Freescale Semiconductor Inc.
Reference Manual

Document Number: MCF52235RM
Rev. 7.1, 04/2015

MCF52235 ColdFire Integrated Microcontroller Reference Manual with Addendum

Rev. 7.1 of the MCF52235 ColdFire Integrated Microcontroller Reference Manual has two parts:

- The addendum to revision 7 of the reference manual, immediately following this cover page.
- Revision 7 of the reference manual, following the addendum. The changes described in the addendum have not been implemented in the specified pages.

Freescale Semiconductor Inc.
Reference Manual Addendum

Document Number: MCF52235RMAD
Rev. 0, 04/2015

Addendum to Rev. 7 of the MCF52235 ColdFire Integrated Microcontroller Reference Manual

This addendum identifies changes to Revision 7 of the MCF52235 ColdFire Integrated Microcontroller Reference Manual. The changes described in the addendum have not been implemented in the specified pages.

1 Interrupt source assignments for Interrupt Controllers 0 and 1

Location	Section 15.3.8.1, Interrupt sources

Replace the content of section 15.3.8.1 with the following text and Tables 15-16 and 15-17, which were present in Rev. 6 of the reference manual but missing in Rev. 7:

Table 15-16 and Table 15-17 list the interrupt source assignments for Interrupt Controllers 0 and 1.

Table 15-16. Kirin2e Interrupt Source Assignment For Interrupt Controller 0

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0	Not used (Reserved)			
1	EPORT	EPF1	Edge port flag 1	Write EPF1 = 1
2		EPF2	Edge port flag 2	Write EPF2 = 1
3		EPF3	Edge port flag 3	Write EPF3 = 1
4		EPF4	Edge port flag 4	Write EPF4 = 1
5		EPF5	Edge port flag 5	Write EPF5 = 1
6		EPF6	Edge port flag 6	Write EPF6 = 1
7		EPF7	Edge port flag 7	Write EPF7 = 1
8	SCM	SWTI	Software watchdog timeout	Cleared when service complete.
9	DMA	DONE	DMA Channel 0 transfer complete	Write DONE = 1
10		DONE	DMA Channel 1 transfer complete	Write DONE = 1
11		DONE	DMA Channel 2 transfer complete	Write DONE = 1
12		DONE	DMA Channel 3 transfer complete	Write DONE = 1
13	UART0	INT	UART0 interrupt	Automatically cleared
14	UART1	INT	UART1 interrupt	Automatically cleared
15	UART2	INT	UART2 interrupt	Automatically cleared
16	Not used (Reserved)			
17	I^2C	IIF	I^2C interrupt	Write IIF = 0
18	QSPI	INT	QSPI interrupt	Write 1 to appropriate QIR bit
19	DTIM0	INT	DTIM0 interrupt	Write 1 to appropriate DTER0 bit
20	DTIM1	INT	DTIM1 interrupt	Write 1 to appropriate DTER1 bit
21	DTIM2	INT	DTIM2 interrupt	Write 1 to appropriate DTER2 bit
22	DTIM3	INT	DTIM3 interrupt	Write 1 to appropriate DTER3 bit

Table 15-16. Kirin2e Interrupt Source Assignment For Interrupt Controller 0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
23	FLEXCAN	BUF0I	Message Buffer 0 Interrupt	Write 1 to BUF0I after reading as 1
24		BUF1I	Message Buffer 1 Interrupt	Write 1 to BUF1I after reading as 1
25		BUF2I	Message Buffer 2 Interrupt	Write 1 to BUF2I after reading as 1
26		BUF3I	Message Buffer 3 Interrupt	Write 1 to BUF3I after reading as 1
27		BUF4I	Message Buffer 4 Interrupt	Write 1 to BUF4I after reading as 1
28		BUF5I	Message Buffer 5 Interrupt	Write 1 to BUF5I after reading as 1
29		BUF6I	Message Buffer 6 Interrupt	Write 1 to BUF6I after reading as 1
30		BUF7I	Message Buffer 7 Interrupt	Write 1 to BUF7I after reading as 1
31		BUF8I	Message Buffer 8 Interrupt	Write 1 to BUF8I after reading as 1
32		BUF9I	Message Buffer 9 Interrupt	Write 1 to BUF9I after reading as 1
33		BUF10I	Message Buffer 10 Interrupt	Write 1 to BUF10I after reading as 1
34		BUF11I	Message Buffer 11 Interrupt	Write 1 to BUF11I after reading as 1
35		BUF12I	Message Buffer 12 Interrupt	Write 1 to BUF12I after reading as 1
36		BUF13I	Message Buffer 13 Interrupt	Write 1 to BUF13I after reading as 1
37		BUF14I	Message Buffer 14 Interrupt	Write 1 to BUF14I after reading as 1
38		BUF15I	Message Buffer 15 Interrupt	Write 1 to BUF15I after reading as 1
39		ERR_INT	Error Interrupt	Read reported error bits in ESR or write 0 to ERR_INT
40		BOFF_INT	Bus-Off Interrupt	Write 0 to BOFF_INT
41	GPT	TOF	Timer overflow	Write TOF = 1 or access TIMCNTH/L if TFFCA = 1
42		PAIF	Pulse accumulator input	Write PAIF = 1 or access PAC if TFFCA = 1
43		PAOVF	Pulse accumulator overflow	Write PAOVF = 1 or access PAC if TFFCA = 1
44		C0F	Timer channel 0	Write C0F = 1 or access IC/OC if TFFCA = 1
45		C1F	Timer channel 1	Write 1 to C1F or access IC/OC if TFFCA = 1
46		C2F	Timer channel 2	Write 1 to C2F or access IC/OC if TFFCA = 1
47		C3F	Timer channel 3	Write 1 to C3F or access IC/OC if TFFCA = 1
48	PMM	LVDF	LVD	Write LVDF = 1
49	ADC	ADCA	ADCA conversion complete	Write 1 to EOSI0
50		ADCB	ADCB conversion complete	Write 1 to EOSI1
51		ADCINT	ADC Interrupt	Write 1 to ZCI, LLMTI and HLMTI
52	Not used (Reserved)			
53	Not used (Reserved)			

Table 15-16. Kirin2e Interrupt Source Assignment For Interrupt Controller 0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
54	Not used (Reserved)			
55	PIT0	PIF	PIT interrupt flag	Write PIF = 1 or write PMR
56	PIT1	PIF	PIT interrupt flag	Write PIF = 1 or write PMR
57	Not Used (Reserved)			
58	Not Used (Reserved)			
59	CFM	CBEIF	SGFM buffer empty	Write CBEIF = 1
60	CFM	CCIF	SGFM command complete	Cleared automatically
61	CFM	PVIF	Protection violation	Cleared automatically
62	CFM	AEIF	Access error	Cleared automatically
63	PWM	PWM	PWM Interrupt	Write PWMIF = 1

Table 15-17. Interrupt Source Assignment For Interrupt Controller 1

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0–7	Not Used (Reserved)			
8	FLEXCAN	BUF0I	Message Buffer 0 Interrupt	Write 1 to BUF0I after reading as 1
9		BUF1I	Message Buffer 1 Interrupt	Write 1 to BUF1I after reading as 1
10		BUF2I	Message Buffer 2 Interrupt	Write 1 to BUF2I after reading as 1
11		BUF3I	Message Buffer 3 Interrupt	Write 1 to BUF3I after reading as 1
12		BUF4I	Message Buffer 4 Interrupt	Write 1 to BUF4I after reading as 1
13		BUF5I	Message Buffer 5 Interrupt	Write 1 to BUF5I after reading as 1
14		BUF6I	Message Buffer 6 Interrupt	Write 1 to BUF6I after reading as 1
15		BUF7I	Message Buffer 7 Interrupt	Write 1 to BUF7I after reading as 1
16		BUF8I	Message Buffer 8 Interrupt	Write 1 to BUF8I after reading as 1
17		BUF9I	Message Buffer 9 Interrupt	Write 1 to BUF9I after reading as 1
18		BUF10I	Message Buffer 10 Interrupt	Write 1 to BUF10I after reading as 1
19		BUF11I	Message Buffer 11 Interrupt	Write 1 to BUF11I after reading as 1
20		BUF12I	Message Buffer 12 Interrupt	Write 1 to BUF12I after reading as 1
21		BUF13I	Message Buffer 13 Interrupt	Write 1 to BUF13I after reading as 1
22		BUF14I	Message Buffer 14 Interrupt	Write 1 to BUF14I after reading as 1
23		BUF15I	Message Buffer 15 Interrupt	Write 1 to BUF15I after reading as 1
24		ERR_INT	Error Interrupt	Read reported error bits in ESR or write 0 to ERR_INT
25		BOFF_INT	Bus-Off Interrupt	Write 0 to BOFF_INT
26–31	Not Used			
32	EPORT	EPF0	Edge port flag 0	Write EPF0 = 1
33		EPF1	Edge port flag 1	Write EPF1 = 1
34		EPF2	Edge port flag 2	Write EPF2 = 1
35		EPF3	Edge port flag 3	Write EPF3 = 1
36		EPF4	Edge port flag 4	Write EPF4 = 1
37		EPF5	Edge port flag 5	Write EPF5 = 1
38		EPF6	Edge port flag 6	Write EPF6 = 1
39		EPF7	Edge port flag 7	Write EPF7 = 1
40–63	Not Used			

MCF52235 ColdFire® Integrated Microcontroller Reference Manual

Devices Supported:
MCF52230
MCF52231
MCF52232
MCF52233
MCF52234
MCF52235
MCF52236

Document Number:
MCF52235RM
Rev. 7
03/2011

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd. Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: MCF52235RM
Rev. 7
03/2011

Chapter 1
Overview

Chapter 2
Signal Descriptions

Chapter 3
ColdFire Core

Chapter 4
Enhanced Multiply-Accumulate Unit (EMAC)

Chapter 5
Cryptographic Acceleration Unit (CAU)

Chapter 6
Random Number Generator (RNG)

Chapter 7
Clock Module

Chapter 8
Real-Time Clock

Chapter 9
Power Management

Chapter 10
Reset Controller Module

Chapter 11
Static RAM (SRAM)

Chapter 12
Chip Configuration Module (CCM)

Chapter 13
System Control Module (SCM)

Chapter 14
General Purpose I/O Module

Chapter 15
Interrupt Controller Module

Chapter 16
Edge Port Modules (EPORT*n*)

Chapter 17
ColdFire Flash Module (CFM)

Chapter 18
Fast Ethernet Controller (FEC)

Chapter 19
Ethernet Physical Transceiver (EPHY)

Chapter 20
DMA Controller Module

Chapter 21
EzPort

Chapter 22
Programmable Interrupt Timers (PIT0–PIT1)

Chapter 23
General Purpose Timer Module (GPT)

Chapter 24
DMA Timers (DTIM0–DTIM3)

Chapter 25
Queued Serial Peripheral Interface (QSPI)

Chapter 26
UART Modules

Chapter 27
I^2C Interface

Chapter 28
Analog-to-Digital Converter (ADC)

Chapter 29
Pulse-Width Modulation (PWM) Module

Chapter 30
FlexCAN

Chapter 31
Debug Module

Chapter 32
IEEE 1149.1 Test Access Port (JTAG)

Appendix A
Register Memory Map Quick Reference

Appendix B
Revision History

Chapter 1
Overview

This chapter provides an overview of the major features and functional components of the MCF52235 family of microcontrollers. The MCF52235 family is a highly integrated implementation of the ColdFire® family of reduced instruction set computing (RISC) microcontrollers that also includes the MCF52230, MCF52231, MCF52232, MC52233, MC52234, and MCF52236. The differences between these parts are summarized in Table 1-1. This document is written from the perspective of the MCF52235.

The MCF52235 represents a family of highly-integrated 32-bit microcontrollers based on the V2 ColdFire microarchitecture. Featuring up to 32 Kbytes of internal SRAM and 256 Kbytes of flash memory, four 32-bit timers with DMA request capability, a 4-channel DMA controller, fast Ethernet controller, a CAN module, an I^2C™ module, 3 UARTs and a queued SPI, the MC52235 family has been designed for general-purpose industrial control applications.

This 32-bit device is based on the Version 2 (V2) ColdFire reduced instruction set computing (RISC) core with an enhanced multiply-accumulate unit (EMAC) and divider providing 56 Dhrystone 2.1 MIPS at a frequency of up to 60 MHz from internal flash. On-chip modules include the following:

- V2 ColdFire core with enhanced multiply-accumulate unit (EMAC)
- Cryptographic Acceleration Unit (CAU)
- Up to 32 Kbytes of internal SRAM
- Up to 256 Kbytes of on-chip flash memory
- Fast Ethernet Controller (FEC) with on-chip transceiver (ePHY)
- Three universal asynchronous receiver/transmitters (UARTs)
- Controller area network 2.0B (FlexCAN) module
- Inter-integrated circuit (I^2C) bus controller
- 10- or 12-bit analog-to-digital converter (ADC)
- Queued serial peripheral interface (QSPI) module
- Four-channel, 32-bit direct memory access (DMA) controller
- Four-channel, 32-bit input capture/output compare timers with optional DMA support
- Two 16-bit periodic interrupt timers (PITs)
- Programmable software watchdog timer
- Two interrupt controllers, each capable of handling up to 63 interrupt sources (126 total)

These devices are ideal for cost-sensitive applications requiring significant control processing for connectivity, data buffering, and user interface, as well as signal processing in a variety of key markets such as security, imaging, networking, gaming, and medical. This leading package of integration and high performance allows fast time to market through easy code reuse and extensive third party tool support.

To locate any published errata or updates for this document, refer to the ColdFire products website at http://www.freescale.com/coldfire.

