note: in this case the 2nd ld/st exception is masked. This behavior is implementation dependent and may differ on other CPUs.
DVC DACx	DTLB Error, with DACy(1)	Take Debug exception, DBSR update setting DACx. DAC_OFST not set. DSRR0 points to 2nd load/store class instruction. No MASx register update. No ESR update. No debug counter update occurs for the 2nd ld/st. Note: in this case the 2nd ld/st exception is masked. This behavior is implementation dependent and may differ on other CPUs.

Table 458. DAC events and Resultant Updates (continued)

1st load/store class instruction	2nd load/store class instruction	Result
DVC DACx	DSI, with DACy(1)	Take Debug exception, DBSR update setting DACx. DAC_OFST not set. DSRR0 points to 2nd load/store class instruction. No MASx register update. No ESR update. No debug counter update occurs for the 2nd ld/st. Note: in this case the 2nd ld/st exception is masked. This behavior is implementation dependent and may differ on other CPUs.
DVC DACx	DACy	Take Debug exception, DBSR update setting DACx, DACy. DAC_OFST set to 2'b01. DSRR0 points to instruction after 2nd load/store class instruction. Debug counter update occurs for the 2nd ld/st as appropriate. Note: in this case debug counter updates can occur for the 2nd ld/st even though the 1st ld/st has a DVC DAC exception. Note: in this case if x==y, then the resultant state of DBSR and DSRR0 may be indistinguishable from the "no DACy" case.
DVC DACx	DVC DACy	Take Debug exception, DBSR update setting DACx, DACy. DAC_OFST set to 2'b01. DSRR0 points to instruction after 2nd load/store class instruction. Debug counter update occurs for the 2nd ld/st as appropriate. Note: in this case debug counter updates occur for the 2nd ld/st even though the 1st ld/st has a DVC DAC exception. Note: in this case if x==y, then the resultant state of DBSR and DSRR0 may be indistinguishable from the "no DACy" case.

1. No DVC can occur since load/store operation not performed. For the case that an earlier transfer in a 2nd ldst class lmw or stmw sequence successfully caused a DACy or DVC DACy, the DACy or DVC DACy is also ignored, unlike in the case where the lmw or stmw was the 1st load/store classes and an earlier transfer in a lmw or stmw sequence successfully caused a DAC or DVC.

36.10.3 Linked Instruction Address and Data Address Compare Event

Data Address Compare debug events may be 'linked' with an Instruction Address Compare event by setting the DAC1LNK and/or DAC2LNK control bits in DBCR2 to further refine when a Data Address Compare debug event is generated. DAC1 may be linked with IAC1, and DAC2 (when not used as a mask or range bounds register) may be linked with IAC3. When linked, a DAC1 (or DAC2) debug event occurs when the same instruction which generates the DAC1 (or DAC2) 'hit' also generates an IAC1 (or IAC3) 'hit'. When linked, the IAC1 (or IAC3) event is not recorded in the Debug Status register, regardless of whether a corresponding DAC1 (or DAC2) event occurs, or whether the IAC1 (or IAC3) event enable is set.

When enabled and execution of a load or store class instruction results in a data access with an address that meets the criteria specified in the DBCR0, DBCR2, DBCR4, DAC1, DAC2, DVC1, and DVC2 Registers, and the instruction also meets the criteria for generating an

Instruction Address Compare event, a Linked Data Address Compare debug event occurs. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. The normal DAC1 and DAC2 status bits in the DBSR are used for recording these events. The IAC1 and IAC3 status bits are not set if the corresponding Instruction Address Compare register is linked.

Linking is enabled using control bits in DBCR2.

Note: Linked DAC events will not be recorded if a load multiple word or store multiple word instruction is interrupted prior to completion by a critical input or external input interrupt.

36.10.4 Trap Debug Event

A Trap debug event (TRAP) occurs if Trap debug events are enabled ($DBCR0_{TRAP}$=1), a Trap instruction (**tw**) is executed, and the conditions specified by the instruction for the trap are met. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When a Trap debug event occurs, the $DBSR_{TRAP}$ bit is set to '1' to record the debug exception.

36.10.5 Branch Taken Debug Event

A Branch Taken debug event (BRT) occurs if Branch Taken debug events are enabled ($DBCR0_{BRT}$=1) and execution is attempted of a branch instruction which will be taken (either an unconditional branch, or a conditional branch whose branch condition is true), and MSR_{DE}=1 or $DBCR0_{EDM}$=1. Branch Taken debug events are not recognized if MSR_{DE}=0 and $DBCR0_{EDM}$=0 at the time of execution of the branch instruction and thus $DBSR_{IDE}$ can not be set by a Branch Taken debug event. When a Branch Taken debug event is recognized, the $DBSR_{BRT}$ bit is set to '1' to record the debug exception, and the address of the branch instruction will be recorded in DSRR0.

36.10.6 Instruction Complete Debug Event

An Instruction Complete debug event (ICMP) occurs if Instruction Complete debug events are enabled ($DBCR0_{ICMP}$=1), execution of any instruction is completed, and MSR_{DE}=1 or $DBCR0_{EDM}$=1. If execution of an instruction is suppressed due to the instruction causing some other exception which is enabled to generate an interrupt, then the attempted execution of that instruction does not cause an Instruction Complete debug event. The ***sc*** instruction does not fall into the category of an instruction whose execution is suppressed, since the instruction actually executes and then generates a System Call interrupt. In this case, the Instruction Complete debug exception will also be set. When an Instruction Complete debug event is recognized, $DBSR_{ICMP}$ is set to '1' to record the debug exception and the address of the next instruction to be executed will be recorded in DSRR0.

Instruction Complete debug events are not recognized if MSR_{DE}=0 and $DBCR0_{EDM}$=0 at the time of execution of the instruction, thus $DBSR_{IDE}$ is not generally set by an ICMP debug event.

Note: Instruction complete debug events are not generated by the execution of an instruction which sets MSR_{DE} to '1' while $DBCR0_{ICMP}$=1, nor by the execution of an instruction which sets $DBCR0_{ICMP}$ to '1' while MSR_{DE}=1 or $DBCR0_{EDM}$=1.

36.10.7 Interrupt Taken Debug Event

An Interrupt Taken debug event (IRPT) occurs if Interrupt Taken debug events are enabled ($DBCR0_{IRPT}$=1) and a non-critical interrupt occurs. Only non-critical class interrupts cause

an Interrupt Taken debug event. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When an Interrupt Taken debug event occurs, the $DBSR_{IRPT}$ bit is set to '1' to record the debug exception. The value saved in DSRR0 will be the address of the non-critical interrupt handler.

36.10.8 Critical Interrupt Taken Debug Event

A Critical Interrupt Taken debug event (CIRPT) occurs if Critical Interrupt Taken debug events are enabled ($DBCR0_{CIRPT}$=1) and a critical interrupt (other than a Debug interrupt when the Debug APU is disabled) occurs. Only critical class interrupts cause a Critical Interrupt Taken debug event. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When a Critical Interrupt Taken debug event occurs, the $DBSR_{CIRPT}$ bit is set to '1' to record the debug exception. The value saved in DSRR0 will be the address of the critical interrupt handler. Note that this debug event should not normally be enabled unless the Debug APU is also enabled to avoid corruption of CSRR0/1.

36.10.9 Return Debug Event

A Return debug event (RET) occurs if Return debug events are enabled ($DBCR0_{RET}$=1) and an attempt is made to execute an **se_rfi** instruction. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When a Return debug event occurs, the $DBSR_{RET}$ bit is set to '1' to record the debug exception.

If MSR_{DE}=0 and $DBCR0_{EDM}$=0 at the time of the execution of the **se_rfi** (i.e. before the MSR is updated by the **se_rfi**), then $DBSR_{IDE}$ is also set to '1' to record the imprecise debug event.

If MSR_{DE}=1 at the time of the execution of the **se_rfi**, a Debug interrupt will occur provided there exists no higher priority exception which is enabled to cause an interrupt. Debug Save/Restore Register 0 will be set to the address of the **se_rfi** instruction.

36.10.10 Critical Return Debug Event

A Critical Return debug event (CRET) occurs if Critical Return debug events are enabled ($DBCR0_{CRET}$=1) and an attempt is made to execute an **se_rfci** instruction. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When a Critical Return debug event occurs, the $DBSR_{CRET}$ bit is set to '1' to record the debug exception.

If MSR_{DE}=0 and $DBCR0_{EDM}$=0 at the time of the execution of the **se_rfci** (i.e. before the MSR is updated by the **se_rfci**), then $DBSR_{IDE}$ is also set to '1' to record the imprecise debug event.

If MSR_{DE}=1 at the time of the execution of the **se_rfci**, a Debug interrupt will occur provided there exists no higher priority exception which is enabled to cause an interrupt. Debug Save/Restore Register 0 will be set to the address of the **se_rfci** instruction. Note that this debug event should not normally be enabled unless the Debug APU is also enabled to avoid corruption of CSRR0/1.

36.10.11 External Debug Event

An External debug event (DEVT1, DEVT2) occurs if External debug events are enabled ($DBCR0_{DEVT1}$=1 or $DBCR0_{DEVT2}$=1), and the respective **p_devt1** or **p_devt2** input signal transitions to the asserted state. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When an External debug event occurs, $DBSR_{DEVT\{1,2\}}$ is set to '1' to record the debug exception.

36.10.12 Unconditional Debug Event

An Unconditional debug event (UDE) occurs when the Unconditional Debug Event (**p_ude**) input transitions to the asserted state, and either $DBCR0_{IDM}$=1 or $DBCR0_{EDM}$=1. The Unconditional debug event is the only debug event which does not have a corresponding enable bit for the event in DBCR0. This event can occur and be recorded in DBSR regardless of the setting of MSR_{DE}. When an Unconditional debug event occurs, the $DBSR_{UDE}$ bit is set to '1' to record the debug exception.

36.11 Debug Registers

This section describes debug-related registers that are software accessible. These registers are intended for use by special debug tools and debug software, not by general application code.

Access to these registers by software is conditioned by the External Debug Mode control bit ($DBCR0_{EDM}$) and the settings of debug control register DBERC0, which can be set by the hardware debug port. If $DBCR0_{EDM}$ is set and if the bit in DBERC0 corresponding to the resource is cleared, software is prevented from modifying debug register values, since the resource is not "owned" by software. Execution of a **mtspr** instruction targeting a debug register or register field not "owned" by software will not cause modifications to occur. In addition, since the external debugger hardware may be manipulating debug register values, the state of these registers or register fields not "owned" by software is not guaranteed to be consistent if accessed (read) by software with a **mfspr** instruction, other than the $DBCR0_{EDM}$ bit and the DBERC0 register itself. Hardware always has full access to all registers and all register fields through the OnCE register access mechanism, and it is up to the debug firmware to properly implement modifications to these registers with read-modify-write operations to implement any control sharing with software. Settings in DBERC0 should be considered by the debug firmware in order to preserve software settings of control and status registers as appropriate when hardware modifications to the debug registers is performed.

36.11.1 Debug Address and Value Registers

Instruction Address Compare registers IAC1, IAC2, IAC3, and IAC4 are used to hold instruction addresses for address comparison purposes. In addition, IAC2 and IAC4 hold mask information for IAC1 and IAC3 respectively when *Address Bit Match* compare modes are selected. Note that when performing instruction address compares, the low order bit of the instruction address and the corresponding IAC register is ignored.

Data Address Compare registers DAC1 and DAC2 are used to hold data access addresses for address comparison purposes. In addition, DAC2 holds mask information for DAC1 when *Address Bit Match* compare mode is selected.

Data Value Compare registers DVC1 and DVC2 are used to hold data values for data comparison purposes. DVC1 and DVC2 are 32-bit registers. Data value comparisons are used to qualify Data Address compare debug events. DVC1 is associated with DAC1, and DVC2 is associated with DAC2. The most significant byte of the DVC1(2) register (labeled B0 in Figure 506) corresponds to the byte data value transferred to/from memory byte offset 0, and the least significant byte of the register (labeled B3 in Figure 506) corresponds to byte offset 3. When enabled for performing data value comparisons, each enabled byte in DVC1(2) is compared with the memory value transferred on the corresponding active byte lane of the data memory interface to determine if a match occurs. Inactive byte lanes do not

participate in the comparison, they are implicitly masked. Software must also program the DVC1(2) register byte positions based on the endian mode and alignment of the access. Misaligned accesses are not fully supported, since the data address and data value comparisons are only performed on the initial access in the case of a misaligned access; thus, accesses which cross a 32-bit boundary cannot be fully matched. For address and size combinations which involve two transfers, only the initial transfer is used for data address and value matching. DVC1 and DVC2 may be read or written using **mtspr** and **mfspr** instructions.

Figure 506. DVC1, DVC2 Registers

SPR - 318 (DVC1), 319 (DVC2);

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	B0								B1							
Reset	0	0	0	0	0	0	0	0	0	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	B2								B3							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

36.11.2 Debug Control and Status Registers

Debug Control Registers (DBCR0, DBCR1, DBCR2, DBCR4, and DBERC0) are used to enable debug events, reset the processor, and set the debug mode of the processor. The Debug Status register (DBSR) records debug exceptions while Internal or External Debug Mode is enabled.

e200z0h requires that a context synchronizing instruction follow a **mtspr** DBCR0-4 or DBSR to ensure that any alterations enabling/disabling debug events are effective. The context synchronizing instruction may or may not be affected by the alteration. Typically, an **isync** instruction is used to create a synchronization boundary beyond which it can be guaranteed that the newly written control values are in effect.

For watchpoint generation, configuration settings contained in DBCR1and DBCR2 are used, even though the corresponding event(s) may be disabled (via DBCR0) from setting DBSR flags.

Debug Control Register 0 (DBCR0)

Debug Control Register 0 is used to enable debug modes and controls which debug events are allowed to set DBSR flags. e200z0h adds some implementation specific bits to this register, as seen in *Figure 507*

Figure 507. DBCR0 Register

SPR - 308;

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R / W	EDM	IDM	RST		ICMP	BRT	IRPT	TRAP	IAC1	IAC2	IAC3	IAC4	DAC1		DAC2	
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RET	0	0	0	0	DEVT1	DEVT2	0	0	CIRPT	CRET	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. $DBCR0_{EDM}$ is affected by **j_trst_b** or **m_por** assertion, and remains reset while in the Test_Logic_Reset state, but is not affected by **p_reset_b**. All other bits are reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources (other than RST) from reset by **p_reset_b,** and only software-owned resources indicated by DBERC0 and the $DBCR0_{RST}$ field will be reset by **p_reset_b**. The $DBCR0_{RST}$ field will always be reset by **p_reset_b** regardless of the value of $DBCR0_{EDM}$.

Table 459 provides bit definitions for Debug Control Register 0.

Table 459. DBCR0 Bit Definitions

Bit(s)	Name	Description
0	EDM	External Debug Mode. This bit is read-only by software. 0 – External debug mode disabled. Internal debug events not mapped into external debug events. 1 – External debug mode enabled. Hardware-owned events will not cause the CPU to vector to interrupt code. Software is not permitted to write to debug registers {DBCRx, DBSR, DBCNT, IAC1–4, DAC1–2} unless permitted by settings in DBERC0. When external debug mode is enabled, hardware-owned resources in debug registers are not affected by processor reset **p_reset_b**. This allows the debugger to set up hardware debug events which remain active across a processor reset.
		Programming Notes: It is recommended that debug status bits in the Debug Status Register be cleared before disabling external debug mode to avoid any internal imprecise debug interrupts. Software may use this bit to determine if external debug has control over the debug registers. The hardware debugger must set the EDM bit to '1' before other bits in this register (and other debug registers) may be altered. On the initial setting of this bit to '1', all other bits are unchanged. This bit is only writable through the OnCE port.
1	IDM	Internal Debug Mode 0 – Debug exceptions are disabled. Debug events do not affect DBSR unless EDM is set. 1 – Debug exceptions are enabled. Enabled debug events will update the DBSR. If MSR_{DE}=1, the occurrence of a debug event, or the recording of an earlier debug event in the Debug Status Register when MSR_{DE} was cleared, will cause a Debug interrupt.

Table 459. DBCR0 Bit Definitions (continued)

Bit(s)	Name	Description
2:3	RST	Reset Control 00 – No function 01 – Reserved 10 – **p_resetout_b** pin asserted by Debug Reset Control. Allows external device to initiate processor reset. 11 – Reserved
4	ICMP	Instruction Complete Debug Event Enable 0 – ICMP debug events are disabled 1 – ICMP debug events are enabled
5	BRT	Branch Taken Debug Event Enable 0 – BRT debug events are disabled 1 – BRT debug events are enabled
6	IRPT	Interrupt Taken Debug Event Enable 0 – IRPT debug events are disabled 1 – IRPT debug events are enabled
7	TRAP	Trap Taken Debug Event Enable 0 – TRAP debug events are disabled 1 – TRAP debug events are enabled
8	IAC1	Instruction Address Compare 1 Debug Event Enable 0 – IAC1 debug events are disabled 1 – IAC1 debug events are enabled
9	IAC2	Instruction Address Compare 2 Debug Event Enable 0 – IAC2 debug events are disabled 1 – IAC2 debug events are enabled
10	IAC3	Instruction Address Compare 3 Debug Event Enable 0 – IAC3 debug events are disabled 1 – IAC3 debug events are enabled
11	IAC4	Instruction Address Compare 4 Debug Event Enable 0 – IAC4 debug events are disabled 1 – IAC4 debug events are enabled
12:13	DAC1	Data Address Compare 1 Debug Event Enable 00 – DAC1 debug events are disabled 01 – DAC1 debug events are enabled only for store-type data storage accesses 10 – DAC1 debug events are enabled only for load-type data storage accesses 11 – DAC1 debug events are enabled for load-type or store-type data storage accesses
14:15	DAC2	Data Address Compare 2 Debug Event Enable 00 – DAC2 debug events are disabled 01 – DAC2 debug events are enabled only for store-type data storage accesses 10 – DAC2 debug events are enabled only for load-type data storage accesses 11 – DAC2 debug events are enabled for load-type or store-type data storage accesses

Table 459. DBCR0 Bit Definitions (continued)

Bit(s)	Name	Description
16	RET	Return Debug Event Enable 0 – RET debug events are disabled 1 – RET debug events are enabled
17:20	—	Reserved
21	DEVT1	External Debug Event 1 Enable 0 – DEVT1 debug events are disabled 1 – DEVT1 debug events are enabled
22	DEVT2	External Debug Event 2 Enable 0 – DEVT2 debug events are disabled 1 – DEVT2 debug events are enabled
23:24	—	Reserved
25	CIRPT	Critical Interrupt Taken Debug Event Enable 0 – CIRPT debug events are disabled 1 – CIRPT debug events are enabled
26	CRET	Critical Return Debug Event Enable 0 – CRET debug events are disabled 1 – CRET debug events are enabled
27:31	—	Reserved

Debug Control Register 1 (DBCR1)

Debug Control Register 1 is used to configure Instruction Address Compare operation. The DBCR1 register is shown in *Figure 508*

Figure 508. DBCR1 Register

SPR - 309;

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IAC1US		IAC1ER		IAC2US		IAC2ER		IAC12M		0	0	0	0	0	0
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IAC3US		IAC3ER		IAC4US		IAC4ER		IAC34M		0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources from reset by **p_reset_b** and only software-owned resources indicated by DBERC0 will be reset by **p_reset_b**.

Table 460 provides bit definitions for Debug Control Register 1.

Table 460. DBCR1 Bit Definitions

Bit(s)	Name	Description
0:1	IAC1US	Instruction Address Compare 1 User/Supervisor Mode 00 – IAC1 debug events not affected by MSR_{PR} 01 – Reserved 10 – IAC1 debug events can only occur if MSR_{PR}=0 (Supervisor mode) 11 – IAC1 debug events can only occur if MSR_{PR}=1. (User mode)
2:3	IAC1ER	Instruction Address Compare 1 Effective/Real Mode 00 – IAC1 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – IAC1 debug events are based on effective address and can only occur if MSR_{IS}=0 11 – IAC1 debug events are based on effective address and can only occur if MSR_{IS}=1
4:5	IAC2US	Instruction Address Compare 2 User/Supervisor Mode 00 – IAC2 debug events not affected by MSR_{PR} 01 – Reserved 10 – IAC2 debug events can only occur if MSR_{PR}=0 (Supervisor mode) 11 – IAC2 debug events can only occur if MSR_{PR}=1. (User mode)
6:7	IAC2ER	Instruction Address Compare 2 Effective/Real Mode 00 – IAC2 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – IAC2 debug events are based on effective address and can only occur if MSR_{IS}=0 11 – IAC2 debug events are based on effective address and can only occur if MSR_{IS}=1
8:9	IAC12M	Instruction Address Compare 1/2 Mode 00 – Exact address compare. IAC1 debug events can only occur if the address of the instruction fetch is equal to the value specified in IAC1. IAC2 debug events can only occur if the address of the instruction fetch is equal to the value specified in IAC2. 01 – Address bit match. IAC1 debug events can occur only if the address of the instruction fetch, ANDed with the contents of IAC2 are equal to the contents of IAC1, also ANDed with the contents of IAC2. IAC2 debug events do not occur. IAC1US and IAC1ER settings are used. 10 – Inclusive address range compare. IAC1 debug events can occur only if the address of the instruction fetch is greater than or equal to the value specified in IAC1 and less than the value specified in IAC2. IAC2 debug events do not occur. IAC1US and IAC1ER settings are used. 11 – Exclusive address range compare. IAC1 debug events can occur only if the address of the instruction fetch is less than the value specified in IAC1 or is greater than or equal to the value specified in IAC2. IAC2 debug events do not occur. IAC1US and IAC1ER settings are used.
10:15	—	Reserved
16:17	IAC3US	Instruction Address Compare 3 User/Supervisor Mode 00 – IAC3 debug events not affected by MSR_{PR} 01 – Reserved 10 – IAC3 debug events can only occur if MSR_{PR}=0 (Supervisor mode) 11 – IAC3 debug events can only occur if MSR_{PR}=1 (User mode)

Table 460. DBCR1 Bit Definitions (continued)

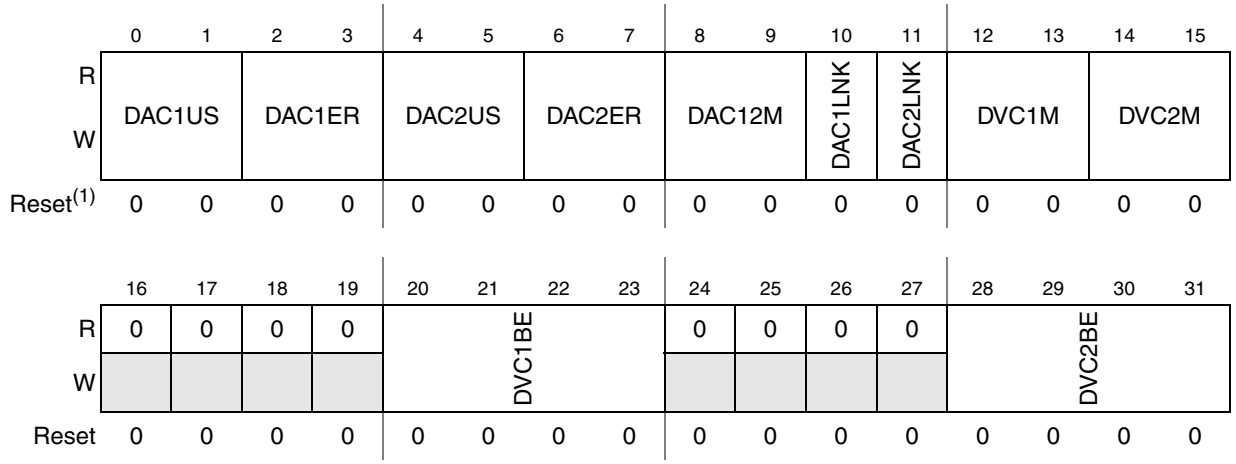
Bit(s)	Name	Description
18:19	IAC3ER	Instruction Address Compare 3 Effective/Real Mode 00 – IAC3 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – IAC3 debug events are based on effective address and can only occur if MSR_{IS}=0 11 – IAC3 debug events are based on effective address and can only occur if MSR_{IS}=1
20:21	IAC4US	Instruction Address Compare 4 User/Supervisor Mode 00 – IAC4 debug events not affected by MSR_{PR} 01 – Reserved 10 – IAC4 debug events can only occur if MSR_{PR}=0 (Supervisor mode). 11 – IAC4 debug events can only occur if MSR_{PR}=1. (User mode)
22:23	IAC4ER	Instruction Address Compare 4 Effective/Real Mode 00 – IAC4 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – IAC4 debug events are based on effective address and can only occur if MSR_{IS}=0 11 – IAC4 debug events are based on effective address and can only occur if MSR_{IS}=1
24:25	IAC34M	Instruction Address Compare 3/4 Mode 00 – Exact address compare. IAC3 debug events can only occur if the address of the instruction fetch is equal to the value specified in IAC3. IAC4 debug events can only occur if the address of the instruction fetch is equal to the value specified in IAC4. 01 – Address bit match. IAC3 debug events can occur only if the address of the instruction fetch, ANDed with the contents of IAC4 are equal to the contents of IAC3, also ANDed with the contents of IAC4. IAC4 debug events do not occur. IAC3US and IAC3ER settings are used. 10 – Inclusive address range compare. IAC3 debug events can occur only if the address of the instruction fetch is greater than or equal to the value specified in IAC3 and less than the value specified in IAC4. IAC4 debug events do not occur. IAC3US and IAC3ER settings are used. 11 – Exclusive address range compare. IAC3 debug events can occur only if the address of the instruction fetch is less than the value specified in IAC3 or is greater than or equal to the value specified in IAC4. IAC4 debug events do not occur. IAC3US and IAC3ER settings are used.
26:31	—	Reserved

Debug Control Register 2 (DBCR2)

Debug Control Register 2 is used to configure Data Address Compare and Data Value Compare operation. The DBCR2 register is shown in *Figure 509*..

Figure 509. DBCR2 Register

SPR - 310



1. Reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources from reset by **p_reset_b** and only software-owned resources indicated by DBERC0 will be reset by **p_reset_b**.

Table 461 provides bit definitions for Debug Control Register 2.

Table 461. DBCR2 Bit Definitions

Bit(s)	Name	Description
0:1	DAC1US	Data Address Compare 1 User/Supervisor Mode 00 – DAC1 debug events not affected by MSR_{PR} 01 – Reserved 10 – DAC1 debug events can only occur if MSR_{PR}=0 (Supervisor mode) 11 – DAC1 debug events can only occur if MSR_{PR}=1. (User mode)
2:3	DAC1ER	Data Address Compare 1 Effective/Real Mode 00 – DAC1 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – DAC1 debug events are based on effective address and can only occur if MSR_{DS}=0 11 – DAC1 debug events are based on effective address and can only occur if MSR_{DS}=1
4:5	DAC2US	Data Address Compare 2 User/Supervisor Mode. 00 – DAC2 debug events not affected by MSR_{PR} 01 – Reserved 10 – DAC2 debug events can only occur if MSR_{PR}=0 (Supervisor mode) 11 – DAC2 debug events can only occur if MSR_{PR}=1. (User mode)
6:7	DAC2ER	Data Address Compare 2 Effective/Real Mode 00 – DAC2 debug events are based on effective address 01 – Unimplemented in e200z0h (Book E real address compare), no match can occur 10 – DAC2 debug events are based on effective address and can only occur if MSR_{DS}=0 11 – DAC2 debug events are based on effective address and can only occur if MSR_{DS}=1

Table 461. DBCR2 Bit Definitions (continued)

Bit(s)	Name	Description
8:9	DAC12M	Data Address Compare 1/2 Mode 00 – Exact address compare. DAC1 debug events can only occur if the address of the data access is equal to the value specified in DAC1. DAC2 debug events can only occur if the address of the data access is equal to the value specified in DAC2. 01 – Address bit match. DAC1 debug events can occur only if the address of the data access ANDed with the contents of DAC2, are equal to the contents of DAC1 also ANDed with the contents of DAC2. DAC2 debug events do not occur. DAC1US and DAC1ER settings are used. 10 – Inclusive address range compare. DAC1 debug events can occur only if the address of the data access is greater than or equal to the value specified in DAC1 and less than the value specified in DAC2. DAC2 debug events do not occur. DAC1US and DAC1ER settings are used. 11 – Exclusive address range compare. DAC1 debug events can occur only if the address of the data access is less than the value specified in DAC1 or is greater than or equal to the value specified in DAC2. DAC2 debug events do not occur. DAC1US and DAC1ER settings are used.
10	DAC1LNK	Data Address Compare 1 Linked 0 – no affect 1 – DAC1 debug events are linked to IAC1 debug events. IAC1 debug events do not affect DBSR When linked to IAC1, DAC1 debug events are conditioned based on whether the instruction also generated an IAC1 debug event
11	DAC2LNK	Data Address Compare 2 Linked 0 – no affect 1 – DAC 2 debug events are linked to IAC3 debug events. IAC3 debug events do not affect DBSR When linked to IAC3, DAC2 debug events are conditioned based on whether the instruction also generated an IAC3 debug event. DAC2 can only be linked if DAC12M specifies *Exact Address Compare* since DAC2 debug events are not generated in the other compare modes.

Table 461. DBCR2 Bit Definitions (continued)

Bit(s)	Name	Description
12:13	DVC1M	Data Value Compare 1 Mode When $DBCR4_{DVC1C}$=0: 00 – DAC1 debug events not affected by data value compares. 01 – DAC1 debug events can only occur when all bytes specified in the DVC1BE field match the corresponding data byte values for active byte lanes of the memory access. 10 – DAC1 debug events can only occur when any byte specified in the DVC1BE field matches the corresponding data byte value for active byte lanes of the memory access. 11 – DAC1 debug events can only occur when all bytes specified in the DVC1BE field within at least one of the halfwords of the data value of the memory access matches the corresponding DVC1 value. Note: Inactive byte lanes of the memory access are automatically masked. When $DBCR4_{DVC1C}$=1: 00 – Reserved 01 – DAC1 debug events can only occur when any byte specified in the DVC1BE field does not match the corresponding data byte value for active byte lanes of the memory access. If all active bytes match, then no event will be generated. 10 – DAC1 debug events can only occur when all bytes specified in the DVC1BE field do not match the corresponding data byte values for active byte lanes of the memory access. If any active byte match occurs, no event will be generated. 11 – Reserved Note: Inactive byte lanes of the memory access are automatically masked.

Table 461. DBCR2 Bit Definitions (continued)

Bit(s)	Name	Description
14:15	DVC2M	Data Value Compare 2 Mode When $DBCR4_{DVC2C}$=0: 00 – DAC2 debug events not affected by data value compares. 01 – DAC2 debug events can only occur when all bytes specified in the DVC2BE field match the corresponding data byte values for active byte lanes of the memory access. 10 – DAC2 debug events can only occur when any byte specified in the DVC2BE field matches the corresponding data byte value for active byte lanes of the memory access. 11 – DAC2 debug events can only occur when all bytes specified in the DVC2BE field within at least one of the halfwords of the data value of the memory access matches the corresponding DVC2 value. Note: Inactive byte lanes of the memory access are automatically masked. When $DBCR4_{DVC2C}$=1: 00 – Reserved 01 – DAC2 debug events can only occur when any byte specified in the DVC2BE field does not match the corresponding data byte value for active byte lanes of the memory access. If all active bytes match, then no event will be generated. 10 – DAC2 debug events can only occur when all bytes specified in the DVC2BE field do not match the corresponding data byte values for active byte lanes of the memory access. If any active byte match occurs, no event will be generated. 11 – Reserved Note: Inactive byte lanes of the memory access are automatically masked.
16:19	—	Reserved
20:23	DVC1BE	Data Value Compare 1 Byte Enables Specifies which bytes in the aligned doubleword value associated with the memory access are compared to the corresponding bytes in DVC1. Inactive byte lanes of a memory access smaller than 32-bits are automatically masked by hardware. If all bits in the DVC1BE field are clear, then a match will occur regardless of the data. 1xxx – Byte lane 0 is enabled for comparison with the value in bits 0:7 of DVC1. x1xx – Byte lane 1 is enabled for comparison with the value in bits 8:15 of DVC1. xx1x – Byte lane 2 is enabled for comparison with the value in bits 16:23 of DVC1. xxx1 – Byte lane 3 is enabled for comparison with the value in bits 24:31 of DVC1.
24:27	—	Reserved
28:31	DVC2BE	Data Value Compare 2 Byte Enables Specifies which bytes in the aligned doubleword value associated with the memory access are compared to the corresponding bytes in DVC2. Inactive byte lanes of a memory access smaller than 32-bits are automatically masked by hardware. If all bits in the DVC2BE field are clear, then a match will occur regardless of the data. 1xxx – Byte lane 0 is enabled for comparison with the value in bits 0:7 of DVC2. x1xx – Byte lane 1 is enabled for comparison with the value in bits 8:15 of DVC2. xx1x – Byte lane 2 is enabled for comparison with the value in bits 16:23 of DVC2. xxx1 – Byte lane 3 is enabled for comparison with the value in bits 24:31 of DVC2.

Debug Control Register 4 (DBCR4)

Debug Control Register 4 is used to extend data value compare matching functionality. DBCR4 is shown in *Figure 510*.

Figure 510. DBCR4 Register

SPR - 563

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	DVC1C	0	DVC2C	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset(1)	0	0	0	0	0	0	0	0	0	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

1. DBCR4 is reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources from reset by **p_reset_b** and only software-owned resources indicated by DBERC0 will be reset by **p_reset_b**.

Table 462 provides bit definitions for Debug Control Register 4.

Table 462. DBCR4 Bit Definitions

Bit(s)	Name	Description
17:20	—	Reserved
0	DVC1C	Data Value Compare 1 Control 0 – Normal DVC1 operation. 1 – Inverted polarity DVC1 operation DVC1C controls whether DVC1 data value comparisons utilize the normal BookE operation, or an alternate “inverted compare” operation. In inverted polarity mode, data value compares perform a not-equal comparison. See details in the DBCR2 register definition
2	—	Reserved
3	DVC2C	Data Value Compare 2 Control 0 – Normal DVC2 operation. 1 – Inverted polarity DVC2 operation DVC2C controls whether DVC2 data value comparisons utilize the normal BookE operation, or an alternate “inverted compare” operation. In inverted polarity mode, data value compares perform a not-equal comparison. See details in the DBCR2 register definition
4:31	—	Reserved

Debug Status Register (DBSR)

The Debug Status Register (DBSR) contains status on debug events and the most recent processor reset. The Debug Status Register is set via hardware, and read and cleared via software. Bits in the Debug Status Register can be cleared using **mtspr** *DBSR,RS*. Clearing is done by writing to the Debug Status Register with a 1 in any bit position that is to be cleared and 0 in all other bit positions. The write data to the Debug Status Register is not direct data, but a mask. A '1' causes the bit to be cleared, and a '0' has no effect. Debug Status bits are set by Debug events only while Internal Debug Mode is enabled or External Debug Mode is enabled. When debug interrupts are enabled (MSR_{DE}=1 $DBCR0_{IDM}$=1 and $DBCR0_{EDM}$=0, or MSR_{DE}=1, $DBCR0_{IDM}$=1 and $DBCR0_{EDM}$=1 and software is allocated resource(s) via DBERC0), a set bit in DBSR owned by software other than MRR or VLES will cause a debug interrupt to be generated. The debug interrupt handler is responsible for clearing DBSR bits prior to returning to normal execution. The Power Architecture technology VLE APU adds the $DBSR_{VLES}$ status bit to indicate debug events occurring due to a Power Architecture technology VLE instruction. When resource sharing is enabled, ($DBCR0_{EDM}$=1 and $DBERC0_{IDM}$=1), only software-owned resources may be modified by software, and all status bits associated with hardware-owned resources will be forced to '0' in DBSR when read by software via a **mfspr** instruction. Hardware always has full access to all registers and all register fields in DBSR through the OnCE register access mechanism.

The DBSR register is shown in *Figure 511*.

Figure 511. DBSR Register

SPR - 304

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IDE	UDE	MRR		ICMP	BRT	IRPT	TRAP	IAC1	IAC2	IAC3	IAC4	DAC1R	DAC1W	DAC2R	DAC2W
W																
Reset(1)	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	RET	0	0	0	0	DEVT1	DEVT2	0	0	CIRPT	CRET	VLES	0	DAC_OFST		0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources from reset by **p_reset_b** and only software-owned resources indicated by DBERC0 will be reset by **p_reset_b.** $DBSR_{MRR}$ is always updated by **p_reset_b** however.

Table 463 provides bit definitions for the Debug Status Register.

Table 463. DBSR Bit Definitions

Bit(s)	Name	Description
0	IDE	Imprecise Debug Event Set to '1' if MSR_{DE}=0, $DBCR0_{IDM}$=1 and a debug event causes its respective Debug Status Register bit to be set to '1'. If $DBERC0_{IDM}$=0, it may also be set to '1' if $DBCR0_{EDM}$=1 and an imprecise debug event occurs due to a DAC event on a load or store which is terminated with errorIf $DBCR0_{EDM}$=1 and $DBERC0_{IDM}$=1, this bit is "owned" by software. The hardware debugger will not be informed directly of an imprecise event.
1	UDE	Unconditional Debug Event Set to '1' if an Unconditional debug event occurred.
2:3	MRR	Most Recent Reset. 00 – No reset occurred since these bits were last cleared by software 01 – A hard reset occurred since these bits were last cleared by software 10 – Reserved 11 – Reserved
4	ICMP	Instruction Complete Debug Event Set to '1' if an Instruction Complete debug event occurred.
5	BRT	Branch Taken Debug Event Set to '1' if an Branch Taken debug event occurred.
6	IRPT	Interrupt Taken Debug Event Set to '1' if an Interrupt Taken debug event occurred.
7	TRAP	Trap Taken Debug Event Set to '1' if a Trap Taken debug event occurred.
8	IAC1	Instruction Address Compare 1 Debug Event Set to '1' if an IAC1 debug event occurred.
9	IAC2	Instruction Address Compare 2 Debug Event Set to '1' if an IAC2 debug event occurred.
10	IAC3	Instruction Address Compare 3 Debug Event Set to '1' if an IAC3 debug event occurred.
11	IAC4	Instruction Address Compare 4 Debug Event Set to '1' if an IAC4 debug event occurred.
12	DAC1R	Data Address Compare 1 Read Debug Event Set to '1' if a read-type DAC1 debug event occurred while $DBCR0_{DAC1}$=0b10 or $DBCR0_{DAC1}$=0b11
13	DAC1W	Data Address Compare 1 Write Debug Event Set to '1' if a write-type DAC1 debug event occurred while $DBCR0_{DAC1}$=0b01 or $DBCR0_{DAC1}$=0b11
14	DAC2R	Data Address Compare 2 Read Debug Event Set to '1' if a read-type DAC2 debug event occurred while $DBCR0_{DAC2}$=0b10 or $DBCR0_{DAC2}$=0b11

Table 463. DBSR Bit Definitions (continued)

Bit(s)	Name	Description
15	DAC2W	Data Address Compare 2 Write Debug Event Set to '1' if a write-type DAC2 debug event occurred while $DBCR0_{DAC2}$=0b01 or $DBCR0_{DAC2}$=0b11
16	RET	Return Debug Event Set to '1' if a Return debug event occurred
17:20	—	Reserved
21	DEVT1	External Debug Event 1 Debug Event Set to '1' if a DEVT1 debug event occurred
22	DEVT2	External Debug Event 2 Debug Event Set to '1' if a DEVT2 debug event occurred
23:24	—	Reserved
25	CIRPT	Critical Interrupt Taken Debug Event Set to '1' if a Critical Interrupt Taken debug event occurred.
26	CRET	Critical Return Debug Event Set to '1' if a Critical Return debug event occurred
27	VLES	VLE Status Set to '1' if an ICMP, BRT, TRAP, RET, CRET, IAC, or DAC debug event occurred on a Power Architecture technology VLE Instruction. Undefined for IRPT, CIRPT, DEVT[1,2], and UDE events
28	—	Reserved
29:30	DAC_OFST	Data Address Compare Offset Indicates offset-1 of saved DSRR0 value from the address of the load or store instruction which took a DAC Debug exception, unless a simultaneous DSI error occurs, in which case this field is set to 2'b00 and $DBSR_{IDE}$ is set to '1'. Normally set to 2'b00 by a non-DVC DAC. A DVC DAC may set this field to any value.
31	—	Reserved

36.11.3 Debug External Resource Control Register (DBERC0)

The Debug External Resource Control Register (DBERC0) controls resource allocation when $DBCR0_{EDM}$ is set to '1'. DBERC0 provides a mechanism for the hardware debugger to share certain debug resources with software. Individual resources are allocated based on the settings of DBERC0 when $DBCR0_{EDM}$=1. DBERC0 settings are ignored when $DBCR0_{EDM}$=0.