1.1 MCF52235 Family Configurations

Table 1-1. MCF52235 Family Configurations

Module	52230	52231	52232	52233	52234	52235	52236
Version 2 ColdFire Core with EMAC (Enhanced Multiply-Accumulate Unit)	•	•	•	•	•	•	•
System Clock (MHz)	60	60	50	60	60	60	50
Performance (Dhrystone 2.1 MIPS)	56	56	46	56	56	56	46
Flash / Static RAM (SRAM)	128/32 Kbytes	128/32 Kbytes	128/32 Kbytes	256/32 Kbytes	256/32 Kbytes	256/32 Kbytes	256/32 Kbytes
Interrupt Controllers (INTC0/INTC1)	•	•	•	•	•	•	•
Fast Analog-to-Digital Converter (ADC)	•	•	•	•	•	•	•
Random Number Generator and Crypto Acceleration Unit (CAU)	—	—	—	—	—	•	—
FlexCAN 2.0B Module	—	•	—	—	•	•	—
Fast Ethernet Controller (FEC) with on-chip interface (EPHY)	•	•	•	•	•	•	•
Four-channel Direct-Memory Access (DMA)	•	•	•	•	•	•	•
Software Watchdog Timer (WDT)	•	•	•	•	•	•	•
Programmable Interrupt Timer	2	2	2	2	2	2	2
Four-Channel General Purpose Timer	•	•	•	•	•	•	•
32-bit DMA Timers	4	4	4	4	4	4	4
QSPI	•	•	•	•	•	•	•
UART(s)	3	3	3	3	3	3	3
I^2C	•	•	•	•	•	•	•
Eight/Four-channel 8/16-bit PWM Timer	•	•	•	•	•	•	•
General Purpose I/O Module (GPIO)	•	•	•	•	•	•	•
Chip Configuration and Reset Controller Module	•	•	•	•	•	•	•
Background Debug Mode (BDM)	•	•	•	•	•	•	•
JTAG - IEEE 1149.1 Test Access Port[1]	•	•	•	•	•	•	•
Package	80 LQFP 112 LQFP	80 LQFP 112 LQFP	80 LQFP	80 LQFP 112 LQFP	112 LQFP 121 MAPBGA	112 LQFP 121 MAPBGA	80 LQFP

[1] The full debug/trace interface is available only on the 112- and 121-pin packages. A reduced debug interface is bonded on the 80-pin package.

1.2 Block Diagram

The superset device in the MCF52235 family comes in a 112-leaded quad flat package (LQFP) and a 121 pin MAPBGA. Figure 1-1 shows a top-level block diagram of the MCF52235.

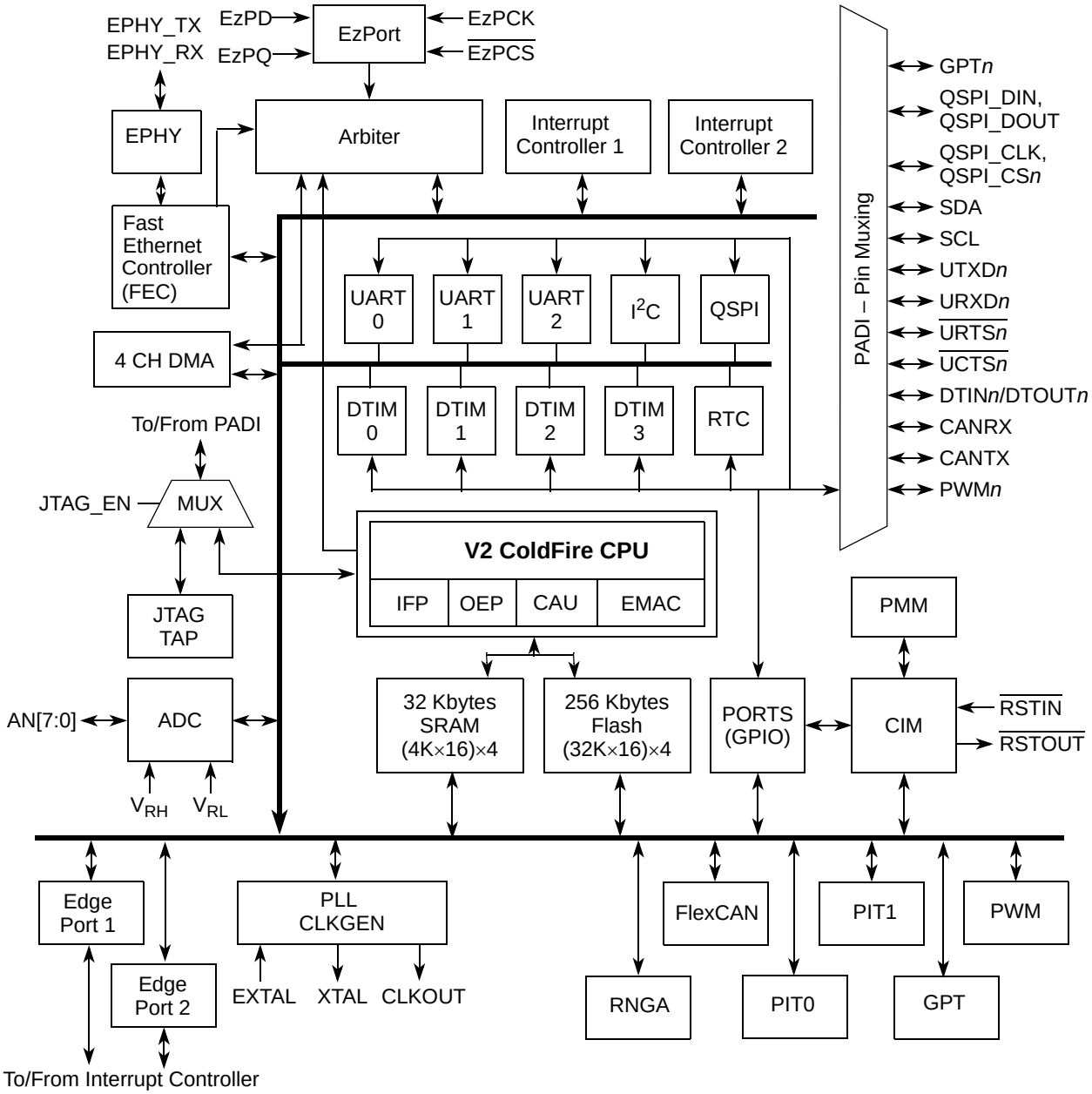


Figure 1-1. MCF52235 Block Diagram

1.3 Part Numbers and Packaging

Table 1-2 summarizes the features of the MCF52235 product family. Several speed/package options are available to match cost- or performance-sensitive applications.

Table 1-2. Orderable Part Number Summary

Freescale Part Number	Description	Speed (MHz)	Flash/SRAM (Kbytes)	Package	Temp range (°C)
MCF52230CAF60	MCF52230 Microcontroller	60	128 / 32	80 LQFP	-40 to +85
MCF52230CAL60	MCF52230 Microcontroller	60	128 / 32	112 LQFP	-40 to +85
MCF52231CAF60	MCF52231 Microcontroller, FlexCAN	60	128 / 32	80 LQFP	-40 to +85
MCF52231CAL60	MCF52231 Microcontroller, FlexCAN	60	128 / 32	112 LQFP	-40 to +85
MCF52232CAF50	MCF52232 Microcontroller	50	128 / 32	80 LQFP	-40 to +85
MCF52232AF50	MCF52232 Microcontroller	50	128 / 32	80 LQFP	0 to +70
MCF52233CAF60	MCF52233 Microcontroller	60	256 / 32	80 LQFP	-40 to +85
MCF52233CAL60	MCF52233 Microcontroller	60	256 / 32	112 LQFP	-40 to +85
MCF52233CAL60A	MCF52233 Microcontroller	60	256 / 32	112 LQFP	-40 to +85
MCF52233CVM60	MCF52233 Microcontroller	60	256 / 32	121 MAPBGA	-40 to +85
MCF52234CAL60	MCF52234 Microcontroller, FlexCAN	60	256 / 32	112 LQFP	-40 to +85
MCF52234CVM60	MCF52234 Microcontroller, FlexCAN	60	256 / 32	121 MAPBGA	-40 to +85
MCF52235CAL60	MCF52235 Microcontroller, FlexCAN, CAU, RNGA	60	256 / 32	112 LQFP	-40 to +85
MCF52235CAL60A	MCF52235 Microcontroller, FlexCAN, CAU, RNGA	60	256 / 32	112 LQFP	-40 to +85
MCF52235CVM60	MCF52235 Microcontroller, FlexCAN, CAU, RNGA	60	256 / 32	121 MAPBGA	-40 to +85
MCF52236CAF50	MCF52236 Microcontroller	50	256 / 32	80 LQFP	-40 to +85
MCF52236AF50	MCF52236 Microcontroller	50	256 / 32	80 LQFP	0 to +70
MCF52236AF50A	MCF52236 Microcontroller	50	256 / 32	80 LQFP	0 to +70

1.4 Features

The MCF52235 family includes the following features:

- Version 2 ColdFire variable-length RISC processor core
  - — Static operation
  - — 32-bit address and data paths on-chip
  - — Up to 60 MHz processor core frequency
  - — Sixteen general-purpose, 32-bit data and address registers
  - — Implements ColdFire ISA_A with extensions to support the user stack pointer register and four new instructions for improved bit processing (ISA_A+)
  - — Enhanced Multiply-Accumulate (EMAC) unit with 32-bit accumulator to support $16 \times 16 \rightarrow 32$ or $32 \times 32 \rightarrow 32$ operations
  - — Cryptography Acceleration Unit (CAU)
    - – Tightly-coupled coprocessor to accelerate software-based encryption and message digest functions
    - – FIPS-140 compliant random number generator

  - Support for DES, 3DES, AES, MD5, and SHA-1 algorithms
  - Illegal instruction decode that allows for 68K emulation support
- System debug support
  - Real time trace for determining dynamic execution path
  - Background debug mode (BDM) for in-circuit debugging (DEBUG_B+)
  - Real time debug support, with six hardware breakpoints (4 PC, 1 address and 1 data) that can be configured into a 1- or 2-level trigger
- On-chip memories
  - Up to 32 Kbytes of dual-ported SRAM on CPU internal bus, supporting core and DMA access with standby power supply support
  - Up to 256 Kbytes of interleaved Flash memory supporting 2-1-1-1 accesses
- Power management
  - Fully static operation with processor sleep and whole chip stop modes
  - Rapid response to interrupts from the low-power sleep mode (wake-up feature)
  - Clock enable/disable for each peripheral when not used
- Fast Ethernet Controller (FEC)
  - 10/100 BaseT/TX capability, half duplex or full duplex
  - On-chip transmit and receive FIFOs
  - Built-in dedicated DMA controller
  - Memory-based flexible descriptor rings
- On-chip Ethernet Transceiver (EPHY)
  - Digital adaptive equalization
  - Supports auto-negotiation
  - Baseline wander correction
  - Full-/Half-duplex support in all modes
  - Loopback modes
  - Supports MDIO preamble suppression
  - Jumbo packet
- FlexCAN 2.0B module
  - Based on and includes all existing features of the Freescale TouCAN module
  - Full implementation of the CAN protocol specification version 2.0B
    - Standard Data and Remote Frames (up to 109 bits long)
    - Extended Data and Remote Frames (up to 127 bits long)
    - 0–8 bytes data length
    - Programmable bit rate up to 1 Mbit/sec
  - Flexible Message Buffers (MBs), totalling up to 16 message buffers of 0–8 byte data length each, configurable as Rx or Tx, all supporting standard and extended messages

  - Unused Message Buffer space can be used as general purpose RAM space
  - Listen only mode capability
  - Content-related addressing
  - No read/write semaphores required
  - Three programmable mask registers: global for MBs 0-13, special for MB14, and special for MB15
  - 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
- Three universal asynchronous/synchronous receiver transmitters (UARTs)
  - 16-bit divider for clock generation
  - Interrupt control logic with maskable interrupts
  - DMA support
  - Data formats can be 5, 6, 7 or 8 bits with even, odd or no parity
  - Up to 2 stop bits in 1/16 increments
  - Error-detection capabilities
  - Modem support includes request-to-send (RTS) and clear-to-send (CTS) lines for two UARTs
  - Transmit and receive FIFO buffers
- I^2C module
  - Interchip bus interface for EEPROMs, LCD controllers, A/D converters, and keypads
  - Fully compatible with industry-standard I^2C bus
  - Master and slave modes support multiple masters
  - Automatic interrupt generation with programmable level
- Queued serial peripheral interface (QSPI)
  - Full-duplex, three-wire synchronous transfers
  - Up to four chip selects available
  - Master mode operation only
  - Programmable bit rates up to half the CPU clock frequency
  - Up to 16 pre-programmed transfers
- Fast analog-to-digital converter (ADC)
  - Eight analog input channels
  - 12-bit resolution
  - Minimum 1.125 µs conversion time
  - Simultaneous sampling of two channels for motor control applications
  - Single-scan or continuous operation
  - Optional interrupts on conversion complete, zero crossing (sign change), or under/over low/high limit

  - Unused analog channels can be used as digital I/O
- Four 32-bit DMA timers
  - 17-ns resolution at 60 MHz
  - Programmable sources for clock input, including an external clock option
  - Programmable prescaler
  - Input capture capability with programmable trigger edge on input pin
  - Output compare with programmable mode for the output pin
  - Free run and restart modes
  - Maskable interrupts on input capture or output compare
  - DMA trigger capability on input capture or output compare
- Four-channel general purpose timers
  - 16-bit architecture
  - Programmable prescaler
  - Output pulse widths variable from microseconds to seconds
  - Single 16-bit input pulse accumulator
  - Toggle-on-overflow feature for pulse-width modulator (PWM) generation
  - One dual-mode pulse accumulation channel
- Pulse-width modulation timer
  - Operates as eight channels with 8-bit resolution or four channels with 16-bit resolution
  - Programmable period and duty cycle
  - Programmable enable/disable for each channel
  - Software selectable polarity for each channel
  - Period and duty cycle are double buffered. Change takes effect when the end of the current period is reached (PWM counter reaches zero) or when the channel is disabled.
  - Programmable center or left aligned outputs on individual channels
  - Four clock sources (A, B, SA, and SB) provide for a wide range of frequencies
  - Emergency shutdown
- Real-Time Clock (RTC)
  - Maintains system time-of-day clock
  - Provides stopwatch and alarm interrupt functions
- Two periodic interrupt timers (PITs)
  - 16-bit counter
  - Selectable as free running or count down
- Software watchdog timer
  - 32-bit counter
  - Low power mode support
- Clock Generation Features