Hardware-owned resources which generate debug events update DBSR and cause entry into debug mode, while software-owned resources which generate debug events if $DBCR0_{IDM}$=1, update DBSR and act as if they occurred in internal debug mode, thus causing debug interrupts to occur if MSR_{DE}=1. DBERC0 is controlled via the OnCE port hardware, and is read-only to software.

Debug status bits in DBSR are set by software-owned debug events only while Internal Debug Mode is enabled. When debug interrupts are enabled (MSR_{DE}=1 $DBCR0_{IDM}$=1 and $DBCR0_{EDM}$=0, or MSR_{DE}=1, $DBCR0_{IDM}$=1 and $DBCR0_{EDM}$=1 and software is allocated

resource(s) via DBERC0), a set bit in DBSR which is software-owned other than MRR or VLES will cause a debug interrupt to be generated.

Debug status bits in DBSR are set by hardware-owned debug events only while External Debug Mode is enabled ($DBCR0_{EDM}$=1).

If $DBERC0_{IDM}$=1, all DBSR status bits corresponding to hardware-owned debug events are masked to '0' when accessed by software. The actual values in the DBSR register is always visible to hardware when accessed via the OnCE port.

Software-owned resources may be modified by software, but only the corresponding control and status bits in DBCR0-4 and DBSR are affected by execution of a **mtspr**, thus only a portion of these registers may be affected, depending on the allocation settings in DBERC0. The debug interrupt handler is still responsible for clearing software-owned DBSR bits prior to returning to normal execution. Hardware always has full access to all registers and register fields through the OnCE register access mechanism, and it is up to the debug firmware to properly implement modifications to these registers with read-modify-write operations to implement any control sharing with software. Settings in DBERC0 should be considered by the debug firmware in order to preserve software settings of control and status registers as appropriate when hardware modifications to the debug registers is performed.

The DBERC0 register is shown in *Figure 512*.

Figure 512. DBERC0 Register(1)

SPR - 569

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IDM	RST	UDE	ICMP	BRT	IRPT	TRAP	IAC1	IAC2	IAC3	IAC4	DAC1	0	DAC2	0
W																
Reset(2)	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	RET	0	0	0	0	DEVT1	DEVT2	0	0	CIRPT	CRET	BKPT	0	0	0	FT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Read-only by Software; Reset - Unaffected by **p_reset_b**, cleared by **m_por** or while in the test-logic-reset OnCE controller state.
2. Reset by processor reset **p_reset_b** if $DBCR0_{EDM}$=0, as well as unconditionally by **m_por**. If $DBCR0_{EDM}$=1, DBERC0 masks off hardware-owned resources from reset by **p_reset_b** and only software-owned resources indicated by DBERC0 will be reset by **p_reset_b.** $DBSR_{MRR}$ is always updated by **p_reset_b** however.

Table 463 provides bit definitions for the Debug External Resource Control Register. Note that DBERC0 controls are disabled when $DBCR0_{EDM}$=0.

Table 464. DBERC0 Bit Definitions

Bit(s)	Name	Description
0	—	Reserved
1	IDM	Internal Debug Mode control 0 – Internal Debug mode may not be enabled by software. $DBCR0_{IDM}$ is owned exclusively by hardware. **mtspr** DBCR0–4, and DBSR is always ignored. No resource sharing occurs, regardless of the settings of other fields in DBERC0. Hardware exclusively owns all resources. 1 – Internal Debug mode may be enabled by software. $DBCR0_{IDM}$ owned by software. $DBCR0_{IDM}$ software readable/writeable. When $DBERC0_{IDM}$=1, writes to hardware-owned bits in DBCR0–4, and DBSR via **mtspr** are ignored.
2	RST	Reset Field Control 0 – $DBCR0_{RST}$ owned exclusively by hardware debug. No **mtspr** access by software to $DBCR0_{RST}$ field. 1 – $DBCR0_{RST}$ accessible by software debug. $DBCR0_{RST}$ is software readable/writeable.
3	UDE	Unconditional Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBSR_{UDE}$ field. 1 – Event owned by software debug. $DBSR_{UDE}$ is software readable/writeable.
4	ICMP	Instruction Complete Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{ICMP}$ or $DBSR_{ICMP}$ fields. 1 – Event owned by software debug. $DBCR0_{ICMP}$ and $DBSR_{ICMP}$ are software readable/writeable.
5	BRT	Branch Taken Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{BRT}$ or $DBSR_{BRT}$ fields. 1 – Event owned by software debug. $DBCR0_{BRT}$ and $DBSR_{BRT}$ are software readable/writeable.
6	IRPT	Interrupt Taken Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{IRPT}$ or $DBSR_{IRPT}$ fields. 1 – Event owned by software debug. $DBCR0_{IRPT}$ and $DBSR_{IRPT}$ are software readable/writeable.
7	TRAP	Trap Taken Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{TRAP}$ or $DBSR_{TRAP}$ fields. 1 – Event owned by software debug. $DBCR0_{TRAP}$ and $DBSR_{TRAP}$ are software readable/writeable.
8	IAC1	Instruction Address Compare 1 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to IAC1 control and status fields. 1 – Event owned by software debug. $DBCR0_{IAC1}$ and $DBSR_{IAC1}$ are software readable/writeable.

Table 464. DBERC0 Bit Definitions (continued)

Bit(s)	Name	Description
9	IAC2	Instruction Address Compare 2 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to IAC2 control and status fields. 1 – Event owned by software debug. IAC2 control and status fields are software readable/writeable.
10	IAC3	Instruction Address Compare 3 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to IAC3 control and status fields. 1 – Event owned by software debug. IAC3 control and status fields are software readable/writeable.
11	IAC4	Instruction Address Compare 4 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to IAC4 control and status fields. 1 – Event owned by software debug. IAC4 control and status fields are software readable/writeable.
12	DAC1	Data Address Compare 1 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to DAC1 control and status fields. 1 – Event owned by software debug. DAC1 control and status fields are software readable/writeable.
13	—	Reserved
14	DAC2	Data Address Compare 2 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to DAC2 control and status fields. 1 – Event owned by software debug. DAC2 control and status fields are software readable/writeable.
15	—	Reserved
16	RET	Return Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{RET}$ or $DBSR_{RET}$ fields. 1 – Event owned by software debug. $DBCR0_{RET}$ and $DBSR_{RET}$ are oftware readable/writeable.
17:20	—	Reserved
21	DEVT1	External Debug Event 1 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{DEVT1}$ or $DBSR_{DEVT1}$ fields. 1 – Event owned by software debug. $DBCR0_{DEVT1}$ and $DBSR_{DEVT1}$ are software readable/writeable.
22	DEVT2	External Debug Event 2 Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{DEVT2}$ or $DBSR_{DEVT2}$ fields. 1 – Event owned by software debug. $DBCR0_{DEVT2}$ and $DBSR_{DEVT2}$ are software readable/writeable.
23:24	—	Reserved

Table 464. DBERC0 Bit Definitions (continued)

Bit(s)	Name	Description
25	CIRPT	Critical Interrupt Taken Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{CIRPT}$ or $DBSR_{CIRPT}$ fields. 1 – Event owned by software debug. $DBCR0_{CIRPT}$ and $DBSR_{CIRPT}$ are software readable/writeable.
26	CRET	Critical Return Debug Event 0 – Event owned by hardware debug. No **mtspr** access by software to $DBCR0_{CRET}$ or $DBSR_{CRET}$ fields. 1 – Event owned by software debug. $DBCR0_{CRET}$ and $DBSR_{CRET}$ are software readable/writeable.
27	BKPT	Breakpoint Instruction Debug Control 0 – Breakpoint owned by hardware debug. Execution of a bkpt instruction (all 0's opcode) results in entry into debug mode. 1 – Breakpoint owned by software debug. Execution of a bkpt instruction (all 0's opcode) results in illegal instruction exception.
28:30	—	Reserved
31	FT	Freeze Timer Debug Control 0 – $DBCR0_{FT}$ owned by hardware debug. No access by software. 1 – $DBCR0_{FT}$ owned by software debug. $DBCR0_{FT}$ is software readable/writeable.

Table 465 shows which resources are controlled by DBERC0 settings.

Table 465. DBERC0 Resource Control

$DBCR0_{EDM}$	$DBERC0_{IDM}$	$DBERC0_{RST}$	$DBERC0_{UDE}$	$DBERC0_{ICMP}$	$DBERC0_{BRT}$	$DBERC0_{IRPT}$	$DBERC0_{TRAP}$	$DBERC0_{IAC1}$	$DBERC0_{IAC2}$	$DBERC0_{IAC3}$	$DBERC0_{IAC4}$	$DBERC0_{DAC1}$	$DBERC0_{DAC2}$	$DBERC0_{RET}$	$DBERC0_{DEVT1}$	$DBERC0_{DEVT2}$			$DBERC0_{CIRPT}$	$DBERC0_{CRET}$	$DBERC0_{BKPT}$	$DBERC0_{DQM}$	$DBERC0_{FT}$	Software Accessible via mtspr, affected by p_reset_b
0	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	All debug registers
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBSR_{MRR}$ (1)
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{IDM}$, $DBSR_{IDE, VLES}$
1	1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{RST}$
1	1	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{UDE}$
1	1	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{ICMP}$, $DBSR_{ICMP}$
1	1	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{BRT}$, $DBSR_{BRT}$
1	1	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{IRPT}$, $DBSR_{IRPT}$
1	1	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR0_{TRAP}$, $DBSR_{TRAP}$
1	1	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	IAC1, $DBCR0_{IAC1}$, $DBCR1_{IAC1US, IAC1ER}$, $DBSR_{IAC1}$
1	1	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	IAC2, $DBCR0_{IAC2}$, $DBCR1_{IAC2US, IAC2ER}$, $DBSR_{IAC2}$

Table 465. DBERC0 Resource Control (continued)

$DBCR0_{EDM}$	$DBERC0_{IDM}$	$DBERC0_{RST}$	$DBERC0_{UDE}$	$DBERC0_{ICMP}$	$DBERC0_{BRT}$	$DBERC0_{IRPT}$	$DBERC0_{TRAP}$	$DBERC0_{IAC1}$	$DBERC0_{IAC2}$	$DBERC0_{IAC3}$	$DBERC0_{IAC4}$	$DBERC0_{DAC1}$	$DBERC0_{DAC2}$	$DBERC0_{RET}$	$DBERC0_{DEVT1}$	$DBERC0_{DEVT2}$			$DBERC0_{CIRPT}$	$DBERC0_{CRET}$	$DBERC0_{BKPT}$	$DBERC0_{DQM}$	$DBERC0_{FT}$	Software Accessible via mtspr, affected by p_reset_b
1	1	—	—	—	—	—	—	1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR1_{IAC12M}$
1	1	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	—	IAC3, $DBCR0_{IAC3}$, $DBCR1_{IAC3US, IAC3ER}$, $DBSR_{IAC3}$
1	1	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	—	IAC4, $DBCR0_{IAC4}$, $DBCR1_{IAC4US, IAC4ER}$, $DBSR_{IAC4}$
1	1	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—	—	—	—	—	—	—	$DBCR1_{IAC34M}$
1	1	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	DAC1, DVC1 $DBCR0_{DAC1}$, $DBCR2_{DAC1US, DAC1ER, DVC1M, DVC1BE}$ $DBCR4_{DVC1C}$ $DBSR_{DAC1, DAC\_OFST}$
1	1	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	—	DAC2, DVC2 $DBCR0_{DAC2}$, $DBCR2_{DAC2US, DAC2ER, DVC2M, DVC2BE}$ $DBCR4_{DVC2C}$ $DBSR_{DAC2, DAC\_OFST}$
1	1	—	—	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—	—	—	—	—	$DBCR2_{DAC12M}$
1	1	—	—	—	—	—	—	1	—	—	—	1	—	—	—	—	—	—	—	—	—	—	—	$DBCR2_{DAC1LNK}$
1	1	—	—	—	—	—	—	—	—	1	—	—	1	—	—	—	—	—	—	—	—	—	—	$DBCR2_{DAC2LNK}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	—	$DBCR0_{RET}$, $DBSR_{RET}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	—	$DBCR0_{DEVT1}$, $DBSR_{DEVT1}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	—	—	—	$DBCR0_{DEVT2}$, $DBSR_{DEVT2}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	—	$DBCR0_{CIRPT}$, $DBSR_{CIRPT}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	—	$DBCR0_{CRET}$, $DBSR_{CRET}$
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	—	—	
1	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	$DBCR0_{FT}$

1. $DBSR_{MRR}$ is always updated by **p_reset_b**, regardless of the value of $DBCR0_{EDM}$ or $DBERC0_{IDM}$

DBERC0 also controls which bits or fields in DBCR0–4 are reset by assertion of **p_reset_b** when $DBCR0_{EDM}$=1. Only software-owned bits or fields as shown in Table 465 are affected in this case, except that $DBCR0_{RST}$ and $DBSR_{MRR}$ are updated by assertion of **p_reset_b** regardless of the value of $DBCR0_{EDM}$ or DBERC0.

36.12 External Debug Support

External debug support is supplied through the e200z0h OnCE controller serial interface which allows access to internal CPU registers and other system state while the CPU is halted in debug mode. All debug resources including DBCR0–4, DBSR, IAC1–4, DVC1–2, DAC1–2 are accessible through the serial OnCE interface in external debug mode. Setting the $DBCR0_{EDM}$ bit to '1' through the OnCE interface enables external debug mode, and unless otherwise permitted by the settings in DBERC0, disables software updates to the debug registers. When $DBCR0_{EDM}$ is set, debug events enabled to set respective DBSR status bits will also cause the CPU to enter Debug Mode as opposed to generating Debug Interrupts, unless the specific events are allocated to software via the settings in DBERC0. In Debug Mode, the CPU is halted at a recoverable boundary, and an external Debug Control Module may control CPU operation through the On-Chip Emulation logic (OnCE).

Note: On the initial setting of $DBCR0_{EDM}$ to '1', other bits in DBCR0 will remain unchanged. After $DBCR0_{EDM}$ has been set, all debug register resources may be subsequently controlled through the OnCE interface. The DBSR register should be cleared as part of the process of enabling external debug activity. The CPU should be placed into debug mode via the OCR_{DR} control bit prior to writing EDM to '1'. This gives the debugger the opportunity to cleanly write to the DBCRx registers and the DBSR to clear out any residual state / control information which could cause unintended operation.

*Note: It is intended for the CPU to remain in external debug mode ($DBCR0_{EDM}$=1) in order to single step or perform other debug mode entry/ reentry via the OCR_{DR}, by performing go+noexit commands, or by assertion of the **jd_de_b** signal.*

*Note: $DBCR0_{EDM}$ operation will be blocked if OnCE operation is disabled (**jd_en_once** negated) regardless of whether it is set or cleared. This means that if $DBCR0_{EDM}$ was previously set, and then **jd_en_once** is negated (this should not occur), entry into debug mode will be blocked, all events are blocked, and watchpoints are blocked.*

Due to clock domain design, the CPU clock (**m_clk**) must be active in order to perform writes to debug registers other than the OnCE Command register (OCMD), the OnCE Control register (OCR), or the $DBCR0_{EDM}$ bit. Register read data is synchronized back to the **j_tclk** clock domain. The OnCE Control register provides the capability of signaling the system level clock controller that the CPU clock should be activated if not already active.

Updates to the DBCRx and DBSR registers via the OnCE interface should be performed with the CPU in debug mode to guarantee proper operation. Due to the various points in the CPU pipeline where control is sampled and event handshaking is performed, it is possible that modifications to these registers while the CPU is running may result in early or late entry into debug mode, and may have incorrect status posted in the DBSR register.

If resource sharing is enabled via DBERC0, updates to the DBERC0, DBCRx, and DBSR registers must be performed with the CPU in debug mode, since simultaneous updates of register portions could otherwise be attempted, and such updates are not guaranteed to properly occur. The results of such an attempt are undefined.

36.12.1 OnCE Introduction

The e200z0h on-chip emulation circuitry (OnCE™/Nexus Class 1 interface) provides a means of interacting with the e200z0h core and integrated system so that a user may examine registers, memory, or on-chip peripherals facilitating hardware/software development. OnCE operation is controlled via an industry standard IEEE 1149.1 TAP

controller. By using public instructions, the external hardware debugger can freeze or halt the CPU, read and write internal state, and resume normal execution. The core does not contain IEEE 1149.1 standard boundary cells on its interface, as it is a building block for further integration. It does not support the JTAG related boundary scan instruction functionality, although JTAG public instructions may be decoded and signaled to external logic.

The OnCE logic provides for Nexus Class 1 static debug capability (utilizing the same set of resources available to software while in internal debug mode), and is present in all e200z0h-based designs. The OnCE module also provides support for directly integrating a Nexus class 2 or class 3 Real-Time Debug unit with the e200z0h core for development of real-time systems where traditional static debug is insufficient. The partitioning between a OnCE module and a connected Nexus module to provide real-time debug allows for capability and cost trade-offs to be made.

The e200z0h core is designed to be a fully integratable module. The OnCE TAP controller and associated enabling logic are designed to allow concatenation with an existing JTAG controller if present in the system. Thus, the e200z0h module can be easily integrated with existing JTAG designs or as a stand-alone controller.

In order to enable full OnCE operation, the **jd_enable_once** input signal must be asserted. In some system integrations, this is automatic, since the input will be tied asserted. Other integrations may require the execution of the Enable OnCE command via the TAP and appropriate entry of serial data. Exact requirements will be documented by the integrated product specification. The **jd_enable_once** input signal should not change state during a debug session, or undefined activity may occur.

The following figures show the TAP controller state model and the TAP registers implemented by the OnCE logic.

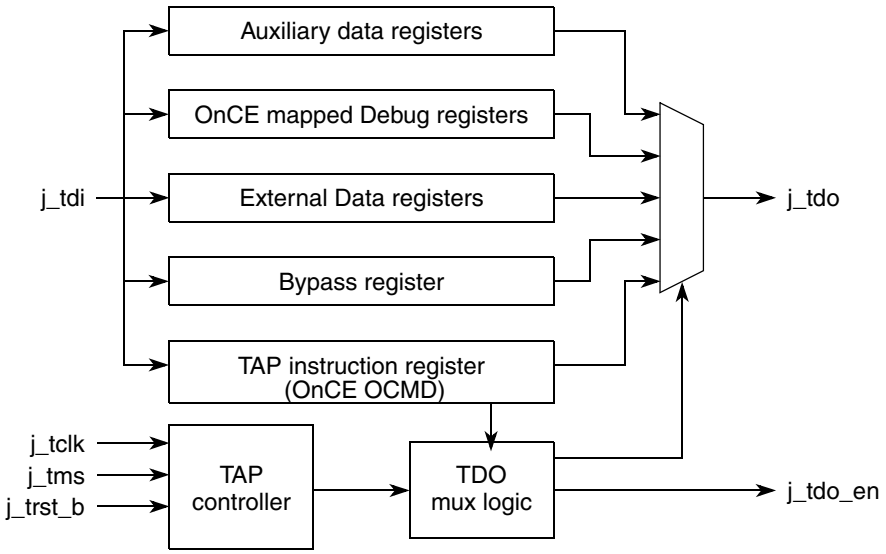
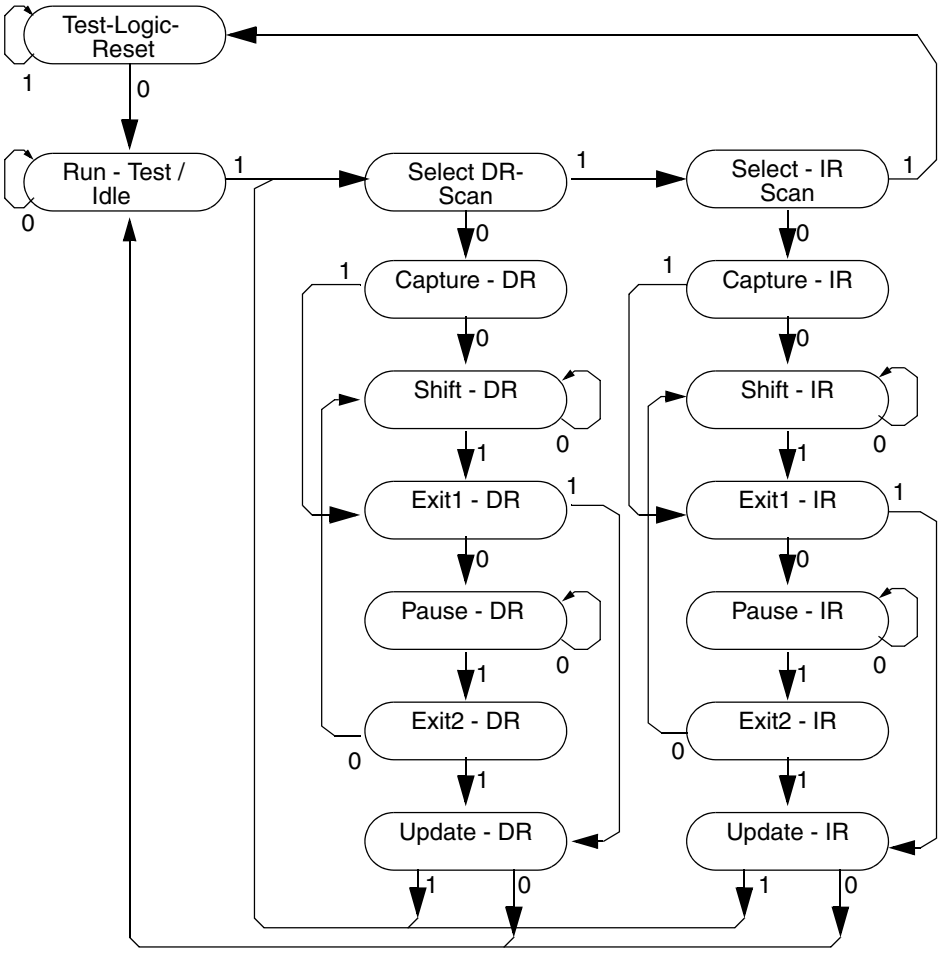


Figure 513. OnCE TAP Controller and Registers

The OnCE controller is implemented as a 16-state FSM (finite state machine), with a one-to-one correspondence to the states defined for the JTAG TAP controller.