  - Crystal input
  - On-chip PLL
  - Provides clock for integrated EPHY
- Dual Interrupt Controllers (INTC0/INTC1)
  - Support for multiple interrupt sources organized as follows:
    - Fully-programmable interrupt sources for each peripheral
    - 7 fixed-level interrupt sources
    - Seven external interrupt signals
  - Unique vector number for each interrupt source
  - Ability to mask any individual interrupt source or all interrupt sources (global mask-all)
  - Support for hardware and software interrupt acknowledge (IACK) cycles
  - Combinatorial path to provide wake-up from low power modes
- DMA controller
  - Four fully programmable channels
  - Dual-address transfer support with 8-, 16-, and 32-bit data capability, along with support for 16-byte (4 x 32-bit) burst transfers
  - Source/destination address pointers that can increment or remain constant
  - 24-bit byte transfer counter per channel
  - Auto-alignment transfers supported for efficient block movement
  - Bursting and cycle steal support
  - Software-programmable DMA requesters for the UARTs (3) and 32-bit timers (4)
- Reset
  - Separate reset in and reset out signals
  - Seven sources of reset:
    - Power-on reset (POR)
    - External
    - Software
    - Watchdog
    - Loss of clock
    - Loss of lock
    - Low-voltage detection (LVD)
  - Status flag indication of source of last reset
- Chip integration module (CIM)
  - System configuration during reset
  - Selects one of three clock modes
  - Configures output pad drive strength
  - Unique part identification number and part revision number

- General purpose I/O interface
  - Up to 56 bits of general purpose I/O
  - Bit manipulation supported via set/clear functions
  - Programmable drive strengths
  - Unused peripheral pins may be used as extra GPIO
- JTAG support for system level board testing

1.4.1 V2 Core Overview

The version 2 ColdFire processor core is comprised of two separate pipelines decoupled by an instruction buffer. The two-stage instruction fetch pipeline (IFP) is responsible for instruction-address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution in the operand execution pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

The V2 core implements the ColdFire instruction set architecture revision A+ with added support for a separate user stack pointer register and four new instructions to assist in bit processing. Additionally, the MCF52235 core includes the enhanced multiply-accumulate (EMAC) unit for improved signal processing capabilities. The EMAC implements a three-stage arithmetic pipeline, optimized for 16×16 bit operations, with support for one 32-bit accumulator. Supported operands include 16- and 32-bit signed and unsigned integers, signed fractional operands, and a complete set of instructions to process these data types. The EMAC provides support for execution of DSP operations within the context of a single processor at a minimal hardware cost.

1.4.2 Integrated Debug Module

The ColdFire processor core debug interface is provided to support system debugging in conjunction with low-cost debug and emulator development tools. Through a standard debug interface, access debug information and real-time tracing capability is provided on 112-and 121-lead packages. This allows the processor and system to be debugged at full speed without the need for costly in-circuit emulators.

The on-chip breakpoint resources include a total of nine programmable 32-bit registers: an address and an address mask register, a data and a data mask register, four PC registers, and one PC mask register. These registers can be accessed through the dedicated debug serial communication channel or from the processor’s supervisor mode programming model. The breakpoint registers can be configured to generate triggers by combining the address, data, and PC conditions in a variety of single- or dual-level definitions. The trigger event can be programmed to generate a processor halt or initiate a debug interrupt exception. The MCF52235 implements revision B+ of the ColdFire Debug Architecture.

The MCF52235’s interrupt servicing options during emulator mode allow real-time critical interrupt service routines to be serviced while processing a debug interrupt event, thereby ensuring that the system continues to operate even during debugging.

To support program trace, the V2 debug module provides processor status (PST[3:0]) and debug data (DDATA[3:0]) ports. These buses and the PSTCLK output provide execution status, captured operand

data, and branch target addresses defining processor activity at the CPU's clock rate. The MCF52235 includes a new debug signal, ALLPST. This signal is the logical AND of the processor status (PST[3:0]) signals and is useful for detecting when the processor is in a halted state (PST[3:0] = 1111).

The full debug/trace interface is available only on the 112 and 121-pin packages. However, every product features the dedicated debug serial communication channel (DSI, DSO, DSCLK) and the ALLPST signal.

1.4.3 JTAG

The MCF52235 supports circuit board test strategies based on the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The test logic includes a test access port (TAP) consisting of a 16-state controller, an instruction register, and three test registers (a 1-bit bypass register, a 256-bit boundary-scan register, and a 32-bit ID register). The boundary scan register links the device's pins into one shift register. Test logic, implemented using static logic design, is independent of the device system logic.

The MCF52235 implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Sample MCF52235 system pins during operation and transparently shift out the result in the boundary scan register
- Bypass the MCF52235 for a given circuit board test by effectively reducing the boundary-scan register to a single bit
- Disable the output drive to pins during circuit-board testing
- Drive output pins to stable levels

1.4.4 On-Chip Memories

1.4.4.1 SRAM

The dual-ported SRAM module provides a general-purpose 16- or 32-Kbyte memory block that the ColdFire core can access in a single cycle. The location of the memory block can be set to any 16- or 32-Kbyte boundary within the 4-Gbyte address space. This memory is ideal for storing critical code or data structures and for use as the system stack. Because the SRAM module is physically connected to the processor's high-speed local bus, it can quickly service core-initiated accesses or memory-referencing commands from the debug module.

The SRAM module is also accessible by the DMA. The dual-ported nature of the SRAM makes it ideal for implementing applications with double-buffer schemes, where the processor and a DMA device operate in alternate regions of the SRAM to maximize system performance.

1.4.4.2 Flash

The ColdFire flash module (CFM) is a non-volatile memory (NVM) module that connects to the processor's high-speed local bus. The CFM is constructed with four banks of 32 K×16-bit flash arrays to generate 256 Kbytes of 32-bit flash memory. These arrays serve as electrically erasable and programmable, non-volatile program and data memory. The flash memory is ideal for program and data

storage for single-chip applications, allowing for field reprogramming without requiring an external high voltage source. The CFM interfaces to the ColdFire core through an optimized read-only memory controller which supports interleaved accesses from the 2-cycle flash arrays. A backdoor mapping of the flash memory is used for all program, erase, and verify operations, as well as providing a read datapath for the DMA. Flash memory may also be programmed via the EzPort, which is a serial flash programming interface that allows the flash to be read, erased and programmed by an external controller in a format compatible with most SPI bus flash memory chips. This allows easy device programming via Automated Test Equipment or bulk programming tools.

1.4.5 Cryptography Acceleration Unit

The MCF52235 device incorporates two hardware accelerators for cryptographic functions. First, the CAU is a coprocessor tightly-coupled to the V2 ColdFire core that implements a set of specialized operations to increase the throughput of software-based encryption and message digest functions, specifically the DES, 3DES, AES, MD5 and SHA-1 algorithms. Second, a random number generator provides FIPS-140 compliant 32-bit values to security processing routines. Both modules supply critical acceleration to software-based cryptographic algorithms at a minimal hardware cost.

1.4.6 Power Management

The MCF52235 incorporates several low-power modes of operation which are entered under program control and exited by several external trigger events. An integrated power-on reset (POR) circuit monitors the input supply and forces an MCU reset as the supply voltage rises. The low voltage detector (LVD) monitors the supply voltage and is configurable to force a reset or interrupt condition if it falls below the LVD trip point.

1.4.7 FlexCAN

The FlexCAN module is a communication controller implementing version 2.0 of the CAN protocol parts A and B. The CAN protocol can be used as an industrial control serial data bus, meeting the specific requirements of reliable operation in a harsh EMI environment with high bandwidth. This instantiation of FlexCAN has 16 message buffers.

1.4.8 UARTs

The MCF52235 has three full-duplex UARTs that function independently. The three UARTs can be clocked by the system bus clock, eliminating the need for an external clock source. On smaller packages, the third UART is multiplexed with other digital I/O functions.

1.4.9 I^2C Bus

The I^2C bus is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange and minimizes the interconnection between devices. This bus is suitable for applications requiring occasional communications over a short distance between many devices on a circuit board.

1.4.10 QSPI

The queued serial peripheral interface (QSPI) provides a synchronous serial peripheral interface with queued transfer capability. It allows up to 16 transfers to be queued at once, minimizing the need for CPU intervention between transfers.

1.4.11 Fast ADC

The Fast ADC consists of an eight-channel input select multiplexer and two independent sample and hold (S/H) circuits feeding separate 10- or 12-bit ADCs. The two separate converters store their results in accessible buffers for further processing.

The ADC can be configured to perform a single scan and halt, perform a scan whenever triggered, or perform a programmed scan sequence repeatedly until manually stopped.

The ADC can be configured for sequential or simultaneous conversion. When configured for sequential conversions, up to eight channels can be sampled and stored in any order specified by the channel list register. Both ADCs may be required during a scan, depending on the inputs to be sampled.

During a simultaneous conversion, both S/H circuits are used to capture two different channels at the same time. This configuration requires that a single channel may not be sampled by both S/H circuits simultaneously.

Optional interrupts can be generated at the end of the scan sequence if a channel is out of range (measures below the low threshold limit or above the high threshold limit set in the limit registers) or at several different zero crossing conditions.

1.4.12 DMA Timers (DTIM0–DTIM3)

There are four independent, DMA transfer capable 32-bit timers (DTIM0, DTIM1, DTIM2, and DTIM3) on the each device. Each module incorporates a 32-bit timer with a separate register set for configuration and control. The timers can be configured to operate from the system clock or from an external clock source using one of the DTINx signals. If the system clock is selected, it can be divided by 16 or 1. The input clock is further divided by a user-programmable 8-bit prescaler which clocks the actual timer counter register (TCRn). Each of these timers can be configured for input capture or reference (output) compare mode. Timer events may optionally cause interrupt requests or DMA transfers.

1.4.13 General Purpose Timer (GPT)

The general purpose timer (GPT) is a 4-channel timer module consisting of a 16-bit programmable counter driven by a 7-stage programmable prescaler. Each of the four channels can be configured for input capture or output compare. Additionally, one of the channels, channel 3, can be configured as a pulse accumulator.

A timer overflow function allows software to extend the timing capability of the system beyond the 16-bit range of the counter. The input capture and output compare functions allow simultaneous input waveform measurements and output waveform generation. The input capture function can capture the time of a selected transition edge. The output compare function can generate output waveforms and timer software delays. The 16-bit pulse accumulator can operate as a simple event counter or a gated time accumulator.

1.4.14 Periodic Interrupt Timers (PIT0 and PIT1)

The two periodic interrupt timers (PIT0 and PIT1) are 16-bit timers that provide interrupts at regular intervals with minimal processor intervention. Each timer can count down from the value written in its PIT modulus register or can be a free-running down-counter.

1.4.15 Pulse Width Modulation (PWM) Timers

The MCF52235 has an 8-channel, 8-bit PWM timer. Each channel has a programmable period and duty cycle as well as a dedicated counter. Each of the modulators can create independent continuous waveforms with software-selectable duty rates from 0 to 100%. The PWM outputs have programmable polarity and can be programmed as left-aligned outputs or center-aligned outputs. For higher period and duty cycle resolution, each pair of adjacent channels ([7:6], [5:4], [3:2], and [1:0]) can be concatenated to form a single 16-bit channel. The module can, therefore, be configured to support 8/0, 6/1, 4/2, 2/3, or 0/4 8-/16-bit channels.

1.4.16 Software Watchdog Timer

The watchdog timer is a 32-bit timer that facilitates recovery from runaway code. The watchdog counter is a free-running down-counter that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown.

1.4.17 Phase Locked Loop (PLL)

The clock module contains a crystal oscillator, 8 MHz on-chip relaxation oscillator (OCO), phase-locked loop (PLL), reduced frequency divider (RFD), low-power divider status/control registers, and control logic. To improve noise immunity, the PLL, crystal oscillator, and relaxation oscillator have their own power supply inputs: VDDPLL and VSSPLL. All other circuits are powered by the normal supply pins, VDD and VSS.

1.4.18 Interrupt Controller (INTC0/INTC1)

There are two interrupt controllers on the MCF52235. These interrupt controllers are organized as seven levels with up to nine interrupt sources per level. Each interrupt source has a unique interrupt vector, and provide each peripheral with all necessary interrupts. Each internal interrupt has a programmable level [1-7] and priority within the level. The seven external interrupts have fixed levels/priorities.

1.4.19 DMA Controller

The direct memory access (DMA) controller provides an efficient way to move blocks of data with minimal processor intervention. It has four channels that allow byte, word, longword, or 16-byte burst line transfers. These transfers are triggered by software explicitly setting a DCRn[START] bit or by the occurrence of certain UART or DMA timer events.

1.4.20 Reset

The reset controller determines the source of reset, asserts the appropriate reset signals to the system, and keeps track of what caused the last reset. There are seven sources of reset:

- External reset input
- Power-on reset (POR)
- Watchdog timer
- Phase locked-loop (PLL) loss of lock
- PLL loss of clock
- Software
- Low-voltage detector (LVD)

Control of the LVD and its associated reset and interrupt are managed by the reset controller. Other registers provide status flags indicating the last source of reset and a control bit for software assertion of the $\overline{\text{RSTO}}$ pin.

1.4.21 GPIO

Nearly all pins on the MCF52235 have general purpose I/O capability in addition to their primary functions and are grouped into 8-bit ports. Some ports do not utilize all 8 bits. Each port has registers that configure, monitor, and control the port pins.

1.5 Memory Map Overview

Table 1-3. System Memory Map

Base Address (Hex)	Size	Use
0x0000_0000	1G	On-Chip Flash/RAM Array2
0x4000_0000	64 bytes	System Control Module
0x4000_0040	64 bytes	Reserved
0x4000_0080	128 bytes	Reserved
0x4000_0100	16 bytes	DMA (Channel 0)
0x4000_0110	16 bytes	DMA (Channel 1)
0x4000_0120	16 bytes	DMA (Channel 2)
0x4000_0130	16 bytes	DMA (Channel 3)
0x4000_0140	196 bytes	Reserved
0x4000_0200	64 bytes	UART0
0x4000_0240	64 bytes	UART1
0x4000_0280	64 bytes	UART2
0x4000_02c0	64 bytes	Reserved

Table 1-3. System Memory Map (continued)

Base Address (Hex)	Size	Use
0x4000_0300	64 bytes	I2C
0x4000_0340	64 bytes	QSPI
0x4000_0380	64 bytes	Reserved
0x4000_03C0	64 bytes	RTC
0x4000_0400	64 bytes	TMR0
0x4000_0440	64 bytes	TMR1
0x4000_0480	64 bytes	TMR2
0x4000_04c0	64 bytes	TMR3
0x4000_0500	1792 bytes	Reserved
0x4000_0c00	256 bytes	Interrupt Cntl 0
0x4000_0d00	256 bytes	Interrupt Cntl 1
0x4000_0e00	256 bytes	Reserved
0x4000_0f00	256 bytes	Global Interrupt Ack Cycles
0x4000_1000	1K	Fast Ethernet Controller - Registers and MIB RAM
0x4000_1400	1K	Fast Ethernet Controller - FIFO Memory
0x4000_1800	1M – 6K	Reserved
0x4010_0000	64K	Ports
0x4011_0000	64K	CIM_IBO
0x4012_0000	64K	Clocks (PLLMRBI)
0x4013_0000	64K	Edge Port 0
0x4014_0000	64K	Edge Port 1
0x4015_0000	64K	Programmable Interval Timer 0
0x4016_0000	64K	Programmable Interval Timer 1
0x4017_0000	64K	Reserved
0x4018_0000	64K	Reserved
0x4019_0000	64K	ADC
0x401a_0000	64K	Timer
0x401b_0000	64K	PWM
0x401c_0000	64K	FlexCAN2
0x401d_0000	64K	CFM (Flash) control registers
0x401e_0000	64K	Ethernet Physical Transceiver

Table 1-3. System Memory Map (continued)

Base Address (Hex)	Size	Use
0x401f_0000	64K	Random Number Generator H/W Accelerator
0x4020_0000	62M	Reserved
0x4400_0000	256K	CFM (Flash) memory for IPS reads and writes
0x4408_0000	1G – 64M – 256K	Reserved
0x8000_0000	2G	Reserved

Chapter 2
Signal Descriptions

2.1 Introduction

This chapter describes signals implemented on this device and includes an alphabetical listing of signals that characterizes each signal as an input or output, defines its state at reset, and identifies whether a pull-up resistor should be used.

NOTE

The terms ‘assertion’ and ‘negation’ are used to avoid confusion when dealing with a mixture of active-low and active-high signals. The term ‘asserted’ indicates that a signal is active, independent of the voltage level. The term ‘negated’ indicates that a signal is inactive.

Active-low signals, such as $\overline{\text{SRAS}}$ and $\overline{\text{TA}}$, are indicated with an overbar.

2.2 Overview

Figure 2-1 shows the block diagram of the device with the signal interface.

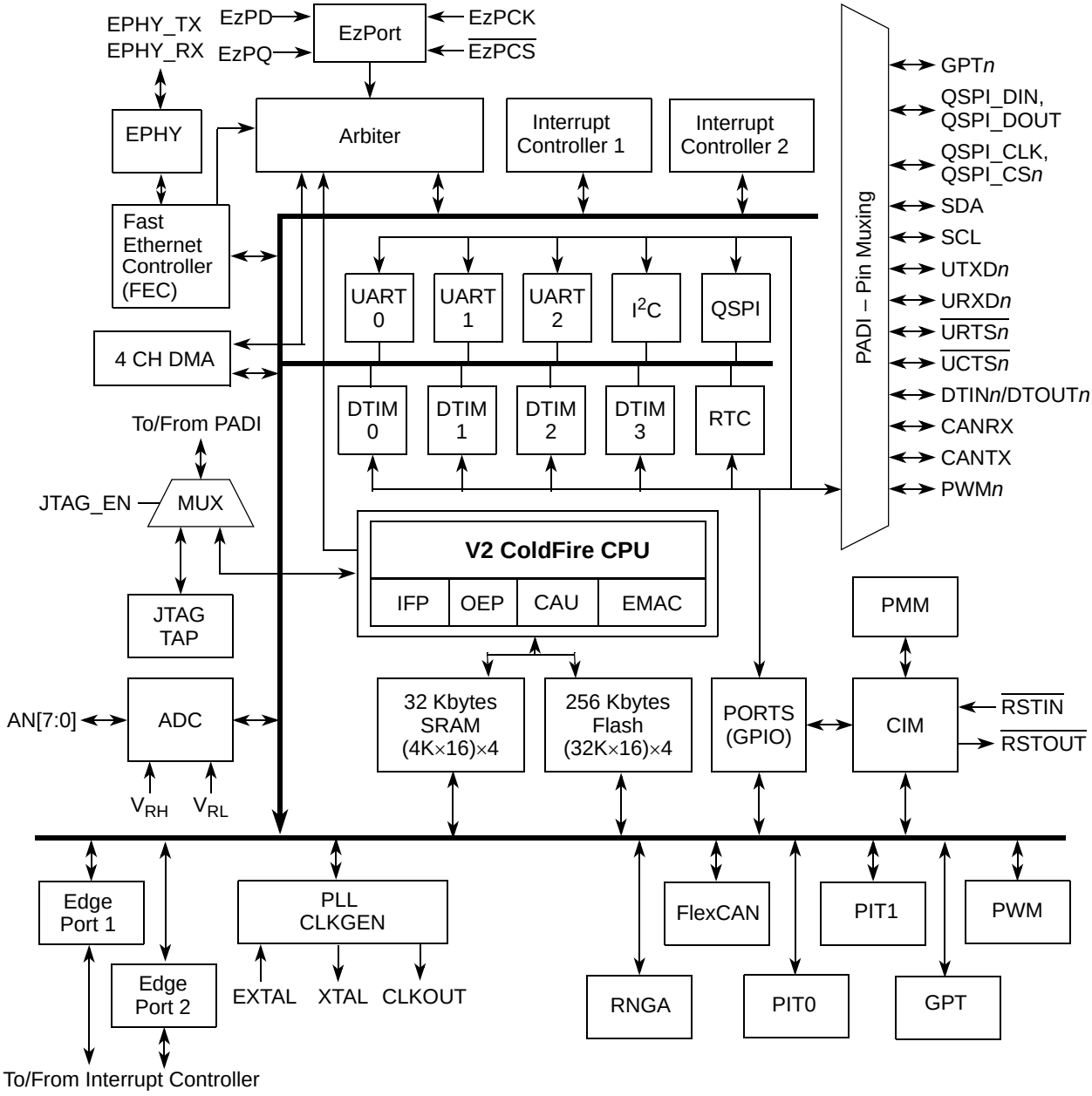


Figure 2-1. Block Diagram with Signal Interfaces

Table 2-1 shows the pin functions by primary and alternate purpose, and illustrates which packages contain each pin.

Table 2-1. Pin Functions by Primary and Alternate Purpose

Pin Group	Primary Function	Secondary Function	Tertiary Function	Quaternary Function	Drive Strength/ Control[1]	Wired OR Control	Pull-up/ Pull-down[2]	Pin on 121 MAPBGA	Pin on 112 LQFP	Pin on 80 LQFP
ADC[3]	AN7	—	—	PAN[7]	Low	—	—	A10	88	64
	AN6	—	—	PAN[6]	Low	—	—	B10	87	63
	AN5	—	—	PAN[5]	Low	—	—	A11	86	62
	AN4	—	—	PAN[4]	Low	—	—	B11	85	61
	AN3	—	—	PAN[3]	Low	—	—	C9	89	65
	AN2	—	—	PAN[2]	Low	—	—	B9	90	66
	AN1	—	—	PAN[1]	Low	—	—	A9	91	67
	AN0	—	—	PAN[0]	Low	—	—	C8	92	68
	SYNCA	CANTX[4]	FEC_MDIO	PAS[3]	PDSR[39]	—	—	K1	28	20
	SYNCB	CANRX[4]	FEC_MDC	PAS[2]	PDSR[39]	—	—	J1	27	19
	VDDA	—	—	—	N/A	N/A	—	A8	93	69
	VSSA	—	—	—	N/A	N/A	—	A7	96	72
	VRH	—	—	—	N/A	N/A	—	B8	94	70
	VRL	—	—	—	N/A	N/A	—	B7	95	71
Clock Generation	EXTAL	—	—	—	N/A	N/A	—	L7	48	36
	XTAL	—	—	—	N/A	N/A	—	J7	49	37
	VDDPLL[5]	—	—	—	N/A	N/A	—	K6	45	33
	VSSPLL	—	—	—	N/A	N/A	—	K7	47	35
Debug Data	ALLPST	—	—	—	High	—	—	D3	7	7
	DDATA[3:0]	—	—	PDD[7:4]	High	—	—	E1, F3,F2, F1	12,13,16,17	—
	PST[3:0]	—	—	PDD[3:0]	High	—	—	D10, D9, E10, E9	80,79,78,77	—

Table 2-1. Pin Functions by Primary and Alternate Purpose (continued)

Pin Group	Primary Function	Secondary Function	Tertiary Function	Quaternary Function	Drive Strength/ Control[1]	Wired OR Control	Pull-up/ Pull-down[2]	Pin on 121 MAPBGA	Pin on 112 LQFP	Pin on 80 LQFP
Ethernet LEDs	ACTLED	—	—	PLD[0]	PDSR[32]	PWOR[8]	—	C11	84	60
	COLLED	—	—	PLD[4]	PDSR[36]	PWOR[12]	—	J9	58	42
	DUPLED	—	—	PLD[3]	PDSR[35]	PWOR[11]	—	J10	59	43
	LNKLED	—	—	PLD[1]	PDSR[33]	PWOR[9]	—	C10	83	59
	SPDLED	—	—	PLD[2]	PDSR[34]	PWOR[10]	—	D11	81	57
	RXLED	—	—	PLD[5]	PDSR[37]	PWOR[13]	—	H9	52	—
	TXLED	—	—	PLD[6]	PDSR[38]	PWOR[14]	—	H8	51	—
	VDDR	—	—	—	—	—	—	D8	82	58
Ethernet PHY	PHY_RBIAS	—	—	—	—	—		J11	66	46
	PHY_RXN	—	—	—	—	—		E11	74	54
	PHY_RXP	—	—	—	—	—		F11	73	53
	PHY_TXN	—	—	—	—	—		H11	71	51
	PHY_TXP	—	—	—	—	—		G11	70	50
	PHY_VDDA[5]	—	—	—		N/A		H10	68	48
	PHY_VDDRX[5]	—	—	—		N/A		F10	75	55
	PHY_VDDTX[5]	—	—	—		N/A		G10	69	49
	PHY_VSSA	—	—	—		N/A		G8	67	47
	PHY_VSSRX	—	—	—		N/A		F9	76	56
	PHY_VSSTX	—	—	—		N/A		G9	72	52
I^2C	SCL	CANTX[4]	UTXD2	PAS[0]	PDSR[0]	—	Pull-Up[6]	A3	111	79
	SDA	CANRX[4]	URXD2	PAS[1]	PDSR[0]	—	Pull-Up[6]	A2	112	80

Table 2-1. Pin Functions by Primary and Alternate Purpose (continued)

Pin Group	Primary Function	Secondary Function	Tertiary Function	Quaternary Function	Drive Strength/ Control[1]	Wired OR Control	Pull-up/ Pull-down[2]	Pin on 121 MAPBGA	Pin on 112 LQFP	Pin on 80 LQFP
Interrupts[3]	IRQ15	—	—	PGP[7]	PSDR[47]	—	Pull-Up[6]	A4	106	—
	IRQ14	—	—	PGP[6]	PSDR[46]	—	Pull-Up[6]	A5	105	—
	IRQ13	—	—	PGP[5]	PSDR[45]	—	Pull-Up[6]	A6	98	—
	IRQ12	—	—	PGP[4]	PSDR[44]	—	Pull-Up[6]	C7	97	—
Continued Interrupts[3]	IRQ11	—	—	PGP[3]	PSDR[43]	—	Pull-Up[6]	K9	57	41
	IRQ10	—	—	PGP[2]	PSDR[42]	—	Pull-Up[6]	L1	29	—
	IRQ9	—	—	PGP[1]	PSDR[41]	—	Pull-Up[6]	E2	11	—
	IRQ8	—	—	PGP[0]	PSDR[40]	—	Pull-Up	E3	10	—
	IRQ7	—	—	PNQ[7]	Low	—	Pull-Up[6]	L9	56	40
	IRQ6	—	FEC_RXER	PNQ[6]	Low	—	Pull-Up[6]	G3	19	—
	IRQ5	—	FEC_RXD[1]	PNQ[5]	Low	—	Pull-Up[6]	G2	20	—
	IRQ4	—	—	PNQ[4]	Low	—	Pull-Up[6]	L5	41	29
	IRQ3	—	FEC_RXD[2]	PNQ[3]	Low	—	Pull-Up[6]	L8	53	—
	IRQ2	—	FEC_RXD[3]	PNQ[2]	Low	—	Pull-Up[6]	K8	54	—
	IRQ1	SYNCA	PWM1	PNQ[1]	High	—	Pull-Up[6]	J8	55	39
JTAG/BDM	JTAG_EN	—	—	—	N/A	N/A	Pull-Down	G4	18	12
	TCLK/ PSTCLK	CLKOUT	—	—	High	—	Pull-Up[7]	A1	1	1
	TDI/DSI	—	—	—	N/A	N/A	Pull-Up[7]	C3	4	4
	TDO/DSO	—	—	—	High	N/A	—	C2	5	5
	TMS/$\overline{\text{BKPT}}$	—	—	—	N/A	N/A	Pull-Up[7]	B1	2	2
	TRST/DSCLK	—	—	—	N/A	N/A	Pull-Up	C1	6	6
Mode Selection	$\overline{\text{RCON}}$/EZPCS	—	—	—	N/A	N/A	Pull-Up	B2	3	3