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 514. IEEE 1149.1-2001 TAP Controller State Machine

Access to processor registers and the contents of memory locations are performed by enabling external debug mode (setting $DBCR0_{EDM}$ to '1'), placing the processor into debug mode, followed by scanning instructions and data into and out of the CPU Scan Chain (CPUSCR); execution of scanned instructions by the CPU is used as the method to access required data. Memory locations may be read by scanning a load instruction into the e200z0h core which will reference the desired memory location, executing the load instruction, and then scanning out the result of the load. Other resources are accessed in a similar manner.

The initial entry by the CPU into the debug state (or mode) from normal, waiting, stopped, halted, or checkstop states (all indicated via the OnCE Status Register (OSR), *Section , "e200z0h OnCE Status Register*) by assertion of one or more debug requests, begins a *debug session*. The **jd_debug_b** output signal indicates that a debug session is in progress, and the OSR will indicate the CPU is in the debug state. Instructions may the be single-stepped by scanning new values into the CPUSCR, and performing a OnCE go+noexit

command (See *Section , "e200z0h OnCE Command Register (OCMD)*). The CPU will then temporarily exit the debug state (but not the debug session) to execute the instruction, and will then return to the debug state (again indicated via the OnCE Status Register (OSR)). The debug session remains in force until the final OnCE go+exit command is executed, at which time the CPU will return to the previous state it was in (unless a new debug request is pending). A scan into the CPUSCR is required prior to executing each go+exit or go+noexit OnCE command.

36.12.2 JTAG/OnCE Pins

The JTAG/OnCE pin interface is used to transfer OnCE instructions and data to the OnCE control block. Depending on the particular resource being accessed, the CPU may need to be placed in the Debug mode. For resources outside of the CPU block and contained in the OnCE block, the processor is not disturbed, and may continue execution. If a processor resource is required, an internal debug request (**dbg_dbgrq**) may be asserted to the CPU by the OnCE controller, and causes the CPU to finish the current instruction being executed, save the instruction pipeline information, enter Debug Mode, and wait for further commands. Asserting **dbg_dbgrq** will cause the chip to exit the low power mode enabled by the setting of MSR_{WE}, as well as temporarily exiting the waiting, stopped or halted power management states.

Table 466 details the primary JTAG/OnCE interface signals.

Table 466. JTAG/OnCE Primary Interface Signals

Signal Name	Type	Description
j_trst_b	I	JTAG test reset
j_tclk	I	JTAG test clock
j_tms	I	JTAG test mode select
j_tdi	I	JTAG test data input
j_tdo	O	Test data out to master controller or pad
j_tdo_en(1)	O	Enables TDO output buffer

1. j_tdo_en is asserted when the TAP controller is in the shift_DR or shift_IR state.

36.12.3 OnCE Internal Interface Signals

The following paragraphs describe the OnCE interface signals to other internal blocks associated with the OnCE controller.

CPU Debug Request (dbg_dbgrq)

The **dbg_dbgrq** signal is asserted by the OnCE control logic to request the CPU to enter the debug state. It may be asserted for a number of different conditions, and causes the CPU to finish the current instruction being executed, save the instruction pipeline information, enter the debug mode, and wait for further commands.

CPU Debug Acknowledge (cpu_dbgack)

The **cpu_dbgack** signal is asserted by the CPU upon entering the debug state. This signal is used as part of the handshake mechanism between the OnCE control logic and the rest

of the CPU. The CPU core may enter debug mode either through a software or hardware event.

CPU Address, Attributes

The CPU address and attribute information are used by a Nexus class 2-4 debug unit with information for real-time address trace information.

CPU Data

The CPU data bus(es) are used to supply a Nexus class 2-4 debug unit with information for real-time data trace capability.

36.12.4 OnCE Interface Signals

The following paragraphs describe additional OnCE interface signals to other external blocks such as a Nexus Controller and external blocks which may need information pertaining to debug operation.

OnCE Enable (jd_en_once)

The OnCE enable signal **jd_en_once** is used to enable the OnCE controller to allow certain instructions and operations to be executed. Assertion of this signal will enable the full OnCE command set, as well as operation of control signals and OnCE Control register functions. When this signal is disabled, only the Bypass, ID and Enable_OnCE commands are executed by the OnCE unit, and all other commands default to a “Bypass” command. The OnCE Status register (OSR) is not visible when OnCE operation is disabled. In addition, OnCE Control register (OCR) functions are disabled, as is the operation of the **jd_de_b** input. Secure systems may choose to leave the **jd_en_once** signal negated until a security check has been performed. Other systems should tie this signal asserted to enable full OnCE operation. The **j_en_once_regsel** output signal is provided to assist external logic performing security checks.

The **jd_en_once** input must only change state during the Test-Logic-Reset, Run-Test/Idle, or Update_DR TAP states. A new value will take affect after one additional **j_tclk** cycle of synchronization. In addition, **jd_enable_once** input signal must not change state during a debug session, or undefined activity may occur.

OnCE Debug Request/Event (jd_de_b, jd_de_en)

If implemented at the SoC level, a system level bidirectional open drain debug event pin **DE_b** (not part of the interface) provides a fast means of entering the Debug Mode of operation from an external command controller (when input) as well as a fast means of acknowledging the entering of the Debug Mode of operation to an external command controller (when output). The assertion of this pin by a command controller causes the CPU core to finish the current instruction being executed, save the instruction pipeline information, enter Debug Mode, and wait for commands to be entered. If **DE_b** was used to enter the Debug Mode then **DE_b** must be negated after the OnCE controller responds with an acknowledge and before sending the first OnCE command. The assertion of this pin by the CPU Core acknowledges that it has entered the Debug Mode and is waiting for commands to be entered.

To support operation of this system pin, the OnCE logic supplies the **jd_de_en** output and samples the **jd_de_b** input when OnCE is enabled (**jd_en_once** asserted). Assertion of **jd_de_b** will cause the OnCE logic to place the CPU into Debug Mode. Once Debug Mode

has been entered, the **jd_de_en** output will be asserted for three **j_tclk** periods to signal an acknowledge. **jd_de_en** can be used to enable the open-drain pulldown of the system level **DE_b** pin.

For systems which do not implement a system level bidirectional open drain debug event pin **DE_b**, the **jd_de_en** and **jd_de_b** signals may still be used to handshake debug entry.

e200z0h OnCE Debug Output (jd_debug_b)

The e200z0h OnCE Debug output **jd_debug_b** is used to indicate to on-chip resources that a debug session is in progress. Peripherals and other units may use this signal to modify normal operation for the duration of a debug session, which may involve the CPU executing a sequence of instructions solely for the purpose of visibility/system control which are not part of the normal instruction stream the CPU would have executed had it not been placed in debug mode. This signal is asserted the first time the CPU enters the debug state, and remains asserted until the CPU is released by a write to the e200z0h OnCE Command Register with the GO and EX bits set, and a register specified as either “No Register Selected” or the CPUSCR. This signal will remain asserted even though the CPU may enter and exit the debug state for each instruction executed under control of the e200z0h OnCE controller. See **Section** for more information on the function of the GO and EX bits. This signal is not normally used by the CPU.

e200z0h CPU Clock On Input (jd_mclk_on)

The e200z0h CPU Clock On input **jd_mclk_on** is used to indicate that the CPU’s **m_clk** input is active. This input signal is expected to be driven by system logic external to the e200z0h core, is synchronized to the **j_tclk** (scan clock) clock domain, and is presented as a status flag on the **j_tdo** output during the Shift_IR state. External firmware may use this signal to ensure proper scan sequences will occur to access debug resources in the **m_clk** clock domain.

Watchpoint Events (jd_watchpt[0:5])

The **jd_watchpt[0:5]** signals may be asserted by the e200z0h OnCE control logic to signal that a watchpoint condition has occurred. Watchpoints do not cause the CPU to be affected. They are provided to allow external visibility only. Watchpoint events are conditioned by the settings in the DBCRx registers.

36.12.5 e200z0h OnCE Controller and Serial Interface

The OnCE Controller contains the OnCE command register, the OnCE decoder, and the status/control register. Figure 515 is a block diagram of the OnCE controller. In operation, the OnCE Command register acts as the IR for the e200z0h TAP controller, and all other OnCE resources are treated as data registers (DR) by the TAP controller. The Command register is loaded by serially shifting in commands during the TAP controller Shift-IR state, and is loaded during the Update-IR state. The Command register selects a resource to be accessed as a data register (DR) during the TAP controller Capture-DR, Shift-DR and Update-DR states.

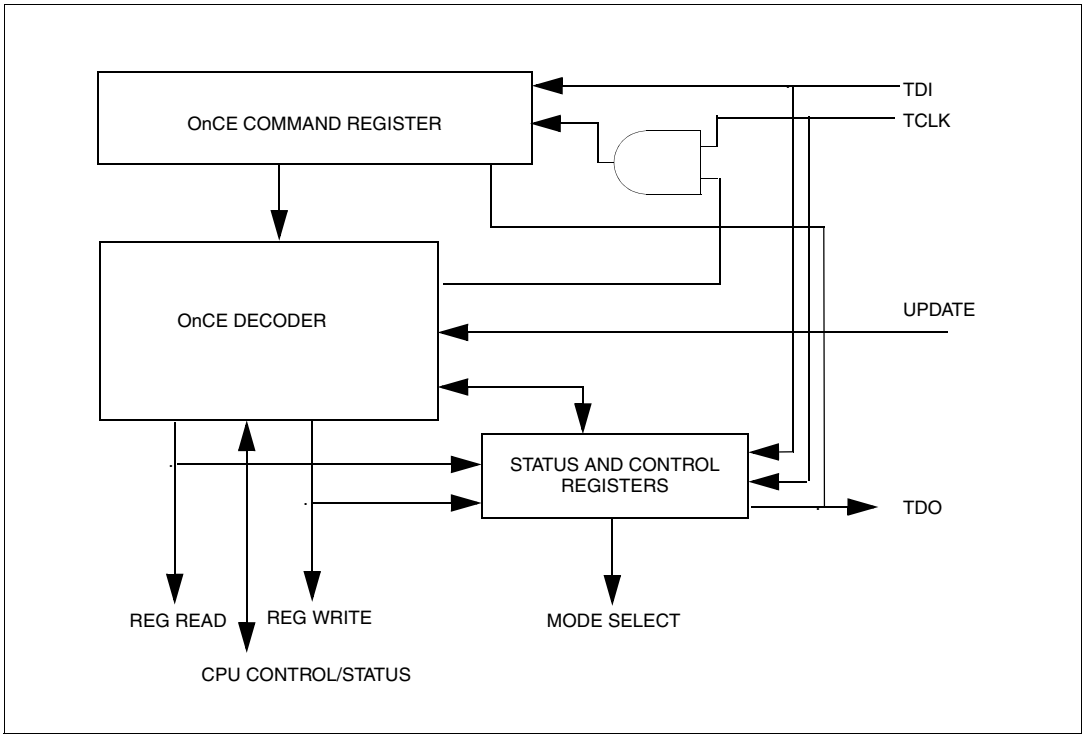


Figure 515. e200z0h OnCE Controller and Serial Interface

e200z0h OnCE Status Register

Status information regarding the state of the e200z0h CPU is latched into the OnCE Status register when the OnCE controller state machine enters the Capture-IR state. When OnCE operation is enabled, this information is provided on the **j_tdo** output in serial fashion when the Shift_IR state is entered following a Capture-IR. Information is shifted out least significant bit first.

MCLK	ERR	CHKSTOP	RESET	HALT	STOP	DEBUG	WAIT	0	1
0	1	2	3	4	5	6	7	8	9

Figure 516. OnCE Status Register

Table 467 provides bit definitions for the Once Status Register.

Table 467. OnCE Status Register Bit Definitions

Bit(s)	Name	Description
0	MCLK	**m_clk** Status Bit 0 – Inactive state 1 – Active state This status bit reflects the logic level on the **jd_mclk_on** input signal after capture by **j_tclk**.
1	ERR	ERROR This bit is used to indicate that an error condition occurred during attempted execution of the last single-stepped instruction (GO+NoExit with CPUSCR or No Register Selected in OCMD), and that the instruction may not have been properly executed. This could occur if an Interrupt (all classes including External, Critical, machine check, Storage, Alignment, Program, etc.) occurred while attempting to perform the instruction single step. In this case, the CPUSCR will contain information related to the first instruction of the Interrupt handler, and no portion of the handler will have been executed.
2	CHKSTOP	CHECKSTOP Mode This bit reflects the logic level on the CPU **p_chkstop** output after capture by **j_tclk**.
3	RESET	RESET Mode This bit reflects the inverted logic level on the CPU **p_reset_b** input after capture by **j_tclk**.
4	HALT	HALT Mode This bit reflects the logic level on the CPU **p_halted** output after capture by **j_tclk**.
5	STOP	STOP Mode This bit reflects the logic level on the CPU **p_stopped** output after capture by **j_tclk**.
6	DEBUG	Debug Mode This bit is asserted once the CPU is in debug mode. It is negated once the CPU exits debug mode (even during a debug session)
7	WAIT	Waiting Mode This bit reflects the logic level on the CPU **p_waiting** output after capture by **j_tclk**.
8	—	Reserved, set to ‘0’ for 1149.1 compliance
9	—	Reserved, set to ‘1’ for 1149.1 compliance

e200z0h OnCE Command Register (OCMD)

The OnCE Command Register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0h OnCE Decoder. The Command Register is shown in Figure 517. The OCMD is updated when the TAP controller enters the Update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the Update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the Update-DR state must be transitioned through in order for an access to occur. In addition, the Update-DR state must also be transitioned through in order for the single-step

and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

R/W	GO	EX	RS[0:6]						
0	1	2	3	4	5	6	7	8	9

Reset - 10'b1000000010 on assertion of **j_trst_b** or **m_por**, or while in the Test_Logic_Reset state

Figure 517. OnCE Command Register

Table 468 provides bit definitions for the Once Command Register.

Table 468. OnCE Command Register Bit Definitions

Bit(s)	Name	Description
0	R/W	Read/Write Command Bit The R/W bit specifies the direction of data transfer. The table below describes the options defined by the R/W bit. 0 – Write the data associated with the command into the register specified by RS[0:6] 1 – Read the data contained in the register specified by RS[0:6] Note: The R/W bit generally ignored for read-only or write-only registers, although the PC FIFO pointer is only guaranteed to be update when R/W=1. In addition, it is ignored for all bypass operations. When performing writes, most registers are sampled in the Capture-DR state into a 32-bit shift register, and subsequently shifted out on j_tdo during the first 32 clocks of Shift-DR.
1	GO	Go Command Bit 0 – Inactive (no action taken) 1 – Execute instruction in IR If the GO bit is set, the chip will execute the instruction which resides in the IR register in the CPUSCR. To execute the instruction, the processor leaves the debug mode, executes the instruction, and if the EX bit is cleared, returns to the debug mode immediately after executing the instruction. The processor goes on to normal operation if the EX bit is set, and no other debug request source is asserted. The GO command is executed only if the operation is a read/write to CPUSCR or a read/write to "No Register Selected". Otherwise the GO bit is ignored.The processor will leave the debug mode after the TAP controller Update-DR state is entered. On a GO+NoExit operation, returning to debug mode is treated as a debug event, thus exceptions such as machine checks and interrupts may take priority and prevent execution of the intended instruction. Debug firmware should mask these exceptions as appropriate. The OSR_{ERR} bit indicates such an occurrence. Note that asynchronous interrupts are blocked on a GO+Exit operation until the first instruction to be executed begins execution.

Table 468. OnCE Command Register Bit Definitions (continued)

Bit(s)	Name	Description
2	EX	Exit Command Bit 0 – Remain in debug mode 1 – Leave debug mode If the EX bit is set, the processor will leave the debug mode and resume normal operation until another debug request is generated. The Exit command is executed only if the Go command is issued, and the operation is a read/write to CPUSCR or a read/write to “No Register Selected”. Otherwise the EX bit is ignored. The processor will leave the debug mode after the TAP controller Update-DR state is entered. Note that if the DR bit in the OnCE control register is set or remains set, or if a hardware-owned bit in DBSR is set and $DBCR0_{EDM}$=1 (external debug mode is enabled), or if another debug request source is asserted, then the processor may return to the debug mode without execution of an instruction, even though the EX bit was set. Note that asynchronous interrupts are blocked on a GO+Exit operation until the first instruction to be executed begins execution.
3:9	RS	Register Select The Register Select bits define which register is source (destination) for the read (write) operation. *Table 469* indicates the e200z0h OnCE register addresses. Attempted writes to read-only registers are ignored.

Table 469 indicates the e200z0h OnCE register addresses.

Table 469. e200z0h OnCE Register Addressing

RS[0:6]	Register Selected
000 0000	Reserved
000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011	Reserved
001 0100 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)

Table 469. e200z0h OnCE Register Addressing (continued)

RS[0:6]	Register Selected
010 0111	Data Value Compare 2 (DVC2)
010 1000 – 010 1011	Reserved
010 1100	Reserved (DBCNT)
010 1101 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100	Reserved (DBCR3)
011 0101	Debug Control Register 4 (DBCR4)
011 0110 – 011 1110	Reserved (do not access)
011 1111	Debug External Resource Control (DBERC0)
100 0000 – 101 0111	Test Register Selects [0:23] (j_test_regsel{0:23])
101 1000 – 101 1111	Reserved (do not access)
110 0000 – 110 1110	Reserved (do not access)
110 1111	Shared Nexus Control Register Select
111 0000 – 111 1001	General Purpose register selects [0:9] (j_gp_regsel[0:9])
111 1010	(Reserved)
111 1011	(Reserved)
111 1100	(Reserved)
111 1101	(Reserved)
111 1110	Enable_OnCE(1)
111 1111	Bypass

1. Causes assertion of the j_en_once_regsel output.

The Once Decoder receives as input the 10-bit command from the OCMD, and status signals from the processor, and generates all the strobes required for reading and writing the selected OnCE registers.

Single stepping of instructions is performed by placing the CPU in debug mode, scanning in appropriate information into the CPUSCR, and setting the Go bit (with the EX bit cleared) with the RS field indicating either the CPUSCR or No Register Selected. After executing a single instruction, the CPU will re-enter debug mode and await further commands. During single-stepping, exception conditions may occur if not properly masked by debug firmware (interrupts, machine checks, bus error conditions, etc.) and may prevent the desired instruction from being successfully executed. The OSR_{ERR} bit is set to indicate this condition. In these cases, values in the CPUSCR will correspond to the first instruction of the exception handler.