Table 2-1. Pin Functions by Primary and Alternate Purpose (continued)

Pin Group	Primary Function	Secondary Function	Tertiary Function	Quaternary Function	Drive Strength/ Control[1]	Wired OR Control	Pull-up/ Pull-down[2]	Pin on 121 MAPBGA	Pin on 112 LQFP	Pin on 80 LQFP
PWM	PWM7	—	—	PTD[3]	PDSR[31]	—	—	C5	104	—
	PWM5	—	—	PTD[2]	PDSR[30]	—	—	B5	103	—
	PWM3	—	—	PTD[1]	PDSR[29]	—	—	C6	100	—
	PWM1	—	—	PTD[0]	PDSR[28]	—	—	B6	99	—
QSPI[3]	QSPI_DIN/ EZPD	CANRX[4]	URXD1	PQS[1]	PDSR[2]	PWOR[4]	—	H4	34	25
	QSPI_DOUT/ EZPQ	CANTX[4]	UTXD1	PQS[0]	PDSR[1]	PWOR[5]	—	J4	35	26
	QSPI_CLK/ EZPCK	SCL	URTS1	PQS[2]	PDSR[3]	PWOR[6]	Pull-Up[8]	K4	36	27
	QSPI_CS3	SYNCA	SYNCB	PQS[6]	PDSR[7]	—	—	K5	40	—
	QSPI_CS2	—	FEC_TXCLK	PQS[5]	PDSR[6]	—	—	J5	39	—
	QSPI_CS1	—	FEC_TXEN	PQS[4]	PDSR[5]	—	—	H5	38	—
	QSPI_CS0	SDA	UCTS1	PQS[3]	PDSR[4]	PWOR[7]	Pull-Up[8]	L4	37	28
Reset[9]	RSTI	—	—	—	N/A	N/A	Pull-Up[9]	J6	44	32
	RSTO	—	—	—	high	—	—	L6	46	34
Test	TEST	—	—	—	N/A	N/A	Pull-Down	H7	50	38
Timers, 16-bit[3]	GPT3	FEC_TXD[3]	PWM7	PTA[3]	PDSR[23]	—	Pull-Up[10]	B4	107	75
	GPT2	FEC_TXD[2]	PWM5	PTA[2]	PDSR[22]	—	Pull-Up[10]	C4	108	76
	GPT1	FEC_TXD[1]	PWM3	PTA[1]	PDSR[21]	—	Pull-Up[10]	D4	109	77
	GPT0	FEC_TXER	PWM1	PTA[0]	PDSR[20]	—	Pull-Up[10]	B3	110	78
Timers, 32-bit	DTIN3	DTOUT3	PWM6	PTC[3]	PDSR[19]	—	—	H1	22	14
	DTIN2	DTOUT2	PWM4	PTC[2]	PDSR[18]	—	—	G1	21	13
	DTIN1	DTOUT1	PWM2	PTC[1]	PDSR[17]	—	—	D1	9	9
	DTIN0	DTOUT0	PWM0	PTC[0]	PDSR[16]	—	—	D2	8	8

Table 2-1. Pin Functions by Primary and Alternate Purpose (continued)

Pin Group	Primary Function	Secondary Function	Tertiary Function	Quaternary Function	Drive Strength/ Control[1]	Wired OR Control	Pull-up/ Pull-down[2]	Pin on 121 MAPBGA	Pin on 112 LQFP	Pin on 80 LQFP
UART 0[3]	$\overline{\text{UCTS0}}$	CANRX[4]	FEC_RXCLK	PUA[3]	PDSR[11]	—	—	J2	26	18
	$\overline{\text{URTS0}}$	CANTX[4]	FEC_RXDV	PUA[2]	PDSR[10]	—	—	H2	25	17
	URXD0	—	FEC_RXD[0]	PUA[1]	PDSR[9]	PWOR[0]	—	K2	30	21
	UTXD0	—	FEC_CRS	PUA[0]	PDSR[8]	PWOR[1]	—	L2	31	22
UART 1[3]	$\overline{\text{UCTS1}}$	SYNCA	URXD2	PUB[3]	PDSR[15]	—	—	J3	24	16
	$\overline{\text{URTS1}}$	SYNCB	UTXD2	PUB[2]	PDSR[14]	—	—	H3	23	15
	URXD1	—	FEC_TXD[0]	PUB[1]	PDSR[13]	PWOR[2]	—	K3	32	23
	UTXD1	—	FEC_COL	PUB[0]	PDSR[12]	PWOR[3]	—	L3	33	24
UART 2	$\overline{\text{UCTS2}}$	—	—	PUC[3]	PDSR[27]	—	—	L10	61	—
	$\overline{\text{URTS2}}$	—	—	PUC[2]	PDSR[26]	—	—	K10	60	—
	URXD2	—	—	PUC[1]	PDSR[25]	—	—	K11	62	—
	UTXD2	—	—	PUC[0]	PDSR[24]	—	—	L11	63	—
FlexCAN	SYNCA	CANTX[4]	FEC_MDIO	PAS[3]	PDSR[39]	—	—	—	28	20
	SYNCB	CANRX[4]	FEC_MDC	PAS[2]	PDSR[39]	—	—	—	27	19
VDD[5,11]	VDD	—	—	—	N/A	N/A	—	D7, E8	65,102	45,74
VDDX	VDDX	—	—	—	N/A	N/A	—	D5, D6, E6, G5, G6, G7, H6	14, 43	10, 31
VSS	VSS	—	—	—	N/A	N/A	—	E4, E5, E7,F4, F5, F6, F7, F8	64,101	44,73
VSSX	VSSX	—	—	—	N/A	N/A	—	—	15, 42	11, 30

1 The PDSR and PSSR registers are described in Chapter 14, "General Purpose I/O Module. All programmable signals default to 2mA drive in normal (single-chip) mode.
2 All signals have a pull-up in GPIO mode.
3 The use of an external PHY limits ADC, interrupt, and QSPI functionality. It also disables the UART0/1 and timer pins.
4 The multiplexed CANTX and CANRX signals do not have dedicated pins, but are available as muxed replacements for other signals.
5 The VDD1, VDD2, VDDPLL, and PHY_VDD pins are for decoupling only and should not have power directly applied to them.
6 For primary and GPIO functions only.
7 Only when JTAG mode is enabled.

[8] For secondary and GPIO functions only.
[9] RSTI has an internal pull-up resistor; however, the use of an external resistor is strongly recommended.
[10] For GPIO function. Primary Function has pull-up control within the GPT module.
[11] This list for power and ground does not include those dedicated power/ground pins included elsewhere, e.g. in the Ethernet PHY.

2.3 Reset Signals

Table 2-2 describes signals that are used to reset the chip or as a reset indication.

Table 2-2. Reset Signals

Signal Name	Abbreviation	Function	I/O
Reset In	$\overline{\text{RSTI}}$	Primary reset input to the device. Asserting $\overline{\text{RSTI}}$ immediately resets the CPU and peripherals.	I
Reset Out	$\overline{\text{RSTO}}$	Driven low for 512 CPU clocks after the reset source has deasserted and PLL locked.	O

2.4 PLL and Clock Signals

Table 2-3 describes signals that are used to support the on-chip clock generation circuitry.

Table 2-3. PLL and Clock Signals

Signal Name	Abbreviation	Function	I/O
External Clock In	EXTAL	Crystal oscillator or external clock input.	I
Crystal	XTAL	Crystal oscillator output.	O
Clock Out	CLKOUT	This output signal reflects the internal system clock.	O

2.5 Mode Selection

Table 2-4 describes signals used in mode selection.

Table 2-4. Mode Selection Signals

Signal Name	Abbreviation	Function	I/O
Reset Configuration	$\overline{\text{RCON}}$	The serial flash programming mode is entered by asserting the $\overline{\text{RCON}}$ pin (with the TEST pin negated) as the chip comes out of reset. During this mode, the EzPort has access to the flash memory which can be programmed from an external device.	
Test	TEST	Reserved for factory testing only and in normal modes of operation should be connected to VSS to prevent unintentional activation of test functions.	I

2.6 External Interrupt Signals

Table 2-5 describes the external interrupt signals.

Table 2-5. External Interrupt Signals

Signal Name	Abbreviation	Function	I/O
External Interrupts	$\overline{\text{IRQ}}$[15:1]	External interrupt sources.	I

2.7 Queued Serial Peripheral Interface (QSPI)

Table 2-6 describes the QSPI signals.

Table 2-6. Queued Serial Peripheral Interface (QSPI) Signals

Signal Name	Abbreviation	Function	I/O
QSPI Synchronous Serial Output	QSPI_DOUT	Provides the serial data from the QSPI and can be programmed to be driven on the rising or falling edge of QSPI_CLK.	O
QSPI Synchronous Serial Data Input	QSPI_DIN	Provides the serial data to the QSPI and can be programmed to be sampled on the rising or falling edge of QSPI_CLK.	I
QSPI Serial Clock	QSPI_CLK	Provides the serial clock from the QSPI. The polarity and phase of QSPI_CLK are programmable.	O
Synchronous Peripheral Chip Selects	QSPI_CS[3:0]	QSPI peripheral chip selects that can be programmed to be active high or low.	O

2.8 Fast Ethernet Controller PHY Signals

Table 7 describes the Fast Ethernet Controller (FEC) Signals.

Table 7. Fast Ethernet Controller (FEC) Signals

Signal Name	Abbreviation	Function	I/O
Twisted Pair Input +	RXP	Differential Ethernet twisted-pair input pin. This pin is high-impedance out of reset.	I
Twisted Pair Input -	RXN	Differential Ethernet twisted-pair input pin. This pin is high-impedance out of reset.	I
Twisted Pair Output +	TXN	Differential Ethernet twisted-pair output pin. This pin is high-impedance out of reset.	O
Twisted Pair Output -	TXP	Differential Ethernet twisted-pair output pin. This pin is high-impedance out of reset.	O
Bias Control Resistor	RBIAS	Connect a 12.4 kΩ (1.0%) external resistor, RBIAS, between the PHY_RBIAS pin and analog ground. Place this resistor as near to the chip pin as possible. Stray capacitance must be kept to less than 10 pF (>50 pF causes instability). No high-speed signals can be permitted in the region of RBIAS.	I
Activity LED	ACT_LED	Indicates when the EPHY is transmitting or receiving	O
Link LED	LINK_LED	Indicates when the EPHY has a valid link	O
Speed LED	SPD_LED	Indicates the speed of the EPHY connection	O
Duplex LED	DUPLED	Indicates the duplex (full or half) of the EPHY connection	O
Collision LED	COLLED	Indicates if the EPHY detects a collision	O
Transmit LED	TXLED	Indicates if the EPHY is transmitting	O
Receive LED	RXLED	Indicates if the EPHY is receiving	O

2.9 I^2C I/O Signals

Table 2-8 describes the I^2C serial interface module signals.

Table 2-8. I^2C I/O Signals

Signal Name	Abbreviation	Function	I/O
Serial Clock	SCL	Open-drain clock signal for the for the I^2C interface. It is driven by the I^2C module when the bus is in master mode or it becomes the clock input when the I^2C is in slave mode.	I/O
Serial Data	SDA	Open-drain signal that serves as the data input/output for the I^2C interface.	I/O

2.10 UART Module Signals

Table 2-9 describes the UART module signals.

Table 2-9. UART Module Signals

Signal Name	Abbreviation	Function	I/O
Transmit Serial Data Output	UTXD*n*	Transmitter serial data outputs for the UART modules. The output is held high (mark condition) when the transmitter is disabled, idle, or in the local loopback mode. Data is shifted out, LSB first, on this pin at the falling edge of the serial clock source.	O
Receive Serial Data Input	URXD*n*	Receiver serial data inputs for the UART modules. Data is received on this pin LSB first. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I
Clear-to-Send	$\overline{\text{UCTS}n}$	Indicate to the UART modules that they can begin data transmission.	I
Request-to-Send	$\overline{\text{URTS}n}$	Automatic request-to-send outputs from the UART modules. This signal can also be configured to be asserted and negated as a function of the RxFIFO level.	O

2.11 DMA Timer Signals

Table 2-10 describes the signals of the four DMA timer modules.

Table 2-10. DMA Timer Signals

Signal Name	Abbreviation	Function	I/O
DMA Timer Input	DTIN*n*	Event input to the DMA timer modules.	I
DMA Timer Output	DTOUT*n*	Programmable output from the DMA timer modules.	O

2.12 ADC Signals

Table 2-11 describes the signals of the analog-to-digital converter.

Table 2-11. ADC Signals

Signal Name	Abbreviation	Function	I/O
Analog Inputs	AN[7:0]	Inputs to the A-to-D converter.	I
Analog Reference	V_{RH}	Reference voltage high and low inputs.	I
	V_{RL}		I
Analog Supply	V_{DDA}	Isolate the ADC circuitry from power supply noise	—
	V_{SSA}		—

2.13 General Purpose Timer Signals

Table 2-12 describes the general purpose timer signals.

Table 2-12. GPT Signals

Signal Name	Abbreviation	Function	I/O
General Purpose Timer Input/Output	GPT[3:0]	Inputs to or outputs from the general purpose timer module	I/O

2.14 Pulse Width Modulator Signals

Table 2-13 describes the PWM signals.