Additionally, the $DBCR0_{EDM}$ bit is forced to '1' internally while single-stepping to prevent Debug events from generating Debug interrupts. Also, during a debug session, the DBSR is frozen from updates due to debug events regardless of $DBCR0_{EDM}$. They may still be modified during a debug session via a single-stepped **mtspr** instruction if $DBCRO_{EDM}$ is programmed to a '0', or via OnCE access if $DBCR0_{EDM}$ is set.

e200z0h OnCE Control Register (OCR)

The e200z0h OnCE Control Register is a 32-bit register used to force the e200z0h core into debug mode and to enable / disable sections of the e200z0h OnCE control logic. The control bits are read/write. These bits are only effective while OnCE is enabled (**jd_en_once** asserted). The OCR is shown in *Figure 518*.

Figure 518. OnCE Control Register

	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	I_DMDIS	0	0	I_DVLE	I_DI	I_DM	0	I_DE
W																
Reset(1)	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	D_DMDIS	0	0	D_DW	D_DI	D_DM	D_DG	D_DE	0	0	0	0	0	WKUP	FDB	DR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. 0xo000_0000 on **m_por**, **j_trst_b**, or entering Test_logic_Reset state

Table 470 provides bit definitions for the OnCE Control Register.

Table 470. OnCE Control Register Bit Definitions

Bit(s)	Name	Description
0:7	—	Reserved
8	I_DMDIS(1)	Instruction Side Debug MMU Disable Control Bit (I_DMDIS) 0 – MMU not disabled for debug sessions 1 – MMU disabled for debug sessions This bit may be used to control whether the MMU is enabled normally, or whether the MMU is disabled during a debug session for Instruction Accesses. When enabled, the MMU functions normally. When disabled, for Instruction Accesses, no address translation is performed (1:1 address mapping), and the TLB VLE, I,M, and E bits are taken from the OCR bits I_VLE, I_DI, I_DM, and I_DE bits. The W and G bits are assumed '0'. The SX and UX access permission control bits are set to'1' to allow full access. When disabled, no TLB miss or TLB exceptions are generated for Instruction accesses. External access errors can still occur.
9:10	—	Reserved

Table 470. OnCE Control Register Bit Definitions (continued)

Bit(s)	Name	Description
11	I_DVLE(1)	Instruction Side Debug TLB 'VLE' Attribute Bit (I_DVLE) This bit is used to provide the 'VLE' attribute bit to be used when the MMU is disabled during a debug session.
12	I_DI(1)	Instruction Side Debug TLB 'I' Attribute Bit (I_DI) This bit is used to provide the 'I' attribute bit to be used for Instruction accesses when the MMU is disabled for Instruction accesses during a debug session.
13	I_DM(1)	Instruction Side Debug TLB 'M' Attribute Bit (I_DM) This bit is used to provide the 'M' attribute bit to be used for Instruction accesses when the MMU is disabled for Instruction accesses during a debug session.
14	—	Reserved
15	I_DE(1)	Instruction Side Debug TLB 'E' Attribute Bit (I_DE) This bit is used to provide the 'E' attribute bit to be used for Instruction accesses when the MMU is disabled for Instruction accesses during a debug session.
16	D_DMDIS(1)	Data Side Debug MMU Disable Control Bit (D_DMDIS) 0 – MMU not disabled for debug sessions 1 – MMU disabled for debug sessions This bit may be used to control whether the MMU is enabled normally, or whether the MMU is disabled during a debug session for Data Accesses. When enabled, the MMU functions normally. When disabled, for Data Accesses, no address translation is performed (1:1 address mapping), and the TLB WIMGE bits are taken from the OCR bits D_DW, D_DI, D_DM, D_DG, and D_DE bits. The SR, SW, UR, and UW access permission control bits are set to'1' to allow full access. When disabled, no TLB miss or TLB exceptions are generated for Data accesses. External access errors can still occur.
17:18	—	Reserved
19	D_DW(1)	Data Side Debug TLB 'W' Attribute Bit (D_DW) This bit is used to provide the 'W' attribute bit to be used for Data accesses when the MMU is disabled for Data accesses during a debug session.
20	D_DI(1)	Data Side Debug TLB 'I' Attribute Bit (D_DI) This bit is used to provide the 'I' attribute bit to be used for Data accesses when the MMU is disabled for Data accesses during a debug session.
21	D_DM(1)	Data Side Debug TLB 'M' Attribute Bit (D_DM) This bit is used to provide the 'M' attribute bit to be used for Data accesses when the MMU is disabled for Data accesses during a debug session.
22	D_DG(1)	Data Side Debug TLB 'G' Attribute Bit (D_DG) This bit is used to provide the 'G' attribute bit to be used for Data accesses when the MMU is disabled for Data accesses during a debug session.
23	D_DE(1)	Data Side Debug TLB 'E' Attribute Bit (D_DE) This bit is used to provide the 'E' attribute bit to be used for Data accesses when the MMU is disabled for Data accesses during a debug session.
24:28	—	Reserved

Table 470. OnCE Control Register Bit Definitions (continued)

Bit(s)	Name	Description
29	WKUP	Wakeup Request Bit (WKUP) This control bit may be used to force the e200z0h **p_wakeup** output signal to be asserted. This control function may be used by debug firmware to request that the chip-level clock controller restore the **m_clk** input to normal operation regardless of whether the CPU is in a low power state to ensure that debug resources may be properly accessed by external hardware through scan sequences.
30	FDB	Force Breakpoint Debug Mode Bit (FDB) This control bit is used to determine whether the processor is operating in breakpoint debug enable mode or not. The processor may be placed in breakpoint debug enable mode by setting this bit. In breakpoint debug enable mode, execution of the '**bkpt**' pseudo-instruction will cause the processor to enter debug mode, as if the $\overline{\textbf{jd\_de\_b}}$ input had been asserted. This bit is qualified with $DBCR0_{EDM}$, which must be set for FDB to take effect.
31	DR	CPU Debug Request Control Bit This control bit is used to unconditionally request the CPU to enter the Debug Mode. The CPU will indicate that Debug Mode has been entered via the data scanned out in the shift-IR state. 0 – No Debug Mode request 1 – Unconditional Debug Mode request When the DR bit is set the processor will enter Debug mode at the next instruction boundary.

1. Unused by Z0Hn2p

36.12.6 Access to Debug Resources

Resources contained in the e200z0h OnCE Module which do not require the e200z0h processor core to be halted for access may be accessed while the e200z0h core is running, and will not interfere with processor execution. Accesses to other resources such as the CPUSCR require the e200z0h core to be placed in debug mode to avoid synchronization hazards. Debug firmware may ensure that it is safe to access these resources by determining the state of the e200z0h core prior to access. Note that a scan operation to update the CPUSCR is required prior to exiting debug mode if debug mode has been entered.

Some cases of write accesses other than accesses to the OnCE Command and Control registers, or the EDM bit of DBCR0 require the e200z0h **m_clk** to be running for proper operation. The OnCE control register provides a means of signaling this need to a system level clock control module.

In addition, since the CPU may cause multiple bits of certain registers to change state, reads of certain registers while the CPU is running (DBSR, etc.) may not have consistent bit settings unless read twice with the same value indicated. In order to guarantee that the contents are consistent, the CPU should be placed into debug mode, or multiple reads should be performed until consistent values have been obtained on consecutive reads.

Table 471 provides a list of access requirements for OnCE registers.

Table 471. OnCE Register Access Requirements

Register Name	Access Requirements				
	Requires jd_en_once to be asserted	Requires DBCR0 [EDM] = 1	Requires m_clk active for Write Access	Requires CPU to be halted for Read Access	Requires CPU to be halted for Write Access
Enable_OnCE	N	N	N	N	—
Bypass	N	N	N	N	N
CPUSCR	Y	Y	Y	Y	Y
DAC1	Y	Y	Y	N	(1)
DAC2	Y	Y	Y	N	(1)
DBCR0(2)	Y	Y	Y	N	(1)
DBCR1	Y	Y	Y	N	(1)
DBCR2	Y	Y	Y	N	(1)
DBCR4	Y	Y	Y	N	(1)
DBERC0	Y	N	Y	N	(1)
DBSR	Y	Y	Y	N(3)	(1)
IAC1–4	Y	Y	Y	N	(1)
JTAG ID(4)	N	N	—	N	—
OCR	Y	N	N	N	N
OSR(5)	Y	N	—	N	—
Cache Debug Access Control (CDACNTL)(6),(7)	Y	N	Y	Y	Y
Cache Debug Access Data (CDADATA)(6),(7)	Y	N	Y	Y	Y
External GPRs	Y	N	N	N	N

1. Writes to these registers while the CPU is running may have unpredictable results due to the pipelined nature of operation, and the fact that updates are not synchronized to a particular clock, instruction, or bus cycle boundary, therefore it is strongly recommended to ensure the processor is first placed into debug mode before updates to these registers are performed.
2. $DBCR0_{EDM}$ access only requires **jd_en_once** asserted.
3. Reads of these registers while the CPU is running may not give data that is self-consistent due to synchronization across clock domains.
4. Read-only.
5. Read-only, accessed by scanning out IR while **jd_en_once** is asserted.
6. Not present on Z0Hn2p
7. CPU must be in debug mode with clocks running.

36.12.7 Methods of Entering Debug Mode

The OnCE Status Register indicates that the CPU has entered the debug mode via the DEBUG status bit. The following sections describe how e200z0h Debug Mode is entered assuming the OnCE circuitry has been enabled. e200z0h OnCE operation is enabled by the assertion of the **jd_en_once** input (see Section).

External Debug Request During RESET

Holding the **jd_de_b** signal asserted during the assertion of **p_reset_b**, and continuing to hold it asserted following the negation of **p_reset_b** causes the e200z0h core to enter Debug Mode. After receiving an acknowledge via the OnCE Status Register DEBUG bit, the external command controller should negate the **jd_de_b** signal before sending the first command. Note that in this case the e200z0h core does not execute an instruction before entering Debug Mode, although the first instruction to be executed may be fetched prior to entering Debug Mode.

In this case, all values in the debug scan chain will be undefined, and the external Debug Control Module is responsible for proper initialization of the chain before debug mode is exited. In particular, the exception processing associated with reset, may not be performed when the debug mode is exited, thus, the Debug controller must initialize the PC, MSR, and IR to the image that the processor would have obtained in performing reset exception processing, or must cause the appropriate reset to be re-asserted.

Debug Request During RESET

Asserting a debug request by setting the DR bit in the OCR during the assertion of **p_reset_b** causes the chip to enter debug mode. In this case the chip may fetch the first instruction of the reset exception handler, but does not execute an instruction before entering debug mode. In this case, all values in the debug scan chain will be undefined, and the external Debug Control Module is responsible for proper initialization of the chain before debug mode is exited. In particular, the exception processing associated with reset may not be performed when the debug mode is exited, thus, the Debug controller must initialize the PC, MSR, and IR to the image that the processor would have obtained in performing reset exception processing, or must cause the appropriate reset to be re-asserted.

Debug Request During Normal Activity

Asserting a debug request by setting the DR bit in the OCR during normal chip activity causes the chip to finish the execution of the current instruction and then enter the debug mode. Note that in this case the chip completes the execution of the current instruction and stops after the newly fetched instruction enters the CPU instruction register. This process is the same for any newly fetched instruction including instructions fetched by the interrupt processing, or those that will be aborted by the interrupt processing.

Debug Request During Waiting, Halted or Stopped State

Asserting a debug request by setting the DR bit in the OCR when the chip is in the Waiting state (**p_waiting** asserted), Halted state (**p_halted** asserted) or Stopped state (**p_stopped** asserted) causes the CPU to exit the state and enter the debug mode once the CPU clock **m_clk** has been restored. Note that in this case, the CPU will negate the **p_waiting, p_halted** and **p_stopped** outputs. Once the debug session has ended, the CPU will return to the state it was in prior to entering debug mode.

To signal the chip-level clock generator to re-enable **m_clk**, the **p_wakeup** output will be asserted whenever the debug block is asserting a debug request to the CPU due to OCR_{DR} being set, or **jd_de_b** assertion, and will remain set from then until the debug session ends (**jd_debug_b** goes from asserted to negated). In addition, the status of the **jd_mclk_on** input (after synchronization to the **j_tclk** clock domain) may be sampled along with other status bits from the **j_tdo** output during the Shift_IR TAP controller state. This status may be used if necessary by external debug firmware to ensure proper scan sequences occur to registers in the **m_clk** clock domain.

Software Request During Normal Activity

Upon executing a '**bkpt'** pseudo-instruction (for e200z0h, defined to be an all 0's instruction opcode) when the OCR register's (FDB) bit is set (debug mode enable control bit is true), and $DBCR0_{EDM}$=1, the CPU enters the debug mode after the instruction following the '**bkpt'** pseudo-instruction has entered the instruction register.

36.12.8 CPU Status and Control Scan Chain Register (CPUSCR)

A number of on-chip registers store the CPU pipeline status and are configured in a single scan chain for access by the e200z0h OnCE controller. The CPUSCR register contains these processor resources, which are used to restore the pipeline and resume normal chip activity upon return from the debug mode, as well as a mechanism for the emulator software to access processor and memory contents. Figure 519 shows the block diagram of the pipeline information registers contained in the CPUSCR. Once debug mode has been entered, it is required to scan in and update this register prior to exiting debug mode.

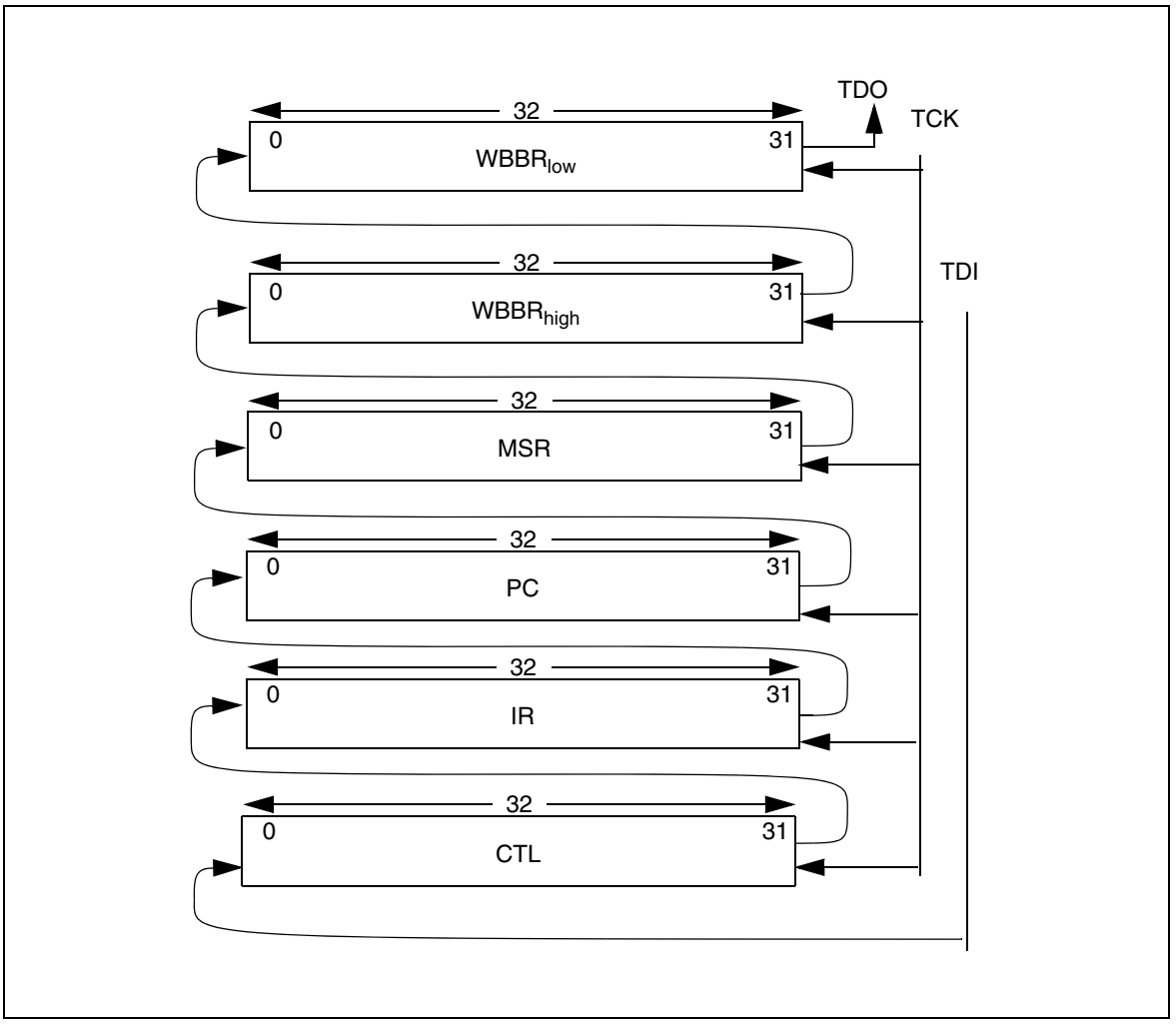


Figure 519. CPU Scan Chain Register (CPUSCR)

Instruction Register (IR)

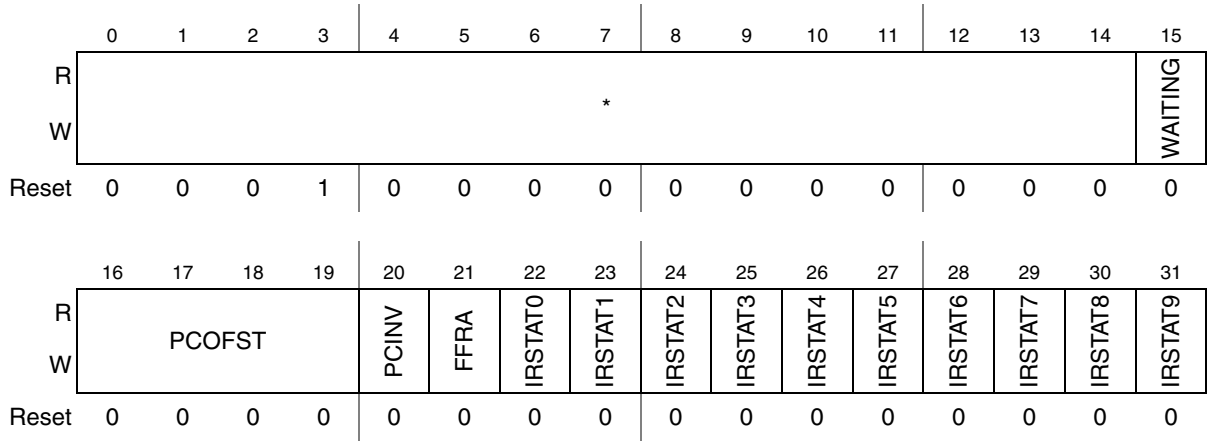
The Instruction Register (IR) provides a mechanism for controlling the debug session by serving as a means for forcing in selected instructions, and then causing them to be executed in a controlled manner by the debug control block. The opcode of the next instruction to be executed when entering debug mode is contained in this register when the scan-out of this chain begins. This value should be saved for later restoration if continuation of the normal instruction stream is desired.

On scan-in, in preparation for exiting debug mode, this register is filled with an instruction opcode selected by debug control software. By selecting appropriate instructions and controlling the execution of those instructions, the results of execution may be used to examine or change memory locations and processor registers. The debug control module external to the processor core will control execution by providing a single-step capability. Once the debug session is complete and normal processing is to be resumed, this register may be loaded with the value originally scanned out.

Control State Register (CTL)

The Control State Register (CTL) is a 32-bit register that stores the value of certain internal CPU state variables before the debug mode is entered. This register is affected by the operations performed during the debug session and should normally be restored by the external command controller when returning to normal mode. In addition to saved internal state variables, two of the bits are used by emulation firmware to control the debug process. In certain circumstances, emulation firmware must modify the content of this register as well as the PC and IR values in the CPUSCR before exiting debug mode. These cases are described below. *Figure 520*.

Figure 520. Control State Register (CTL)



WAITING — Waiting State Status

This bit indicates whether the CPU was in the waiting state prior to entering debug mode. If set, the CPU was in the waiting state. Upon exiting a debug session, the value of this bit in the restored CPUSCR will determine whether the CPU re-enters the waiting state on a go+exit.

0: CPU was not in the waiting state when debug mode was entered

1: CPU was in the waiting state when debug mode was entered

PCOFST — PC Offset Field

This field indicates whether the value in the PC portion of the CPUSCR must be adjusted prior to exiting debug mode. Due to the pipelined nature of the CPU, the PC value must be backed-up by emulation software in certain circumstances. The PCOFST field specifies the value to be subtracted from the original value of the PC. This adjusted PC value should be restored into the PC portion of the CPUSCR just prior to exiting debug mode with a go+exit. In the event the PCOFST is non-zero, the IR should be loaded with a nop instruction instead

of the original IR value, otherwise the original value of IR should be restored. (But see PCINV which overrides this field)

0000: No correction required.

0001: Subtract 0x04 from PC.

0010: Subtract 0x08 from PC.

0011: Subtract 0x0C from PC.

0100: Subtract 0x10 from PC.

0101: Subtract 0x14 from PC.

all other encodings are reserved

* — Internal State Bits

These control bits represent internal processor state and should be restored to their original value after a debug session is completed, i.e when a e200z0h OnCE command is issued with the GO and EX bits set and not ignored. When performing instruction execution during a debug session (see *Section , "e200z0h OnCE Debug Output (jd_debug_b)*) which is not part of the normal program execution flow, these bits should be set to a 0.

PCINV — PC and IR Invalid Status Bit

This status bit indicates that the values in the IR and PC portions of the CPUSCR are invalid. Exiting debug mode with the saved values in the PC and IR will have unpredictable results. Debug firmware should initialize the PC and IR values in the CPUSCR with desired values prior to exiting debug mode if this bit was set when debug mode was initially entered.

0: No error condition exists.

1: Error condition exists. PC and IR are corrupted.

FFRA— Feed Forward RA Operand Bit

This control bit causes the content of the $WBBR_{low}$ to be used as the RA This control bit causes the content of the WBBR to be used as the RA operand value (RS for logical, mtspr, mtdcr, cntlzw, and shift operations, RX for VLE se_ instructions, RT for e_{logical_op}2i type instructions, and the value to use as the PC for calculating the LR update value for branch with link type instructions) of the first instruction to be executed following an update of the CPUSCR. This allows the debug firmware to update processor registers — initialize the WBBR with the desired value, set the FFRA bit, and execute a ori Rx,Rx,0 instruction to the desired register.

0: No action.

1: Content of $WBBR_{low}$ used as operand value

IRStat0 — IR Status Bit 0

This control bit indicates a TEA status for the IR.

0: No TEA occurred on the fetch of this instruction.

1: TEA occurred on the fetch of this instruction.

IRStat1 — IR Status Bit 1

This control bit indicates a TLB Miss status for the IR. (Note that this bit is reserved.)

0: No TLB Miss occurred on the fetch of this instruction.

1: TLB Miss occurred on the fetch of this instruction.

IRStat2 — IR Status Bit 2

This control bit indicates an Instruction Address Compare 1 event status for the IR.

0: No Instruction Address Compare 1 event occurred on the fetch of this instruction.

1: An Instruction Address Compare 1 event occurred on the fetch of this instruction.

IRStat3 — IR Status Bit 3

This control bit indicates an Instruction Address Compare 2 event status for the IR.

0: No Instruction Address Compare 2 event occurred on the fetch of this instruction.

1: An Instruction Address Compare 2 event occurred on the fetch of this instruction.

IRStat4 — IR Status Bit 4

This control bit indicates an Instruction Address Compare 3 event status for the IR.

0: No Instruction Address Compare 3 event occurred on the fetch of this instruction.

1: An Instruction Address Compare 3 event occurred on the fetch of this instruction.

IRStat5 — IR Status Bit 5

This control bit indicates an Instruction Address Compare 4 event status for the IR.

0: No Instruction Address Compare 4 event occurred on the fetch of this instruction.

1: An Instruction Address Compare 4 event occurred on the fetch of this instruction.

IRStat6 — IR Status Bit 6

This control bit indicates a Parity Error status for the IR. (Note that this bit is reserved.)

0: No Parity Error occurred on the fetch of this instruction.

1: Parity Error occurred on the fetch of this instruction.

IRStat7 — IR Status Bit 7

This control bit indicates a Precise External Termination Error status for the IR.

0: No Precise External Termination Error occurred on the fetch of this instruction.

1: Precise External Termination Error occurred on the fetch of this instruction.

IRStat8 — IR Status Bit 8

This control bit indicates the Power Architecture technology VLE status for the IR. (Note that this bit is always set on Z0Hn2p.)

0: IR contains a BookE instruction.

1: IR contains a Power Architecture technology VLE instruction, aligned in the Most Significant Portion of IR if 16-bit.

IRStat9 — IR Status Bit 9

This control bit indicates the Power Architecture technology VLE Byte-ordering Error status for the IR, or a BookE misaligned instruction fetch, depending on the state of IRStat8. (Note that this bit is reserved on Z0Hn2p.)

0: IR contains an instruction without a byte-ordering error and no Misaligned Instruction Fetch Exception has occurred (no MIF).

1: If IRStat8 = ‘0’, A BookE Misaligned Instruction Fetch Exception has occurred while filling the IR.
If IRStat8 = ‘1’, IR contains an instruction with a byte-ordering error due to mismatched VLE page attributes, or due to E indicating little-endian for a VLE page.

Emulation firmware should modify the content of the CTL, PC, and IR values in the CPUSCR during execution of debug related instructions as well as just prior to exiting debug

with a go+exit command. During the debug session, the CTL register should be written with the FFRA bit set as appropriate, and all other bits set to '0', and the IR set to the value of the desired instruction to be executed. IRStat8 will be used to determine the type of instruction present in the IR.

Just prior to exiting debug mode with a go+exit, the PCINV status bit which was originally present when debug mode was first entered should be tested, and if set, the PC and IR initialized for performing whatever recovery sequence is appropriate for a faulted exception vector fetch. If the PCINV bit is cleared, then the PCOFST bits should be examined to determine whether the PC value must be adjusted. Due to the pipelined nature of the CPU, the PC value must be backed-up by emulation software in certain circumstances. The PCOFST field specifies the value to be subtracted from the original value of the PC. This adjusted PC value should be restored in to the PC portion of the CPUSCR just prior to exiting debug mode with a go+exit. In the event the PCOFST is non-zero, the IR should be loaded with a nop instruction (such as **ori r0,r0,0**) instead of the original IR value, otherwise the original value of IR should be restored. Note that when a correction is made to the PC value, it will generally point to the last completed instruction, although that instruction will not be re-executed. The nop instruction is executed instead, and instruction fetch and execution will resume at location PC+4. IRStat8 will be used to determine the type of instruction present in the IR, thus should be cleared in this case. Note that debug events which may occur on the nop (ICMP) will be generated (and optionally counted) if enabled.

For the CTL register, the internal state bits should be restored to their original value. The IRStatus bits should be set to '0's if the PC was adjusted. If no PC adjustment was performed, emulation firmware should determine whether IRStat2-5 should be set to '0' to avoid re-entry into debug mode for an instruction breakpoint request. Upon exiting debug mode with go+exit, if one of these bits is set, debug mode will be re-entered prior to any further instruction execution.

Program Counter Register (PC)

The PC is a 32-bit register that stores the value of the program counter which was present when the chip entered the debug mode. It is affected by the operations performed during the debug mode and must be restored by the external command controller when the CPU returns to normal mode. PC normally points to the instruction contained in the IR portion of CPUSCR. If debug firmware wishes to redirect program flow to an arbitrary location, the PC and IR should be initialized to correspond to the first instruction to be executed upon resumption of normal processing. Alternatively, the IR may be set to a nop and the PC set to point to the location prior to the location at which it is desired to redirect flow to. On exiting debug mode, the nop will be executed, and instruction fetch and execution will resume at PC+4.

Write-Back Bus Register ($WBBR_{low}$, $WBBR_{high}$)

WBBR is used as a means of passing operand information between the CPU and the external command controller. Whenever the external command controller needs to read the contents of a register or memory location, it will force the chip to execute an instruction that brings that information to WBBR. $WBBR_{low}$ holds the 32-bit result of most instructions including load data returned for a load or load with update instruction. $WBBR_{high}$ holds the updated effective address calculated by a load with update instruction. It is undefined for other instructions.

As an example, to read the lower 32 bits of processor register **r1**, an **e_ori r1,r1,0** instruction is executed, and the result value of the instruction will be latched into $WBBR_{low}$. The contents of $WBBR_{low}$ can then be delivered serially to the external command controller. To

update a processor resource, this register is initialized with a data value to be written, and an **e_ori** instruction is executed which uses this value as a substitute data value. The Control State register FFRA bit forces the value of the $WBBR_{low}$ to be substituted for the normal RS source value of the **e_ori** instruction, thus allowing updates to processor registers to be performed (refer to Section for more detail on the CTL_{FFRA} bit).

$WBBR_{low}$ and $WBBR_{high}$ are generally undefined on instructions which do not writeback a result, and due to control issues are not defined on **lmw** or branch instructions as well.

Machine State Register (MSR)

The MSR is a 32-bit register used to read/write the Machine State Register. Whenever the external command controller needs to save or modify the contents of the Machine State Register, this register is used.This register is affected by the operations performed during the debug mode and must be restored by the external command controller when returning to normal mode.

36.13 Watchpoint Support

e200z0h supports the generation and signalling of watchpoints when operating in internal debug mode ($DBCR0_{IDM}$=1) or in external debug mode ($DBCR0_{EDM}$=1). Watchpoints are indicated with a dedicated set of interface signals. The **jd_watchpoint[0:5]** output signals are used to indicate that a watchpoint has occurred.

Each debug address compare function (IAC1–4, DAC1–2) is capable of triggering a watchpoint output. The DBCRx control fields are used to configure watchpoints, regardless of whether events are enabled in DBCR0. Watchpoints may occur whenever an associated event would have been posted in the Debug Status Register if enabled. No explicit enable bits are provided for watchpoints; they are always enabled by definition (except during a debug session). during a debug session. If not desired, the base address values for these events may be programmed to an unused system address. MSR_{DE} has no effect on watchpoint generation.

External logic may monitor the assertion of these signals for debugging purposes. Watchpoints are signaled in the clock cycle following the occurrence of the actual event. The Nexus2+ module also monitors assertion of these signals for various development control purposes.

Table 472. Watchpoint Output Signal Assignments

Signal Name	Type	Description
jd_watchpt[0]	IAC1	Instruction Address Compare 1 watchpoint Asserted whenever an IAC1 compare occurs regardless of being enabled to set DBSR status
jd_watchpt[1]	IAC2	Instruction Address Compare 2 watchpoint Asserted whenever an IAC2 compare occurs regardless of being enabled to set DBSR status
jd_watchpt[2]	IAC3	Instruction Address Compare 3 watchpoint Asserted whenever an IAC3 compare occurs regardless of being enabled to set DBSR status

Table 472. Watchpoint Output Signal Assignments (continued)

Signal Name	Type	Description
jd_watchpt[3]	IAC4	Instruction Address Compare 4 watchpoint Asserted whenever an IAC4 compare occurs regardless of being enabled to set DBSR status
jd_watchpt[4]	DAC1(1)	Data Address Compare 1 watchpoint Asserted whenever a DAC1 compare occurs regardless of being enabled to set DBSR status
jd_watchpt[5]	DAC2(1)	Data Address Compare 2 watchpoint Asserted whenever a DAC2 compare occurs regardless of being enabled to set DBSR status

1. If the corresponding event is completely disabled in DBCR0, either load-type or store-type data accesses are allowed to generate watchpoints, otherwise watchpoints are generated only for the enabled conditions.

36.14 Basic Steps for Enabling, Using, and Exiting External Debug Mode

The following steps show one possible scenario for a debugger wishing to use the external debug facilities. *This simplified flow is intended to illustrate basic operations, but does not cover all potential methods in depth.*

Enabling External Debug Mode and initializing Debug registers

- The debugger should ensure that the **jd_en_once** control signal is asserted in order to enable OnCE operation
- Select the OCR and write a value to it in which OCR_{DR}, OCR_{WKUP}, are set to '1'. The tap controller must step through the proper states as outlined earlier. This step will place the CPU in a debug state in which it is halted and awaiting single-step commands or a release to normal mode
- Scan out the value of the OSR to determine that the CPU clock is running and the CPU has entered the Debug state. This can be done in conjunction with a Read of the CPUSCR. The OSR is shifted out during the Shift_IR state. The CPUSCR will be shifted out during the Shift_DR state. The debugger should save the scanned-out value of CPUSCR for later restoration.
- Select the DBCR0 register and update it with the $DBCR0_{EDM}$ bit set
- Clear the DBSR status bits
- Write appropriate values to the DBCRx, IAC, DAC registers. Note that the initial write to DBCR0 will only affect the EDM bit, so the remaining portion of the register must now be initialized, keeping the EDM bit set

At this point the system is ready to commence debug operations. Depending on the desired operation, different steps must occur.

- Optionally, set the OCR_{I_DMDIS} and/or OCR_{D_DMDIS} control bits to ensure that no TLB misses will occur while performing the debug operations
- Optionally, ensure that the values entered into the MSR portion of the CPUSCR during the following steps cause interrupt to be disabled (clearing MSR_{EE} and MSR_{CE}). This will ensure that external interrupt sources do not cause single-step errors.

To single-step the CPU:

- The debugger scans in either a new or a previously saved value of the CPUSCR (with appropriate modification of the PC and IR as described in *Section , "Control State Register (CTL)*), with a Go+Noexit OnCE Command value.
- The debugger scans out the OSR with "no-register selected", Go cleared, and determines that the PCU has re-entered the Debug state and that no ERR condition occurred

To return the CPU to normal operation (without disabling external debug mode)

- The OCR_{DMDIS}, OCR_{DR}, control bits should be cleared, leaving the OCR_{WKUP} bit set
- The debugger restores the CPUSCR with a previously saved value of the CPUSCR (with appropriate modification of the PC and IR as described in *Section , "Control State Register (CTL)*), with a Go+Exit OnCE Command value.
- The OCR_{WKUP} bit may then be cleared

To exit External Debug Mode

- The debugger should place the CPU in the debug state via the OCR_{DR} with OCR_{WKUP} asserted, scanning out and saving the CPUSCR
- The debugger should write the DBCRx registers as needed, likely clearing every enable except the $DBCR0_{EDM}$ bit
- The debugger should write the DBSR to a cleared state
- The debugger should rewrite the DBCR0 with all bits including EDM cleared
- The debugger should clear the OCR_{DR} bit
- The debugger restores the CPUSCR with the previously saved value of the CPUSCR (with appropriate modification of the PC and IR as described in *Section , "Control State Register (CTL)*), with a Go+Exit OnCE Command value.
- The OCR_{WKUP} bit may then be cleared

Note: These steps are meant by way of examples, and are not meant to be an exact template for debugger operation.

36.15 Functional description

The NDI block is implemented by integrating the following blocks on the SPC560P40/34:

- Nexus e200z0 development interface (OnCE subblock)
- Nexus port controller (NPC) block

36.15.1 Enabling Nexus clients for TAP access

After the conditions have been met to bring the NDI out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in *Table 473*. After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in *Table 474*. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Table 473. JTAGC Instruction opcodes to enable Nexus clients

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z0 TAP controller

Table 474. Nexus client JTAG instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus ENABLE instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z0 OnCE JTAG Instruction Opcodes(1)		
BYPASS	Opcode for the e200z0 OnCE BYPASS instruction (10-bits)	0x7F

1. Refer to the e200z0 reference manual for a complete list of available OnCE instructions.

36.15.2 Debug mode control

On SPC560P40/34, program breaks can be requested when a Nexus event is triggered.

Appendix A Registers Under Protection

For SPC560P40/34, the Register Protection module is operable on the registers listed in *Table 475*.

Table 475. Registers under protection

Module	Register	Register size (bits)	Register offset	Protected bitfields
Code Flash—Base address: 0xC3F8_8000 4 registers to protect				
Code Flash	MCR	32	0x0000	bits[0:31]
Code Flash	PFCR0	32	0x001C	bits[0:31]
Code Flash	PFCR1	32	0x0020	bits[0:31]
Code Flash	PFAPR	32	0x0024	bits[0:31]
Data Flash—Base address: 0xC3F8_C000 1 register to protect				
Data Flash	MCR	32	0x0000	bits[0:31]
SIU lite—Base address: 0xC3F9_0000 97 registers to protect				
SIUL	IRER	32	0x0018	bits[0:31]
SIUL	IREER	32	0x0028	bits[0:31]
SIUL	IFEER	32	0x002C	bits[0:31]
SIUL	IFER	32	0x0030	bits[0:31]
SIUL	PCR0	16	0x0040	bits[0:15]
SIUL	PCR1	16	0x0042	bits[0:15]
SIUL	PCR2	16	0x0044	bits[0:15]
SIUL	PCR3	16	0x0046	bits[0:15]
SIUL	PCR4	16	0x0048	bits[0:15]
SIUL	PCR5	16	0x004A	bits[0:15]
SIUL	PCR6	16	0x004C	bits[0:15]
SIUL	PCR7	16	0x004E	bits[0:15]
SIUL	PCR8	16	0x0050	bits[0:15]
SIUL	PCR9	16	0x0052	bits[0:15]
SIUL	PCR10	16	0x0054	bits[0:15]
SIUL	PCR11	16	0x0056	bits[0:15]
SIUL	PCR12	16	0x0058	bits[0:15]
SIUL	PCR13	16	0x005A	bits[0:15]

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR14	16	0x005C	bits[0:15]
SIUL	PCR15	16	0x005E	bits[0:15]
SIUL	PCR16	16	0x0060	bits[0:15]
SIUL	PCR17	16	0x0062	bits[0:15]
SIUL	PCR18	16	0x0064	bits[0:15]
SIUL	PCR19	16	0x0066	bits[0:15]
SIUL	PCR20	16	0x0068	bits[0:15]
SIUL	PCR21	16	0x006A	bits[0:15]
SIUL	PCR22	16	0x006C	bits[0:15]
SIUL	PCR23	16	0x006E	bits[0:15]
SIUL	PCR24	16	0x0070	bits[0:15]
SIUL	PCR25	16	0x0072	bits[0:15]
SIUL	PCR26	16	0x0074	bits[0:15]
SIUL	PCR27	16	0x0076	bits[0:15]
SIUL	PCR28	16	0x0078	bits[0:15]
SIUL	PCR29	16	0x007A	bits[0:15]
SIUL	PCR30	16	0x007C	bits[0:15]
SIUL	PCR31	16	0x007E	bits[0:15]
SIUL	PCR32	16	0x0080	bits[0:15]
SIUL	PCR33	16	0x0082	bits[0:15]
SIUL	PCR48	16	0x00A0	bits[0:15]
SIUL	PCR49	16	0x00A2	bits[0:15]
SIUL	PCR50	16	0x00A4	bits[0:15]
SIUL	PCR51	16	0x00A6	bits[0:15]
SIUL	PCR52	16	0x00A8	bits[0:15]
SIUL	PCR53	16	0x00AA	bits[0:15]
SIUL	PCR54	16	0x00AC	bits[0:15]
SIUL	PCR55	16	0x00AE	bits[0:15]
SIUL	PCR56	16	0x00B0	bits[0:15]
SIUL	PCR57	16	0x00B2	bits[0:15]
SIUL	PCR58	16	0x00B4	bits[0:15]
SIUL	PCR59	16	0x00B6	bits[0:15]