Table 2-13. PWM Signals

Signal Name	Abbreviation	Function	I/O
PWM Output Channels	PWM[7:0]	Pulse width modulated output for PWM channels	O

2.15 Debug Support Signals

The signals in Table 2-14 are used as the interface to the on-chip JTAG controller and also to interface to the BDM logic.

Table 2-14. Debug Support Signals

Signal Name	Abbreviation	Function	I/O
JTAG Enable	JTAG_EN	Select between debug module and JTAG signals at reset	I
Test Reset	$\overline{\text{TRST}}$	This active-low signal is used to initialize the JTAG logic asynchronously.	I
Test Clock	TCLK	Used to synchronize the JTAG logic.	I
Test Mode Select	TMS	Used to sequence the JTAG state machine. TMS is sampled on the rising edge of TCLK.	I
Test Data Input	TDI	Serial input for test instructions and data. TDI is sampled on the rising edge of TCLK.	I
Test Data Output	TDO	Serial output for test instructions and data. TDO is three-stateable and is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
Development Serial Clock	DSCLK	Development Serial Clock. Internally synchronized input. (The logic level on DSCLK is validated if it has the same value on two consecutive rising bus clock edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is PSTCLK/5. At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.	I
Breakpoint	$\overline{\text{BKPT}}$	Breakpoint. Input used to request a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status signals (PST[3:0]) as the value 0xF.	I

Table 2-14. Debug Support Signals (continued)

Signal Name	Abbreviation	Function	I/O
Development Serial Input	DSI	Development Serial Input. Internally synchronized input that provides data input for the serial communication port to the debug module after the DSCLK has been seen as high (logic 1).	I
Development Serial Output	DSO	Development Serial Output. Provides serial output communication for debug module responses. DSO is registered internally. The output is delayed from the validation of DSCLK high.	O
Debug Data	DDATA[3:0]	Debug data. Displays captured processor data and breakpoint status. The CLKOUT signal can be used by the development system to know when to sample DDATA[3:0].	O
Processor Status Clock	PSTCLK	Processor Status Clock. Delayed version of the processor clock. Its rising edge appears in the center of valid PST and DDATA output. PSTCLK indicates when the development system should sample PST and DDATA values. If real-time trace is not used, setting CSR[PCD] keeps PSTCLK, and PST and DDATA outputs from toggling without disabling triggers. Non-quiescent operation can be reenabled by clearing CSR[PCD], although the external development systems must resynchronize with the PST and DDATA outputs. PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing.	O
Processor Status Outputs	PST[3:0]	Indicate core status. Debug mode timing is synchronous with the processor clock; status is unrelated to the current bus transfer. The CLKOUT signal can be used by the development system to know when to sample PST[3:0].	O
All Processor Status Outputs	ALLPST	Logical AND of PST[3.0]	O

2.16 EzPort Signal Descriptions

Table 2-15 contains a list of EzPort external signals

Table 2-15. EzPort Signal Descriptions

Signal Name	Abbreviation	Function	I/O
EzPort Clock	EZPCK	Shift clock for EzPort transfers	I
EzPort Chip Select	$\overline{\text{EZPCS}}$	Chip select for signaling the start and end of serial transfers	I
EzPort Serial Data In	EZPD	EZPD is sampled on the rising edge of EZPCK	I
EzPort Serial Data Out	EZPQ	EZPQ transitions on the falling edge of EZPCK	O

2.17 Power and Ground Pins

The pins described in Table 2-16 provide system power and ground to the chip. Multiple pins are provided for adequate current capability. All power supply pins must have adequate decoupling (bypass capacitance) for high-frequency noise suppression.

Table 2-16. Power and Ground Pins

Signal Name	Abbreviation	Function	I/O
PLL Analog Supply	VDDPLL, VSSPLL	Dedicated power supply signals to isolate the sensitive PLL analog circuitry from the normal levels of noise present on the digital power supply.	I
Positive Supply	VDD	These pins supply positive power to the core logic.	I
Ground	VSS	This pin is the negative supply (ground) to the chip.	

Some of the V_{DD} and V_{SS} pins on the device are only to be used for noise bypass. Figure 2 shows a typical connection diagram. Pay particular attention to those pins which show only capacitor connections.

CAUTION

Avoid connecting power-supply voltage directly to pins in Figure 2 which show only capacitor connections, as doing so could damage the device severely.

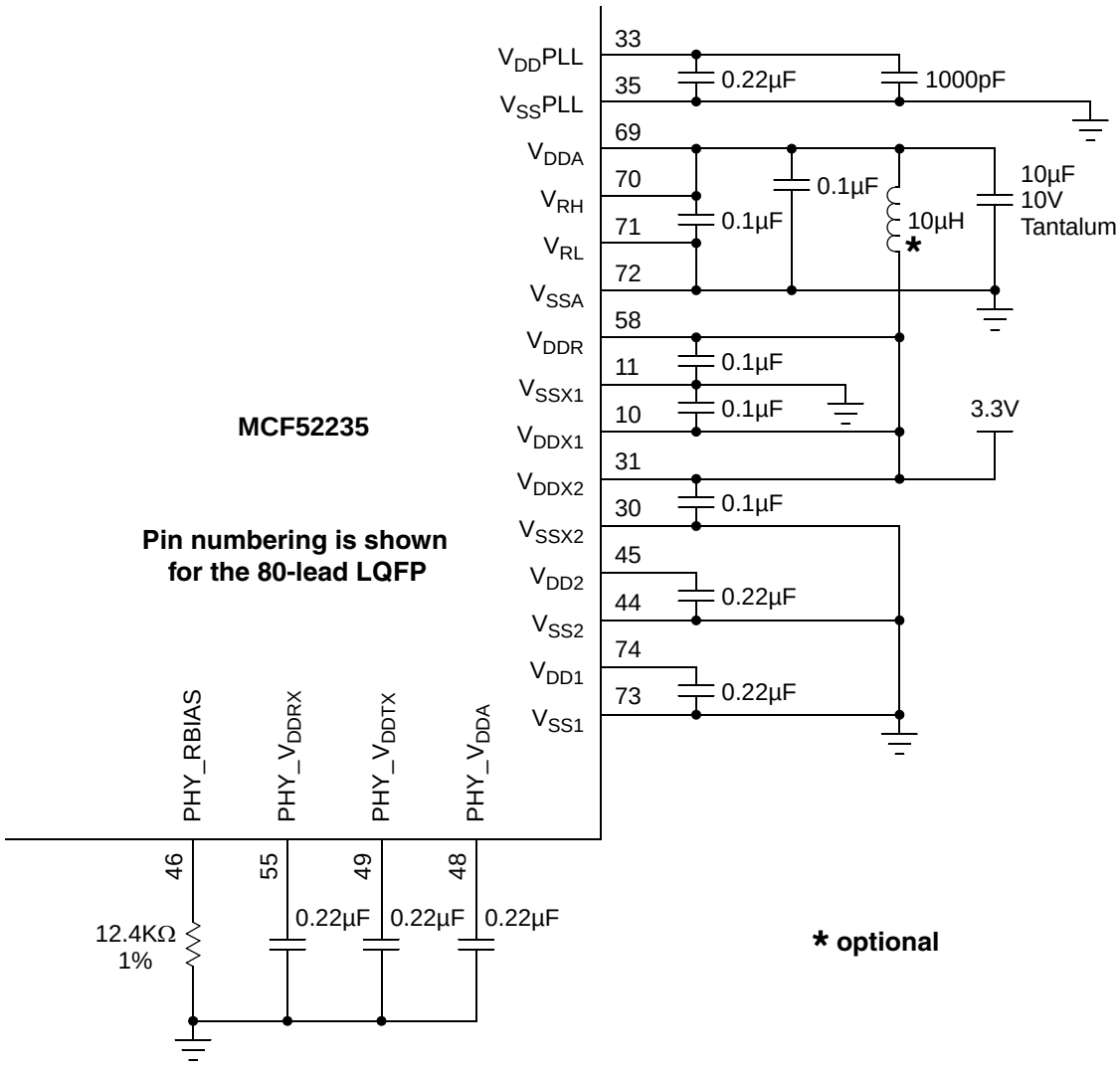


Figure 2. Suggested connection scheme for Power and Ground

Chapter 3
ColdFire Core

3.1 Introduction

This section describes the organization of the Version 2 (V2) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA_A+ definition in the *ColdFire Family Programmer's Reference Manual*.

3.1.1 Overview

As with all ColdFire cores, the V2 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.

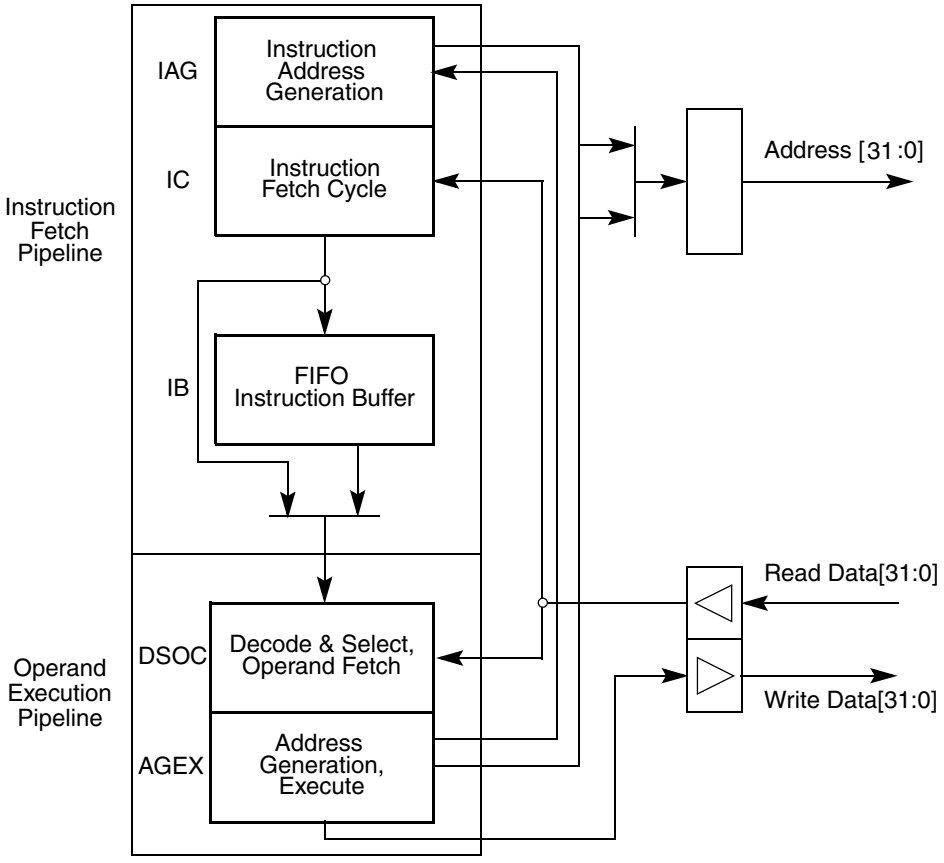


Figure 3-1. V2 ColdFire Core Pipelines

The instruction fetch pipeline (IFP) is a two-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the two-stage operand execution pipeline (OEP), that decodes the

instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer serving as a FIFO queue, the IFP is able to prefetch instructions in advance of their actual use by the OEP thereby minimizing time stalled waiting for instructions.

The V2 ColdFire core pipeline stages include the following:

- Two-stage instruction fetch pipeline (IFP) (plus optional instruction buffer stage)
  - — Instruction address generation (IAG) — Calculates the next prefetch address
  - — Instruction fetch cycle (IC)—Initiates prefetch on the processor's local bus
  - — Instruction buffer (IB) — Optional buffer stage minimizes fetch latency effects using FIFO queue
- Two-stage operand execution pipeline (OEP)
  - — Decode and select/operand fetch cycle (DSOC)—Decodes instructions and fetches the required components for effective address calculation, or the operand fetch cycle
  - — Address generation/execute cycle (AGEX)—Calculates operand address or executes the instruction

When the instruction buffer is empty, opcodes are loaded directly from the IC cycle into the operand execution pipeline. If the buffer is not empty, the IFP stores the contents of the fetched instruction in the IB until it is required by the OEP.

For register-to-register and register-to-memory store operations, the instruction passes through both OEP stages once. For memory-to-register and read-modify-write memory operations, an instruction is effectively staged through the OEP twice; the first time to calculate the effective address and initiate the operand fetch on the processor's local bus, and the second time to complete the operand reference and perform the required function defined by the instruction.

The resulting pipeline and local bus structure allow the V2 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

3.2 Memory Map/Register Description

The following sections describe the processor registers in the user and supervisor programming models. The programming model is selected based on the processor privilege level (user mode or supervisor mode) as defined by the S bit of the status register (SR). Table 3-1 lists the processor registers.

The user-programming model consists of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)
- 8-bit condition code register (CCR)
- EMAC registers (described fully in Chapter 4, "Enhanced Multiply-Accumulate Unit (EMAC
  - — Four 48-bit accumulator registers partitioned as follows:
    - – Four 32-bit accumulators (ACC0–ACC3)
    - – Eight 8-bit accumulator extension bytes (two per accumulator). These are grouped into two 32-bit values for load and store operations (ACCEXT01 and ACCEXT23).

Accumulators and extension bytes can be loaded, copied, and stored; results from EMAC arithmetic operations generally affect the entire 48-bit destination.