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR60	16	0x00B8	bits[0:15]
SIUL	PCR61	16	0x00BA	bits[0:15]
SIUL	PCR62	16	0x00BC	bits[0:15]
SIUL	PCR63	16	0x00BE	bits[0:15]
SIUL	PCR64	16	0x00C0	bits[0:15]
SIUL	PCR65	16	0x00C2	bits[0:15]
SIUL	PCR66	16	0x00C4	bits[0:15]
SIUL	PCR67	16	0x00C6	bits[0:15]
SIUL	PCR68	16	0x00C8	bits[0:15]
SIUL	PCR69	16	0x00CA	bits[0:15]
SIUL	PCR70	16	0x00CC	bits[0:15]
SIUL	PCR71	16	0x00CE	bits[0:15]
SIUL	PSMI0_3	32	0x0500	bits[0:31]
SIUL	PSMI4_7	32	0x0504	bits[0:31]
SIUL	PSMI8_11	32	0x0508	bits[0:31]
SIUL	PSMI12_15	32	0x050C	bits[0:31]
SIUL	PSMI16_19	32	0x0510	bits[0:31]
SIUL	PSMI20_23	32	0x0514	bits[0:31]
SIUL	PSMI24_27	32	0x0518	bits[0:31]
SIUL	PSMI28_31	32	0x051C	bits[0:31]
SIUL	PSMI32_35	32	0x0520	bits[0:31]
SIUL	IFMC0	32	0x1000	bits[0:31]
SIUL	IFMC1	32	0x01004	bits[0:31]
SIUL	IFMC2	32	0x1008	bits[0:31]
SIUL	IFMC3	32	0x100C	bits[0:31]
SIUL	IFMC4	32	0x1010	bits[0:31]
SIUL	IFMC5	32	0x1014	bits[0:31]
SIUL	IFMC6	32	0x1018	bits[0:31]
SIUL	IFMC7	32	0x101C	bits[0:31]
SIUL	IFMC8	32	0x1020	bits[0:31]
SIUL	IFMC9	32	0x1024	bits[0:31]
SIUL	IFMC10	32	0x1028	bits[0:31]

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	IFMC11	32	0x102C	bits[0:31]
SIUL	IFMC12	32	0x1030	bits[0:31]
SIUL	IFMC13	32	0x1034	bits[0:31]
SIUL	IFMC14	32	0x1038	bits[0:31]
SIUL	IFMC15	32	0x103C	bits[0:31]
SIUL	IFMC16	32	0x1040	bits[0:31]
SIUL	IFMC17	32	0x1044	bits[0:31]
SIUL	IFMC18	32	0x1048	bits[0:31]
SIUL	IFMC19	32	0x104C	bits[0:31]
SIUL	IFMC20	32	0x1050	bits[0:31]
SIUL	IFMC21	32	0x1054	bits[0:31]
SIUL	IFMC22	32	0x1058	bits[0:31]
SIUL	IFMC23	32	0x105C	bits[0:31]
SIUL	IFMC24	32	0x1060	bits[0:31]
SIUL	IFCP	32	0x1080	bits[0:31]
Power Management Unit—Base address: 0xC3FE_8080 1 register to protect				
PMU	VREG_CTL	32	0x0000	bits[0:31]
MC Mode Entry—Base address: 0xC3FD_C000 39 registers to protect				
MC ME	ME_ME	32	0x0008	bits[0:31]
MC ME	ME_IM	32	0x0010	bits[0:31]
MC ME	ME_TEST_MC	32	0x0024	bits[0:31]
MC ME	ME_SAFE_MC	32	0x0028	bits[0:31]
MC ME	ME_DRUN_MC	32	0x002C	bits[0:31]
MC ME	ME_RUN0_MC	32	0x0030	bits[0:31]
MC ME	ME_RUN1_MC	32	0x0034	bits[0:31]
MC ME	ME_RUN2_MC	32	0x0038	bits[0:31]
MC ME	ME_RUN3_MC	32	0x003C	bits[0:31]
MC ME	ME_HALT0_MC	32	0x0040	bits[0:31]
MC ME	ME_STOP0_MC	32	0x0048	bits[0:31]
MC ME	ME_RUN_PC0	32	0x0080	bits[0:31]
MC ME	ME_RUN_PC1	32	0x0084	bits[0:31]

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC ME	ME_RUN_PC2	32	0x0088	bits[0:31]
MC ME	ME_RUN_PC3	32	0x008C	bits[0:31]
MC ME	ME_RUN_PC4	32	0x0090	bits[0:31]
MC ME	ME_RUN_PC5	32	0x0094	bits[0:31]
MC ME	ME_RUN_PC6	32	0x0098	bits[0:31]
MC ME	ME_RUN_PC7	32	0x009C	bits[0:31]
MC ME	ME_LP_PC0	32	0x00A0	bits[0:31]
MC ME	ME_LP_PC1	32	0x00A4	bits[0:31]
MC ME	ME_LP_PC2	32	0x00A8	bits[0:31]
MC ME	ME_LP_PC3	32	0x00AC	bits[0:31]
MC ME	ME_LP_PC4	32	0x00B0	bits[0:31]
MC ME	ME_LP_PC5	32	0x00B4	bits[0:31]
MC ME	ME_LP_PC6	32	0x00B8	bits[0:31]
MC ME	ME_LP_PC7	32	0x00BC	bits[0:31]
MC ME	ME_PCTL[4]	8	0x00C4	bits[0:31]
MC ME	ME_PCTL[5]	8	0x00C5	bits[0:31]
MC ME	ME_PCTL[6]	8	0x00C6	bits[0:31]
MC ME	ME_PCTL[16]	8	0x00D0	bits[0:7]
MC ME	ME_PCTL[26]	8	0xC0DA	bits[0:7]
MC ME	ME_PCTL[32]	8	0xC0E0	bits[0:7]
MC ME	ME_PCTL[35]	8	0xC0E3	bits[0:7]
MC ME	ME_PCTL[38]	8	0xC0E6	bits[0:7]
MC ME	ME_PCTL[41]	8	0xC0E9	bits[0:7]
MC ME	ME_PCTL[48]	8	0xC0F0	bits[0:7]
MC ME	ME_PCTL[49]	8	0xC0F1	bits[0:7]
MC ME	ME_PCTL[92]	8	0xC11C	bits[0:7]
MC Clock Generation Module—Base address: 0xC3FE_0000 2 registers to protect				
MC CGM	CGM_OC_EN	8	0x0370	bits[0:7]
MC CGM	CGM_OCDS_SC	8	0x0374	bits[0:7]
XOSC—Base address: 0xC3FE_0000 1 register to protect				
XOSC	OSC_CTL	32	0x0000	bits[0:31]

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
IRC_OSC—Base address: 0xC3FE_0060 1 register to protect				
IRC_OSC	RC_CTL	32	0x0000	bits[0:31]
FM PLL 0—Base address: 0xC3FE_00A0 2 registers to protect				
FMPLL 0	CR	32	0x0000	bits[0:31]
FMPLL 0	MR	32	0x0004	bits[0:31]
CMU 0—Base address: 0xC3FE_0100 1 register to protect				
CMU 0	CMU_CSR	32	0x0000	bits[0:31]
MC Reset Generation Module—Base address: 0xC3FE_4000 5 registers to protect				
MC RGM	RGM_FERD	16	0x0004	bits[0:15]
MC RGM	RGM_DERD	16	0x0006	bits[0:15]
MC RGM	RGM_FEAR	16	0x0010	bits[0:15]
MC RGM	RGM_FESS	16	0x0018	bits[0:15]
MC RGM	RGM_FBRE	16	0x001C	bits[0:15]
MCPower Control Unit—Base address: 0xC3FE_8000 1 register to protect				
MC PCU	PCONF2	32	0x0008	bits[0:31]
PIT_RTI—Base address: 0xC3FF_0000 9 registers to protect				
PIT_RTI	PIT_RTI_PITMCR	32	0x0000	32-bit
PIT_RTI	PIT_RTI_LDVAL0	32	0x0100	32-bit
PIT_RTI	PIT_RTI_TCTRL0	32	0x0108	32-bit
PIT_RTI	PIT_RTI_LDVAL1	32	0x0110	32-bit
PIT_RTI	PIT_RTI_TCTRL1	32	0x0118	32-bit
PIT_RTI	PIT_RTI_LDVAL2	32	0x0120	32-bit
PIT_RTI	PIT_RTI_TCTRL2	32	0x0128	32-bit
PIT_RTI	PIT_RTI_LDVAL3	32	0x0130	32-bit
PIT_RTI	PIT_RTI_TCTRL3	32	0x0138	32-bit
ADC 0—Base address: 0xFFE0_0000 10 registers to protect				
ADC 0	CLR0	32	0x0000	32-bit

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
ADC 0	CLR1	32	0x0004	32-bit
ADC 0	CLR2	32	0x0008	32-bit
ADC 0	CLR3	32	0x000C	32-bit
ADC 0	CLR4	32	0x0010	32-bit
ADC 0	TRC0	32	0x0034	32-bit
ADC 0	TRC1	32	0x0038	32-bit
ADC 0	TRC2	32	0x003C	32-bit
ADC 0	TRC3	32	0x0040	32-bit
ADC 0	PREREG	32	0x00A8	32-bit
eTimer 0—Base address: 0xFFE1_8000 24 registers to protect				
eTimer 0	CH0_CTRL	16	0x000E	16-bit
eTimer 0	CH0_CTRL2	16	0x0010	16-bit
eTimer 0	CH0_CTRL3	16	0x0012	16-bit
eTimer 0	CH0_CCCTRL	16	0x001C	16-bit
eTimer 0	CH1_CTRL	16	0x002E	16-bit
eTimer 0	CH1_CTRL2	16	0x0030	16-bit
eTimer 0	CH1_CTRL3	16	0x0032	16-bit
eTimer 0	CH1_CCCTRL	16	0x003C	16-bit
eTimer 0	CH2_CTRL	16	0x004E	16-bit
eTimer 0	CH2_CTRL2	16	0x0050	16-bit
eTimer 0	CH2_CTRL3	16	0x0052	16-bit
eTimer 0	CH2_CCCTRL	16	0x005C	16-bit
eTimer 0	CH3_CTRL	16	0x006E	16-bit
eTimer 0	CH3_CTRL2	16	0x0070	16-bit
eTimer 0	CH3_CTRL3	16	0x0072	16-bit
eTimer 0	CH3_CCCTRL	16	0x007C	16-bit
eTimer 0	CH4_CTRL	16	0x008E	16-bit
eTimer 0	CH4_CTRL2	16	0x0090	16-bit
eTimer 0	CH4_CTRL3	16	0x0092	16-bit
eTimer 0	CH4_CCCTRL	16	0x009C	16-bit
eTimer 0	CH5_CTRL	16	0x00AE	16-bit
eTimer 0	CH5_CTRL2	16	0x00B0	16-bit

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
eTimer 0	CH5_CTRL3	16	0x00B2	16-bit
eTimer 0	CH5_CCCTRL	16	0x00BC	16-bit
FlexPWM—Base address: 0xFFE2_4000 41 registers to protect				
FlexPWM	SUB0_CTRL2	16	0x0004	16-bit
FlexPWM	SUB0_CTRL	16	0x0006	16-bit
FlexPWM	SUB0_OCTRL	16	0x0018	16-bit
FlexPWM	SUB0_INTEN	16	0x001C	16-bit
FlexPWM	SUB0_DMAEN	16	0x001E	16-bit
FlexPWM	SUB0_TCTRL	16	0x0020	16-bit
FlexPWM	SUB0_DISMAP	16	0x0022	16-bit
FlexPWM	SUB0_DTCNT0	16	0x0024	16-bit
FlexPWM	SUB0_DTCNT1	16	0x0026	16-bit
FlexPWM	SUB1_CTRL2	16	0x0054	16-bit
FlexPWM	SUB1_CTRL	16	0x0056	16-bit
FlexPWM	SUB1_OCTRL	16	0x0068	16-bit
FlexPWM	SUB1_INTEN	16	0x006C	16-bit
FlexPWM	SUB1_DMAEN	16	0x006E	16-bit
FlexPWM	SUB1_TCTRL	16	0x0070	16-bit
FlexPWM	SUB1_DISMAP	16	0x0072	16-bit
FlexPWM	SUB1_DTCNT0	16	0x0074	16-bit
FlexPWM	SUB1_DTCNT1	16	0x0076	16-bit
FlexPWM	SUB2_CTRL2	16	0x00A4	16-bit
FlexPWM	SUB2_CTRL	16	0x00A6	16-bit
FlexPWM	SUB2_OCTRL	16	0x00B8	16-bit
FlexPWM	SUB2_INTEN	16	0x00BC	16-bit
FlexPWM	SUB2_DMAEN	16	0x00BE	16-bit
FlexPWM	SUB2_TCTRL	16	0x00C0	16-bit
FlexPWM	SUB2_DISMAP	16	0x00C2	16-bit
FlexPWM	SUB2_DTCNT0	16	0x00C4	16-bit
FlexPWM	SUB2_DTCNT1	16	0x00C6	16-bit
FlexPWM	SUB3_CTRL2	16	0x00F4	16-bit
FlexPWM	SUB3_CTRL	16	0x00F6	16-bit

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
FlexPWM	SUB3_OCTRL	16	0x0108	16-bit
FlexPWM	SUB3_INTEN	16	0x010C	16-bit
FlexPWM	SUB3_DMAEN	16	0x010E	16-bit
FlexPWM	SUB3_TCTRL	16	0x0110	16-bit
FlexPWM	SUB3_DISMAP	16	0x0112	16-bit
FlexPWM	SUB3_DTCNT0	16	0x0114	16-bit
FlexPWM	SUB3_DTCNT1	16	0x0116	16-bit
FlexPWM	MASK	16	0x0142	16-bit
FlexPWM	SWCOUT	16	0x0144	16-bit
FlexPWM	DTSRCSEL	16	0x0146	16-bit
FlexPWM	MCTRL	16	0x0148	16-bit
FlexPWM	FCTRL	16	0x014C	16-bit
DSPI 0—Base address: 0xFFF9_0000 11 registers to protect				
DSPI 0	DSPI_MCR	32	0x0000	32-bit
DSPI 0	DSPI_TCR	32	0x0008	32-bit
DSPI 0	DSPI_CTAR0	32	0x000C	32-bit
DSPI 0	DSPI_CTAR1	32	0x0010	32-bit
DSPI 0	DSPI_CTAR2	32	0x0014	32-bit
DSPI 0	DSPI_CTAR3	32	0x0018	32-bit
DSPI 0	DSPI_CTAR4	32	0x001C	32-bit
DSPI 0	DSPI_CTAR5	32	0x0020	32-bit
DSPI 0	DSPI_CTAR6	32	0x0024	32-bit
DSPI 0	DSPI_CTAR7	32	0x0028	32-bit
DSPI 0	DSPI_RSER	32	0x0030	32-bit
DSPI 1—Base address: 0xFFF9_4000 11 registers to protect				
DSPI 1	DSPI_MCR	32	0x0000	32-bit
DSPI 1	DSPI_TCR	32	0x0008	32-bit
DSPI 1	DSPI_CTAR0	32	0x000C	32-bit
DSPI 1	DSPI_CTAR1	32	0x0010	32-bit
DSPI 1	DSPI_CTAR2	32	0x0014	32-bit
DSPI 1	DSPI_CTAR3	32	0x0018	32-bit

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
DSPI 1	DSPI_CTAR4	32	0x001C	32-bit
DSPI 1	DSPI_CTAR5	32	0x0020	32-bit
DSPI 1	DSPI_CTAR6	32	0x0024	32-bit
DSPI 1	DSPI_CTAR7	32	0x0028	32-bit
DSPI 1	DSPI_RSER	32	0x0030	32-bit
DSPI 2—Base address: 0xFFF9_8000 11 registers to protect				
DSPI 2	DSPI_MCR	32	0x0000	32-bit
DSPI 2	DSPI_TCR	32	0x0008	32-bit
DSPI 2	DSPI_CTAR0	32	0x000C	32-bit
DSPI 2	DSPI_CTAR1	32	0x0010	32-bit
DSPI 2	DSPI_CTAR2	32	0x0014	32-bit
DSPI 2	DSPI_CTAR3	32	0x0018	32-bit
DSPI 2	DSPI_CTAR4	32	0x001C	32-bit
DSPI 2	DSPI_CTAR5	32	0x0020	32-bit
DSPI 2	DSPI_CTAR6	32	0x0024	32-bit
DSPI 2	DSPI_CTAR7	32	0x0028	32-bit
DSPI 2	DSPI_RSER	32	0x0030	32-bit
FlexCAN—Base address: 0xFFFC_0000 7 registers to protect				
FlexCAN	CANx_MCR	32	0x0000	32-bit
FlexCAN	CANx_CTRL	32	0x0004	32-bit
FlexCAN	CANx_RXGMASK	32	0x0010	32-bit
FlexCAN	CANx_RX14MASK	32	0x0014	32-bit
FlexCAN	CANx_RX15MASK	32	0x0018	32-bit
FlexCAN	CANx_IMASK2	32	0x0024	32-bit
FlexCAN	CANx_IMASK	32	0x0028	32-bit
Safety Port—Base address: 0xFFFE_8000 7 registers to protect				
Safety port	CANx_MCR	32	0x0000	32-bit
Safety port	CANx_CTRL	32	0x0004	32-bit
Safety port	CANx_RXGMASK	32	0x0010	32-bit
Safety port	CANx_RX14MASK	32	0x0014	32-bit

Table 475. Registers under protection (continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
Safety port	CANx_RX15MASK	32	0x0018	32-bit
Safety port	CANx_IMASK2	32	0x0024	32-bit
Safety port	CANx_IMASK	32	0x0028	32-bit

Document revision history

Table 476 summarizes revisions to this document.

Table 476. Revision history

Date	Revision	Changes
10-Dec-2009	1	Initial release
30-Jun-2010	2	Preface Organization: – Corrected cross-reference to Nexus chapter – Removed bullet “Appendix B, “Memory Map” Register figure conventions: Minor editorial correction Table ii, “Acronyms and Abbreviated Terms,” on page 21: Removed entries not used in this reference manual *Chapter 1, “Overview* *Section 1.2, “Chip*: Minor formatting change Updated *Section 1.4, “Features* Updated SPC560P40/34 block diagram *Section 1.6, “Chip-level features*: Editorial updates *Chapter 3, “Signal Description* *Section 2.3, “64-pin LQFP pinout*: Removed sentence introducing figures *Section 3.4.3, “Pin muxing*: Minor editorial change *4, “Clock Description* Reformatted SPC560P40/34 system clock generation Updated SPC560P40/34 system clock distribution Part B CR field descriptions: Corrected field name: was NDIVNDIV; is NDIV *6, “Mode Entry Module (MC_ME)* *Section 6.4.4, “Protection of Mode Configuration Registers*: Added bullet “The 4 MHz crystal oscillator must be on if the system PLL is on. Therefore, when writing a ‘1’ to PLL0ON, a ‘1’ must also be written to XOSC0ON.” *7, “Power Control Unit (MC_PCU)*: Unchanged from previous revision *8, “Reset Generation Module (MC_RGM)*: Unchanged from previous revision *9, “Interrupt Controller (INTC)* Removed MC_ prefixes: – was MC_ME; is ME – was MC_RGM; is RGM *Section 9.6, “Functional description*: Replaced “INTC_PSR211” with “INTC_PSR221” in note *10, “System Status and Configuration Module (SSCM)*: Unchanged from previous revision *11, “System Integration Unit Lite (SIUL)* *Section , “External interrupt request input pins (EIRQ[0:24])*: Replaced SIU_ with SIUL_ MCU ID Register #1 (MIDR1): Updated reset value for field PKG[4:0] in bitmap

Table 476. Revision history (continued)

Date	Revision	Changes
30-Jun-2010	2	*12, "e200z0 and e200z0h Core* *Section 12.2, "Features*: Removed bullet "Power saving modes: doze, nap, sleep, and wait" *13, "Peripheral Bridge (PBRIDGE)*: Unchanged from previous revision *14, "Crossbar Switch (XBAR)*: Unchanged from previous revision *15, "Error Correction Status Module (ECSM)* Replaced occurrences of "AXBS_lite" and "AXBS" with "XBAR" *Section 15.4.3, "ECSM_reg_protection*: Replaced AIPS with PBRIDGE *16, "Internal Static RAM (SRAM)* Editorial and formatting changes SRAM memory map: Replaced SRA with "—" in register description column *17, "Flash Memory* *Section 17.2.6, "Basic interface protocol*: Specified that haddr represents the Flash address *Section 17.2.17, "Wait state emulation*: Specified that haddr represents the Flash address *18, "Enhanced Direct Memory Access (eDMA)* Minor formatting changes *19, "DMA Channel Mux (DMA_MUX)* Updated DMA channel mapping *20, "Deserial Serial Peripheral Interface (DSPI)* Minor formatting changes DSPI memory map: Updated reset values DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)—Changed reset value of field FMSZ[3:0] from '0000' to '1111' *21, "LIN Controller (LINFlex)* Minor formatting changes Updated *Table 232., "Error calculation for programmed baud rates* (baud rate 10417) and LFDIV note above table *22, "FlexCAN* Minor editorial changes; minor formatting changes to bitmap reset value rows *Section 22.3.1, "FlexCAN memory mapping*: Updated address offset ranges and RAM sizes *Section , "Module Configuration Register (MCR)*: Changed DOZE field to reserved bit Module Configuration Register (MCR): Changed reset value for field MAXMB[5:0] from 000000 to 001111 *Section , "Rx Individual Mask Registers (RXIMR0–RXIMR31)*: Replaced "RXIM63" with "RXIMR31" in bitmap and field description titles *Section , "Freeze mode*: Deleted "(Disable, Doze, Stop)" in first paragraph *Section 22.4.11, "Bus interface*: Updated address offset ranges

Table 476. Revision history (continued)

Date	Revision	Changes
30-Jun-2010	2	*23, Analog-to-Digital Converter (ADC)* ADC digital registers: Removed Channel Pending Registers (CEOCFR[x]) and Decode Signals Delay Register (DSDR) *Section 23.3.3, ADC sampling and conversion timing*: Corrected instances of bitfield name INPSAMPLE to INPSAMP *Section 23.3.7, Interrupts*: Removed content concerning register CEOCFR *24, "Cross Triggering Unit (CTU)* *Section 24.4.1, "ADC commands list*: Minor editorial change *Section 24.8.13, "FIFO threshold register (FTH)*: Amended and corrected register name *25, "FlexPWM* Editorial and formatting changes "Capture Value 0 Cycle register (CVAL0CYC)" and "Capture Value 1 Cycle register (CVAL1CYC)" use only bits [13:15] instead of bits[12:15] Replaced instances of WAIT mode with WAIT/HALT mode Changed FSTS register reset value from 0x0000 to 0x0303 *26, "eTimer*: Unchanged from previous revision *27, "Functional Safety*: Unchanged from previous revision *28, "Fault Collection Unit (FCU)* *Section , "Test mode*: Removed sentence referencing software-triggered faults not being supported by the FCU_FFGR Register summary: – In FCU_FFR, changed field SRF1 to read-only with value 0 – In FCU_FFFR, changed field FRSRF1 to read-only with value 0 *Section , "Fault Flag Register (FCU_FFR)*: Removed sentence referencing clearing the software fault flag SRF1; changed field SRF1 to read-only with value 0 Hardware/software fault description: Marked SRF1 as "Not used" *Section , "Frozen Fault Flag Register (FCU_FFFR)*: Changed field FRSRF1 to read-only with value 0 FCU_FER field descriptions: Added note that field ESF1 not implemented FCU_TER field descriptions: Added note that field TESF1 not implemented *Section , "Microcontroller State Register (FCU_MCSR)*: Updated bit values *Section , "Frozen MC State Register (FCU_FMCSR)*: Updated bit values *29, "Wakeup Unit (WKPU)*: Unchanged from previous revision *30, "Periodic Interrupt Timer (PIT)* *Section 30.3, "Memory map and registers description*: Minor formatting changes throughout *Section , "PIT Module Control Register (PITMCR)*: Added field MDIS (bit 30) *31, "System Timer Module (STM)*: Unchanged from previous revision *32, "Cyclic Redundancy Check (CRC)* CRC computation flow: Replaced CRC_CNTX_NUM with "n" Improved readability of DMA-CRC Transmission Sequence and of DMA-CRC Reception Sequence

Table 476. Revision history (continued)

Date	Revision	Changes
30-Jun-2010	2	*33, "Boot Assist Module (BAM)* Minor editorial and formatting changes *Section 33.3, "Boot modes*: Minor editorial changes *Section 33.5.1, "Entering boot modes*: Editorial changes Boot mode selection: Added Autobaud Scan boot mode *Section 33.5.3, "Reset Configuration Half Word (RCHW)*: Removed the word "Source" from title *Section , "UART boot mode download protocol*: Modified title; editorial changes *Section 33.6, "FlexCAN boot mode download protocol*: Modified title; editorial changes in several subsections *34, "Voltage Regulators and Power Supplies* *Section 34.1, "Voltage regulator*: Replaced "The internal voltage regulator requires an external capacitance (CREG)" with "The internal voltage regulator requires an external ballast transistor and (external) capacitance (CREG)" *Section 34.1.2, "Low Voltage Detectors (LVD) and Power On Reset (POR)*: Removed sentence "The other LVD_DIG is placed in the standby domain and senses the standby 1.2 V supply level notifying that the 1.2 V output is stable." Voltage Regulator Control register (VREG_CTL): Changed reset values of register bits 23 and 27 Voltage Regulator Status register (VREG_STATUS): Changed reset values of register bits 23 and 27 *Section 34.2, "Power supply strategy*: – Updated description of bullet "LV—Low voltage internal power supply" – Removed bullet "BV—High voltage supply for voltage regulator ballast" and associated description *35, "IEEE 1149.1 Test Access Port Controller (JTAGC)*: Unchanged from previous revision *36, "Nexus Development Interface (NDI)* Minor editorial and formatting changes Replaced several references to "Zen" with "e200z0h" throughout chapter Replaced several references to PowerPC with references to Power Architecture™ *Section 36.2.1, "Features not supported*: Replaced "Nexux3" with "Nexus3" IEEE 1149.1-2001 TAP Controller State Machine: Added figure title and inserted note *Appendix A, "Registers Under Protection*: Unchanged from previous revision

Table 476. Revision history (continued)

Date	Revision	Changes
11-May-2011	3	“Preface” chapter, entirely rewrote “SPC560P40/34 block diagram”, made arrow going from peripheral bridge to crossbar switch bidirectional “Clock Description” chapter – In the “Functional description” section, replaced all occurences of XTALOUT with EXTAL. – Removed “Nexus Message Clock (MCKO)” and “Software controlled power management/clock gating”. “Interrupt controller” chapter: – Added back MC_ prefixes: — was ME; is MC_ME — was RGM; is MC_RGM – In the “Interrupt vector table”, the row 63 was “reserved”; is ADC_ER. – In the “INTC memory map” table, removed access and reset columns. “System Integration Unit Lite (SIUL)” chapter: In the "MCU ID Register # (MIDR2)" section: replaced reset value of EE:x is now 0 renamed the bit field "FR" with "FF" In the "MIDR2 field description" table, renamed the bit field "FR" with "FF" and replaced its description with "N/A" In the "MCU ID Register #2 (MIDR2)" section: – replaced reset value of EE:x is now 0 – shown both FLASH_SIZE_2 and FF fields as reserved – renamed FLASH_SIZE_1 to FLASH_SIZE – changed the description of FLASH_SIZE_1: was “0011: 128 KB”, is now “0011: 192 KB” “e200z0 and e200z0h Core” chapter: In the “Core registers and programmer’s model” section, removed note concerning the register numbering. “Error Correction Status Module (ECSM)” chapter Replaced all ocurences of “AXBS” with “XBAR” In the “Spp_Ips_Reg_Protection block diagram” figure, replaced “AIPS_LITE” with “PBRIDGE” “Flash Memory” chapter In the “Memory map” section, added a “caution” and a “note” concerning the register management. “DMA Channel Mux (DMA_MUX)” chapter In the “DMA_MUX memory map” table, removed access and reset columns. In the “DMA mux channel 4–15 block diagram” figure, added “Always #1” and “Always #9” arrow labels

Table 476. Revision history (continued)

Date	Revision	Changes
11-May-2011	3 cont'd	"Deserial Serial Peripheral Interface (DSPI)" chapter In the "DSPI memory map" table, removed access and reset columns. In the "DSPI block diagram" figure, replace arrow labels from "1" to "3". In the DSPIx_MCR.CONT_SCKE filed description, added a note. In the "Continuous Serial Communications clock" section, added a note. In the "Continuous Selection Format" section, added a note. "LIN Controller (LINFlex)" chapter In the "IFER field descriptions" table, switched "activated" and "deactivated" in order to match with "IFER[FACT] configuration" table. In the "UART mode" section, in the "9-bit frames" subsection, changed "sum of the 7 data bits" to "sum of the 8 data bits". "FLexCAN" chapter In the "Module Configuration Register (MCR)" section, changed DOZE field to reserved bit In the "Freeze mode", replaced "(Disable, Doze, Stop)" with "(Halt, Stop)" in first paragraph In the "Module Configuration Register (MCR)" figure, changed reset value for field MAXMB[5:0] from 000000 to 001111 In the "Freeze mode" section: Deleted "(Halt, Stop)" in first paragraph In the "FlexCAN module memory map" table, removed access and reset columns. In the "Message Buffer lock mechanism" section, updated a footnote "Analog-to-Digital Converter (ADC)" chapter: In the "Conversion timing registers CTR" section, removed footnote. in the Device-specific features section, removed "internal standard channels" from "Sampling and conversion time register" bullet In the "Mask registers" section changed the number of channel from 96 to 16. In the "Channel Data Register (CDR)" section changed the number of channel from 95 to 15 "FlexPWM" chapter "Capture Value 0 Cycle register (CVAL0CYC)" and "Capture Value 1 Cycle register (CVAL1CYC)" use only bits [13:15] instead of bits[12:15] Replaced instances of WAIT mode with WAIT/HALT mode Device-specific changes: DMAEN field descriptions: Removed reference to bits CAxDE and CBxDE from FAND field description "Functional Safety" chapter: In the " Register protection memory map" table, removed access and reset columns. In the "SWT memory map" table, removed access and reset columns per new agreement. In the "SWT Control Register (SWT_CR)" table, changed the reset value from 0x4000_011B to 0xFF00_011B. In the "SWT memory map" table, inserted the SWT_1 offset value

Table 476. Revision history (continued)

Date	Revision	Changes
11-May-2011	3 cont'd	"Nexus Development Interface (NDI)" chapter: Minor editorial and formatting changes Throught the chapter, removed all DOZE occurrences. Replaced several references to Power Architecture™ with references to Power Architecture In the "OnCE Register Access Requirements" table, removed "Notes" column. In the "Hardware Implementation Dependent Register 0 (HID0)" register, made the DOZE fiel as reserved. In the "NDI functional block diagram" figure, removed ownership trace and watchpoint trace blocks. In the "e200z0h OnCE Register Addressing" table, replaced "Nexus2/3-Access" and "LSRL Select (see Test Specification)"with (Reserved). In the "OnCE Register Access Requirements" table, removed both "Nexus2/3-Access" and "LSRL Select " rows. In the "Nexus client JTAG instructions" table, removed "NEXUS2_ACCESS" row. Removed "Configuring the NDI for Nexus messaging" section.

Table 476. Revision history (continued)

Date	Revision	Changes
11-May-2011	3 cont'd	"Fault Collection and Control Unit (FCCU)" chapter: Changed the chapter title in "Fault Collection and Control Unit (FCCU)" instead of "Fault Collection Unit (FCU)". In the "FCCU memory map" table, removed access and reset columns per new agreement. "Test mode" section: Removed sentence referencing software-triggered faults not being supported by the FCU_FFGR Register summary: In FCU_FFR, changed field SRF1 to read-only with value 0 In FCU_FFFR, changed field FRSRF1 to read-only with value 0 "Fault Flag Register (FCCU_FFR)" section: Removed sentence referencing clearing the software fault flag SRF1; changed field SRF1 to read-only with value 0 Hardware/software fault description: Marked SRF1 as "Not used" "Frozen Fault Flag Register (FCCU_FFFR)" section: Changed field FRSRF1 to read-only with value 0 FCCU_FER field descriptions: Added note that field ESF1 not implemented FCCU_TER field descriptions: Added note that field TESF1 not implemented "Microcontroller State Register (FCCU_MCSR)" section: Updated bit values "Frozen MC State Register (FCCU_FMCSR)" section: Updated bit values SRF1: was "not used", is now "FCU Software-triggered error". Switched the module of SFR2 with that of SFR4. Updated all registers according the "Hardware/software fault description" table. "Wakeup Unit (WKPU)" chapter In the "External Signal description" section, added note about Wakeup pin termination. "Boot Assist Module (BAM)" chapter: "Boot modes": Minor editorial changes Boot mode selection: Added Autobaud Scan boot mode "Reset Configuration Half Word (RCHW)" section: Removed the word "Source" from title System clock frequency related to external clock frequency: Minor editorial changes In the "Hardware configuration to select boot mode" table: rewrote the column title from "ABS[1:0]" to ABS[2,0] added footnote stating: ABS[1] is "don't care" Changed the "Boot Mode" description for FAB=1 and ABS[2,0]=10 "Voltage Regulators and Power Supplies" chapter In the "Power supply strategy" section: - Removed the BV domain - Update the "supply domains" "IEEE 1149.1 Test Access Port Controller (JTAGC)" chapter: In the "TAP sharing mode" section: - Removed "ACCESS_AUX_TAP_eTPU". - Replaced ACCESS_AUX_TAP_DMA with ACCESS_AUX_TAP_NPC.

Table 476. Revision history (continued)

Date	Revision	Changes
12-Mar-2012	4	*Chapter 2: SPC560P40/34 memory map* *Table 3 (Memory map)*: Changed “Data Flash Array 0 Test Sector” size from 16K to 8K *Chapter 3: Signal Description*: In the *Table 6 (Pin muxing)*:removed Port E[0] *Chapter 4: Clock Description*: *Table 9 (Crystal oscillator truth table)*: removed in normal mode with XOSC enabled the configuration of oscillator providing external clock. *Section 4.6, IRC 16 MHz internal RC oscillator (RC_CTL)*: reworded register description *Chapter 6: Mode Entry Module (MC_ME)*: *Table 37 (MC_ME Register Description)*: changed ME_PCTL26 description from “PERIPH26 Control“ to “SafetyPort Control“ *Table 38 (MC_ME Memory Map)*: ME_PS0 register, changed bit name from “S_PERIPH26” to “S_SafetyPort” *Figure 57 (Peripheral Status Register 0 (ME_PS0))*: changed bit name from “S_PERIPH26” to “S_SafetyPort” *Chapter 11: System Integration Unit Lite (SIUL)*: Added new *Table 100 (PCR bit implementation by pad type)*. *Chapter 17: Flash Memory*: *Figure 175 (Non-Volatile User Options register (NVUSRO))*: removed OSCILLATOR_MARGIN bit. *Table 171 (NVUSRO field descriptions)*: removed OSCILLATOR_MARGIN field and updated UOx bit fields *Figure 151 (Data Flash module structure)*: Changed “Test Sector” size from 16K to 8K *Table 137 (Flash-related regions in the system memory map)*: Changed “Data Flash Array 0 Test Sector” size from 16K to 8K *Table 143 (64 KB data Flash module sectorization)*: changed reserved area from 0x0081_0000 to 0x00C0_FFFF to 0x0081_0000 to 0x00C0_1FFF *Table 144 (TestFlash structure)*: Changed reserved area for “Code TestFlash” and “Data TestFlash” *Section 17.2.4, Memory map and registers description*, added a note. *Table 146 (Flash registers)*: added note for UT2, UMISR2, UMISR3 and UMISR4 registers *Table 148 (Flash 64 KB bank1 register map)*: replaced UT2, UMISR2, UMISR3 and UMISR4 registers with reserved space in the *Section 17.3.2, Main features*: Removed “One Time Programmable (OTP) area in TestFlash block" Added note in *Section , User Test 2 register (UT2)*, *Section , User Multiple Input Signature Register 2 (UMISR2)*, *Section , User Multiple Input Signature Register 3 (UMISR3)* and *Section , User Multiple Input Signature Register 4 (UMISR4)* Renamed “Programming considerations“ section with *Section 17.3.8, Code Flash programming considerations*

Table 476. Revision history (continued)

Date	Revision	Changes
12-Mar-2012	4 cont'd	*Chapter 21: LIN Controller (LINFlex)* *Figure 237 (LIN status register (LINSR))*: changed LINS access from read/write to write only *Figure 242 (LIN output compare register (LINOCR))*: changed note from LINTCSR[LTOM] = 1 to LINTCSR[LTOM] = 0 Updated *Section , Identifier filter enable register (IFER)*: *Figure 251 (Identifier filter enable register (IFER))*: changed FACT field from 8 bit to 16 bit Updated *Table 254 (IFER field descriptions)* Removed "IFER[FACT] configuration" table *Figure 254 (Identifier filter control register (IFCR2n))*: changed ID access from read/write "w1c' to read/write only in initialization mode *Figure 255 (Identifier filter control register (IFCR2n + 1))*: changed ID access from read/write "w1c' to read/write only in initialization mode Added *Section , Overrun* *Section , Data transmission (transceiver as publisher)*: changed BDAR register with BDR register Reworded *Section , Overrun* *Section , Filter mode*: changed sentence "eight IFCR registers" with "sixteen IFCR registers" *Section , Identifier filter mode configuration*: changed sentence "the filter must first be deactivated by programming IFER[FACT] = 0" with "the filter must first be activated by programming IFER[FACT] = 1" *Section , Automatic resynchronization method* now is a section *Section , LIN timeout mode*: changed sentence "Setting the LTOM bit" with "Resetting the LTOM bit" *Section , Output compare mode*: changed sentence "Programming LINTCSR[LTOM] = 0 enables the output compare mode" with "Programming LINTCSR[LTOM] = 1 enables the output compare mode" *Chapter 23: Analog-to-Digital Converter (ADC)*: Renamed section "Analog watchdog pulse width modulation bus" with *Section , Analog watchdog functionality*, and reworded the section *Section 23.3.7, Interrupts*: removed sentence "Interrupts can be individually enabled on a channel by channel basis by programming the CIMR (Channel Interrupt Mask Register)." *Section 23.3.8, Power-down mode*: replaced sentence: "If the CTU is enabled and the CSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set." with "If the CTU is enabled and the MSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set." *Table 294 (ADC digital registers)*: removed CIMR0 register, set address as reserved. Removed "Channel Interrupt Mask Register (CIMR[0])" section *Chapter 24: Cross Triggering Unit (CTU)*: *Figure 309 (Trigger Generator Sub-unit Input Selection Register (TGSISR))*: converted fields I14_FE and I14_RE to 'Reserved' and implemented fields I4_FE and I4_RE *Figure 316 (Trigger handler control register 1 (THCR1))*: changed access type from read only to read/write *Chapter 26: eTimer*: *Section , Comparator Load register 1 (CMPLD1)*: Modified the description of CMPLD field to read, "Specifies the preload value for the COMP1 register" instead of "Specifies the preload value for the COMP2 register".

Table 476. Revision history (continued)

Date	Revision	Changes
12-Mar-2012	4 cont'd	*Chapter 27: Functional Safety*: *Figure 433 (SWT Counter Output register (SWT_CO))*: changed the access permission from read/write to read only *Table 394 (SWT_TO field descriptions)*: updated fied description, was (SWT_CR.WENSWT_CR.=0) is (SWT_CR[WEN]=0). *Chapter 33: Boot Assist Module (BAM)* *Table 435 (Hardware configuration to select boot mode)*: rewrote the column title from "ABS[2,0]" to ABS[1:0] removed footnote stating: ABS[1] is "don't care" *Section 33.5.1, Entering boot modes* : added PAD A[2] note, added the Boot Configuration Pins. *Table 435 (Hardware configuration to select boot mode)* : Added footnote, "During reset the boot configuration pins are weak pull down." Added *Section 33.5.2, SPC560P40/34 boot pins*. Updated *Section , Configuration* with new *Figure 487 (BAM Autoscan code flow)*. *Section , Boot from UART with autobaud enabled* : Updated "UART boot mode" with "FlexCAN boot mode". Added *Section 33.7, Censorship* *Chapter 34: Voltage Regulators and Power Supplies*: Updated *Section 34.1.1, High Power or Main Regulator (HPREG)* *Chapter 35: IEEE 1149.1 Test Access Port Controller (JTAGC)*: Updated *Section 35.9, e200z0 OnCE controller* to remove reference of Nexus2+ configuration registers. Updated *Table 457 (e200z0 OnCE register addressing)*: Nexus 2+ Access was replaced with "Reserved". *Chapter 36: Nexus Development Interface (NDI)*: Removed the following sections : - All sections between *Section 36.8, "Interrupts and Exceptions* and *Section 36.9, "Debug support overview*. - Section 36.26.2, "Nexus Messaging". - Section 36.8.1, "Hardware Implementation Dependent Register 0 (HID0)". *Section 36.4, "Features*: Removed the following features they are not supported by Nexus Class 1: - Program Trace - Ownershipe Trace - Watchpoint messaging - Watchpoint trigger - Registers for program trace, ownership trace, and watchpoint trigger - Run-time access to the on-chip memory map Updated *Figure 504, NDI functional block diagram*. *Section 36.9.1, Software Debug Facilities*: removed cross-reference at "Debug Interrupt (IVOR15)" section.
17-Sep-2013	5	Updated Disclaimer