— One 16-bit mask register (MASK)

— One 32-bit Status register (MACSR) including four indicator bits signaling product or accumulation overflow (one for each accumulator: PAV0–PAV3)

The supervisor programming model is to be used only by system control software to implement restricted operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire processors are in the supervisor programming model, that consists of registers available in user mode as well as the following control registers:

- 16-bit status register (SR)
- 32-bit supervisor stack pointer (SSP)
- 32-bit vector base register (VBR)

- Two 32-bit memory base address registers (RAMBAR, FLASHBAR)

Table 3-1. ColdFire Core Programming Model

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
Supervisor/User Access Registers						
Load: 0x080 Store: 0x180	Data Register 0 (D0)	32	R/W	0xCF20_60	No	3.2.1/3-5
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x10A0_1070	No	3.2.1/3-5
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	3.2.1/3-5
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	3.2.2/3-5
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	3.2.3/3-6
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	4.2.1/4-4
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	4.2.2/4-6
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	4.2.3/4-8
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	4.2.4/4-8
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	4.2.4/4-8
0x80E	Condition Code Register (CCR)	8	R/W	Undefined	No	3.2.4/3-7
0x80F	Program Counter (PC)	32	R/W	Contents of location 0x0000_0004	No	3.2.5/3-8

Table 3-1. ColdFire Core Programming Model (continued)

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
Supervisor Access Only Registers						
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x0000_0000	No	3.2.3/3-6
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.6/3-8
0x80E	Status Register (SR)	16	R/W	0x27--	No	3.2.7/3-8
0xC04	Flash Base Address Register (FLASHBAR)	32	R/W	0x0000_0000	Yes	3.2.8/3-9
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	3.2.8/3-9

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 31, “Debug Module”.

3.2.1 Data Registers (D0–D7)

D0–D7 data registers are for bit (1-bit), byte (8-bit), word (16-bit) and longword (32-bit) operations; they can also be used as index registers.

NOTE

Registers D0 and D1 contain hardware configuration details after reset. See Section 3.3.4.15, “Reset Exception” for more details.

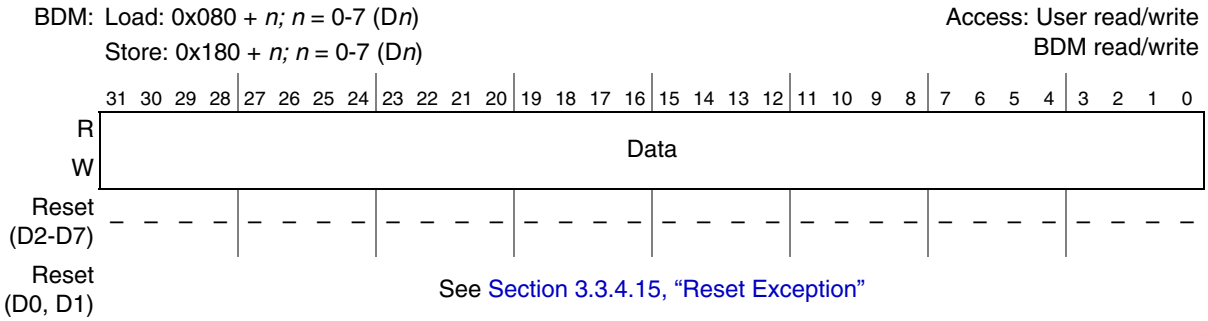


Figure 3-2. Data Registers (D0–D7)

3.2.2 Address Registers (A0–A6)

These registers can be used as software stack pointers, index registers, or base address registers. They can also be used for word and longword operations.

BDM: Load: 0x088 + *n*; *n* = 0–6 (A*n*)
Store: 0x188 + *n*; *n* = 0–6 (A*n*)

Access: User read/write
BDM read/write

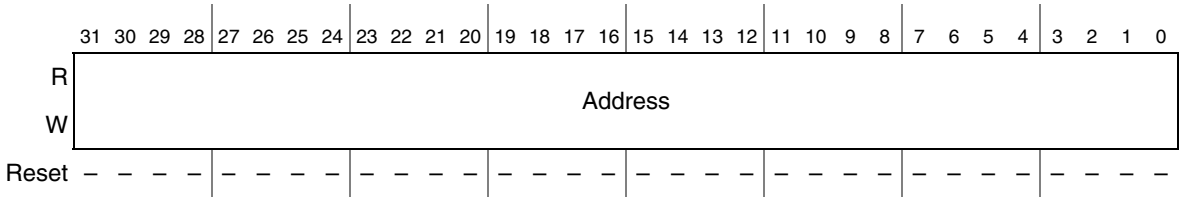


Figure 3-3. Address Registers (A0–A6)

3.2.3 Supervisor/User Stack Pointers (A7 and OTHER_A7)

The ColdFire architecture supports two independent stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two program-visible 32-bit registers does not identify one as the SSP and the other as the USP. Instead, the hardware uses one 32-bit register as the active A7 and the other as OTHER_A7. Thus, the register contents are a function of the processor operation mode, as shown in the following:

```
if SR[S] = 1
        then    A7 = Supervisor Stack Pointer
                OTHER_A7 = User Stack Pointer
        else    A7 = User Stack Pointer
                OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports direct reads and writes to the (active) A7 and OTHER_A7. It is the responsibility of the external development system to determine, based on the setting of SR[S], the mapping of A7 and OTHER_A7 to the two program-visible definitions (SSP and USP).

To support dual stack pointers, the following two supervisor instructions are included in the ColdFire instruction set architecture to load/store the USP:

```
move.l Ay,USP;move to USP
move.l USP,Ax;move from USP
```

These instructions are described in the *ColdFire Family Programmer's Reference Manual*. All other instruction references to the stack pointer, explicit or implicit, access the active A7 register.

NOTE

The USP must be initialized using the `move.l Ay,USP` instruction before any entry into user mode.

The SSP is loaded during reset exception processing with the contents of location 0x0000_0000.

BDM: Load: 0x08F (A7)
Store: 0x18F (A7)
0x800 (OTHER_A7)

Access: A7: User or BDM read/write
OTHER_A7: Supervisor or BDM read/write

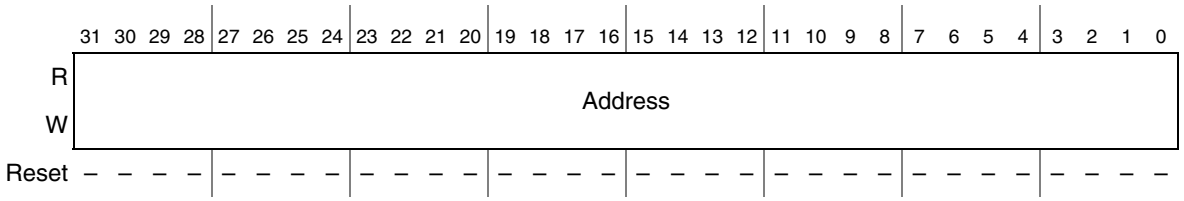


Figure 3-4. Stack Pointer Registers (A7 and OTHER_A7)

3.2.4 Condition Code Register (CCR)

The CCR is the LSB of the processor status register (SR). Bits 4–0 act as indicator flags for results generated by processor operations. The extend bit (X) is also an input operand during multiprecision arithmetic computations.

NOTE

The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

BDM: LSB of Status Register (SR)

Access: User read/write
BDM read/write

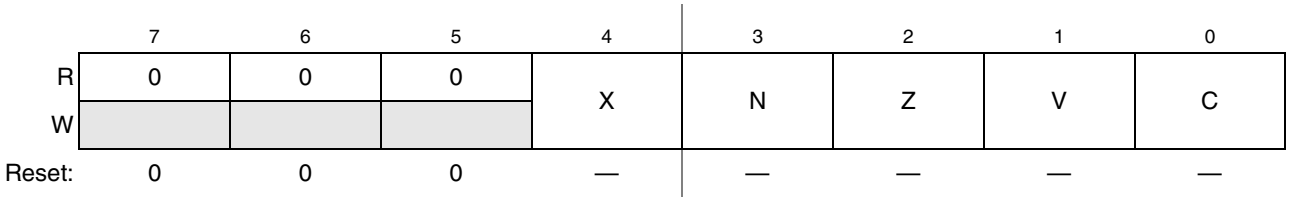


Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4 X	Extend condition code bit. Set to the C-bit value for arithmetic operations; otherwise not affected or set to a specified result.
3 N	Negative condition code bit. Set if most significant bit of the result is set; otherwise cleared.
2 Z	Zero condition code bit. Set if result equals zero; otherwise cleared.
1 V	Overflow condition code bit. Set if an arithmetic overflow occurs implying the result cannot be represented in operand size; otherwise cleared.
0 C	Carry condition code bit. Set if a carry out of the operand msb occurs for an addition or if a borrow occurs in a subtraction; otherwise cleared.

3.2.5 Program Counter (PC)

The PC contains the currently executing instruction address. During instruction execution and exception processing, the processor automatically increments PC contents or places a new value in the PC. The PC is a base address for PC-relative operand addressing.

The PC is initially loaded during reset exception processing with the contents at location 0x0000_0004.

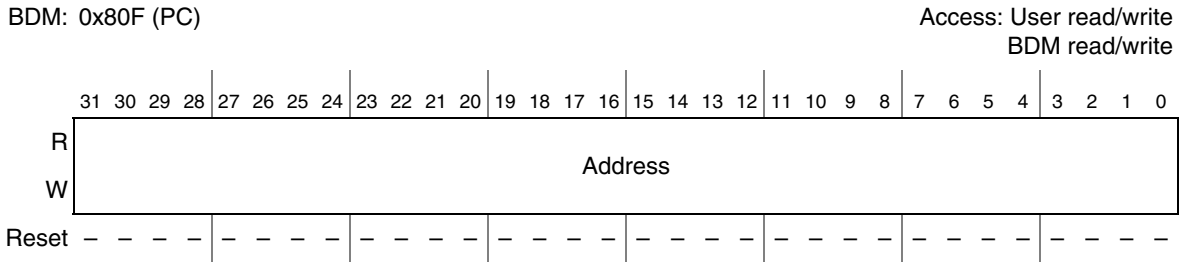


Figure 3-6. Program Counter Register (PC)

3.2.6 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in the memory. To access the vector table, the displacement of an exception vector is added to the value in VBR. The lower 20 bits of the VBR are not implemented by ColdFire processors. They are assumed to be zero, forcing the table to be aligned on a 1 MB boundary.

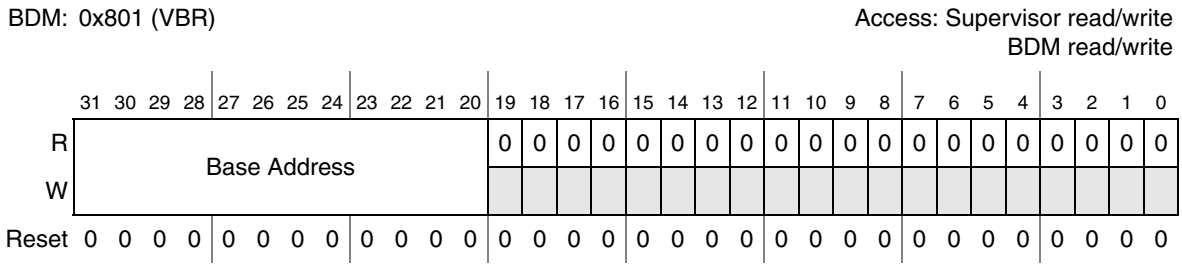


Figure 3-7. Vector Base Register (VBR)

3.2.7 Status Register (SR)

The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T bit), supervisor or user mode (S bit), and master or interrupt state (M bit). All defined bits in the SR have read/write access when in supervisor mode.

NOTE

The lower byte of the SR (the CCR) must be loaded explicitly after reset and before any compare (CMP), Bcc, or Scc instructions execute.

BDM: 0x80E (SR)

Access: Supervisor read/write
BDM read/write

	System Byte								Condition Code Register (CCR)							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	T	0	S	M	0	I			0	0	0	X	N	Z	V	C
W																
Reset	0	0	1	0	0	1	1	1	0	0	0	—	—	—	—	—

Figure 3-8. Status Register (SR)

Table 3-3. SR Field Descriptions

Field	Description
15 T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	Reserved, must be cleared.
13 S	Supervisor/user state. 0 User mode 1 Supervisor mode
12 M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	Reserved, must be cleared.
10–8 I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0 CCR	Refer to Section 3.2.4, “Condition Code Register (CCR)”.

3.2.8 Memory Base Address Registers (RAMBAR, FLASHBAR)

The memory base address registers are used to specify the base address of the internal SRAM and flash modules and indicate the types of references mapped to each. Each base address register includes a base address, write-protect bit, address space mask bits, and an enable bit. FLASHBAR determines the base address of the on-chip flash, and RAMBAR determines the base address of the on-chip RAM. For more information, refer to Section 11.2.1, “SRAM Base Address Register (RAMBAR)” and Section 18.3.2, “Flash Base Address Register (FLASHBAR)”.

3.3 Functional Description

3.3.1 Version 2 ColdFire Microarchitecture

From the block diagram in Figure 3-1, the non-Harvard architecture of the processor is readily apparent. The processor interfaces to the local memory subsystem via a single 32-bit address and two unidirectional 32-bit data buses. This structure minimizes the core size without compromising performance to a large degree.

A more detailed view of the hardware structure within the two pipelines is presented in Figure 3-9 and Figure 3-10 below. In these diagrams, the internal structure of the instruction fetch and operand execution pipelines is shown:

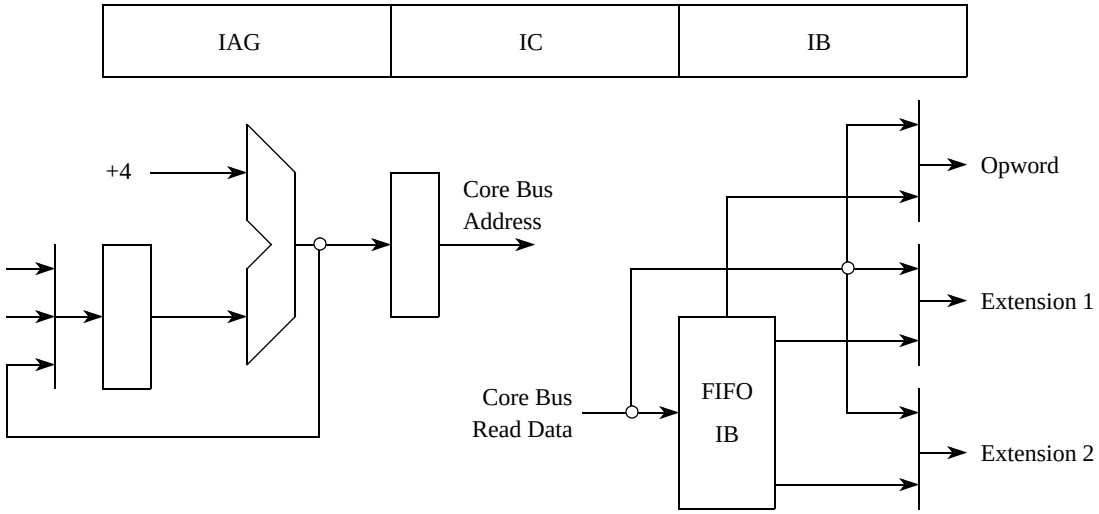


Figure 3-9. Version 2 ColdFire Processor Instruction Fetch Pipeline Diagram

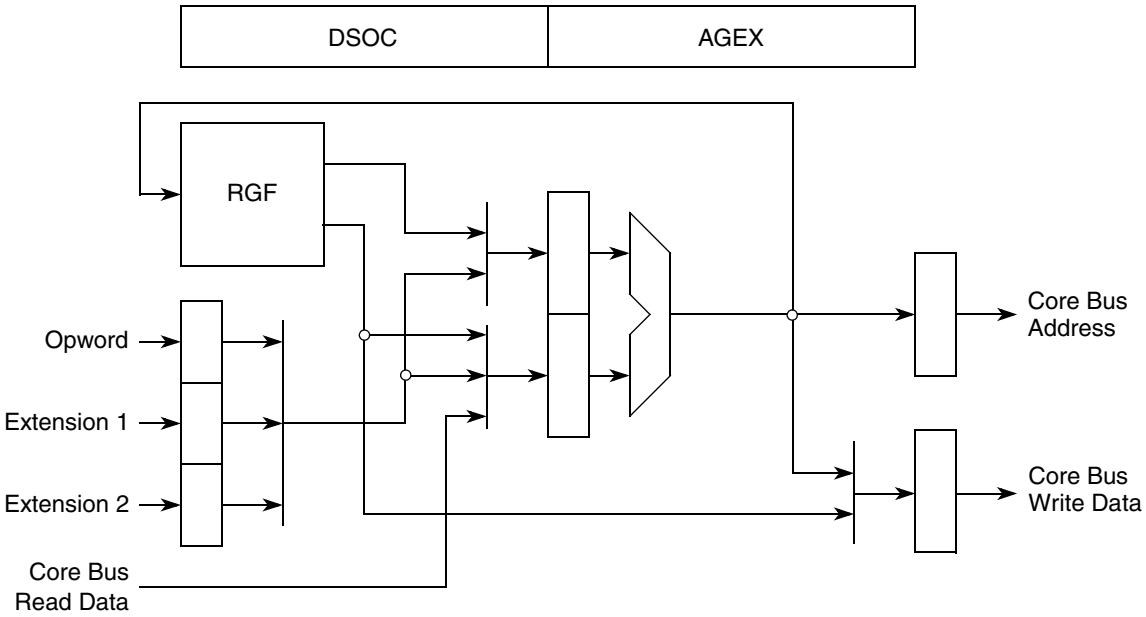


Figure 3-10. Version 2 ColdFire Processor Operand Execution Pipeline Diagram

The instruction fetch pipeline prefetches instructions from local memory using a two-stage structure. For sequential prefetches, the next instruction address is generated by adding four to the last prefetch address. This function is performed during the IAG stage and the resulting prefetch address gated onto the core bus (if there are no pending operand memory accesses assigned a higher priority). After the prefetch address is driven onto the core bus, the instruction fetch cycle accesses the appropriate local memory and returns the instruction read data back to the IFP during the cycle. If the accessed data is not present in a local memory (e.g., an instruction cache miss, or an external access cycle is required), the IFP is stalled in the

IC stage until the referenced data is available. As the prefetch data arrives in the IFP, it can be loaded into the FIFO instruction buffer or gated directly into the OEP.

The V2 design uses a simple static conditional branch prediction algorithm (forward-assumed as not-taken, backward-assumed as taken), and all change-of-flow operations are calculated by the OEP and the target instruction address fed back to the IFP.

The IFP and OEP are decoupled by the FIFO instruction buffer, allowing instruction prefetching to occur with the available core bus bandwidth not used for operand memory accesses. For the V2 design, the instruction buffer contains three 32-bit locations.

Consider the operation of the OEP for three basic classes of non-branch instructions:

- Register-to-register:
  ```
  op    Ry,Rx
  ```
- Embedded load:
  ```
  op    <mem>y,Rx
  ```
- Register-to-memory (store)
  ```
  move    Ry,<mem>x
  ```

For simple register-to-register instructions, the first stage of the OEP performs the instruction decode and fetching of the required register operands (OC) from the dual-ported register file, while the actual instruction execution is performed in the second stage (EX) in one of the execute engines (e.g., ALU, barrel shifter, divider, EMAC). There are no operand memory accesses associated with this class of instructions, and the execution time is typically a single machine cycle. See Figure 3-11.

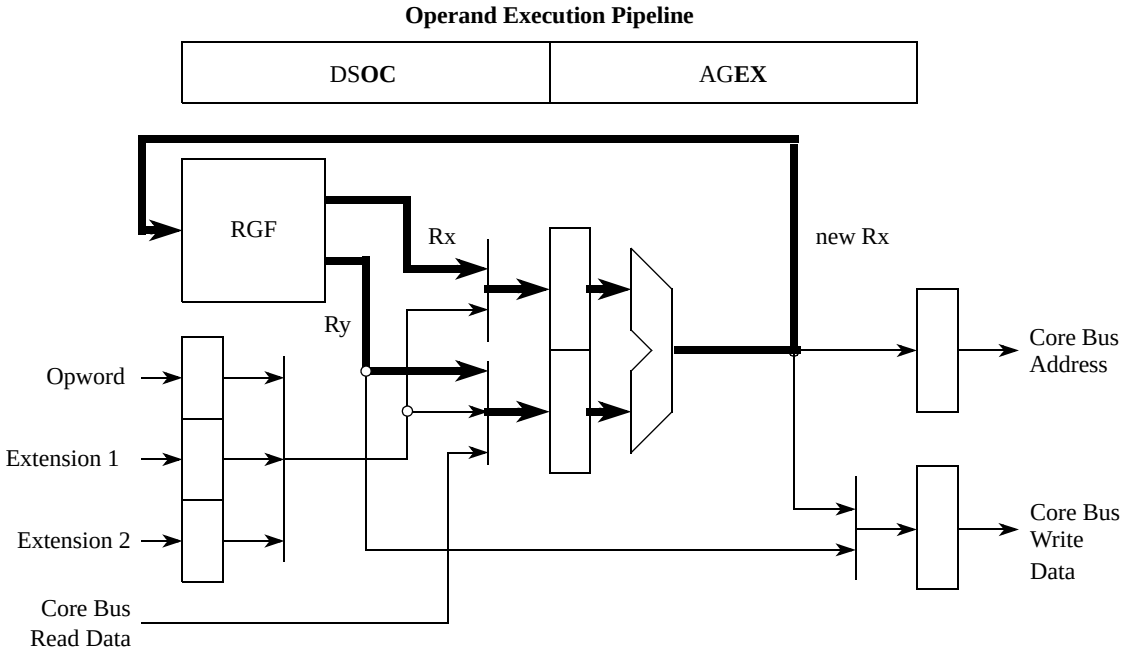


Figure 3-11. V2 OEP Register-to-Register

For memory-to-register (embedded-load) instructions, the instruction is effectively staged through the OEP twice with a basic execution time of three cycles. First, the instruction is decoded and the components

of the operand address (base register from the RGF and displacement) are selected (DS). Second, the operand effective address is generated using the ALU execute engine (AG). Third, the memory read operand is fetched from the core bus, while any required register operand is simultaneously fetched (OC) from the RGF. Finally, in the fourth cycle, the instruction is executed (EX). The heavily-used 32-bit load instruction (`move.l <mem>y,Rx`) is optimized to support a two-cycle execution time. The following example in Figure 3-12 shows an effective address of the form <ea>y = (d16,Ay), i.e., a 16-bit signed displacement added to a base register Ay.

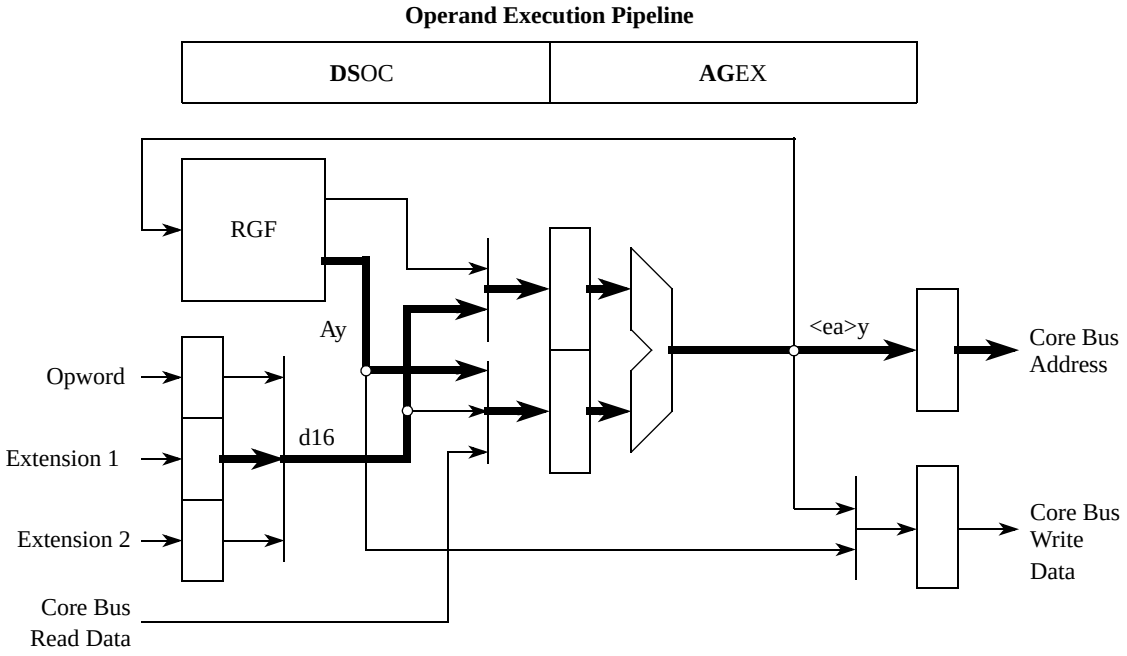


Figure 3-12. V2 OEP Embedded-Load Part 1

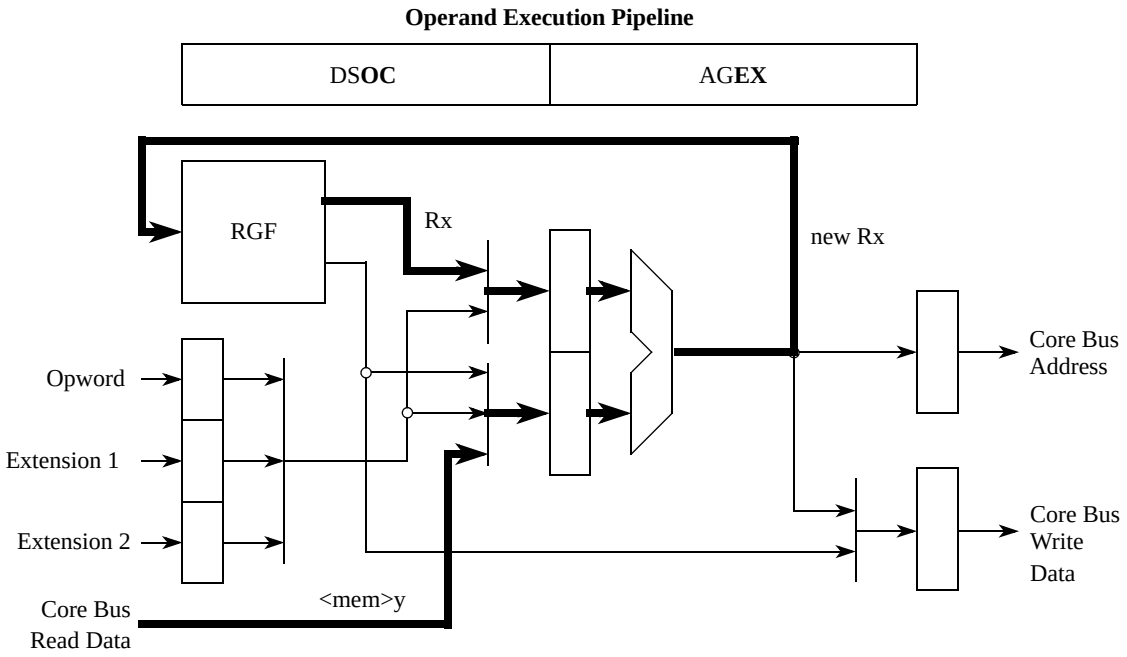


Figure 3-13. V2 OEP Embedded-Load Part 2

For register-to-memory (store) operations, the stage functions (DS/OC, AG/EX) are effectively performed simultaneously allowing single-cycle execution. See Figure 3-14 where the effective address is of the form <ea>x = (d16,Ax), i.e., a 16-bit signed displacement added to a base register Ax.

For read-modify-write instructions, the pipeline effectively combines an embedded-load with a store operation for a three-cycle execution time.

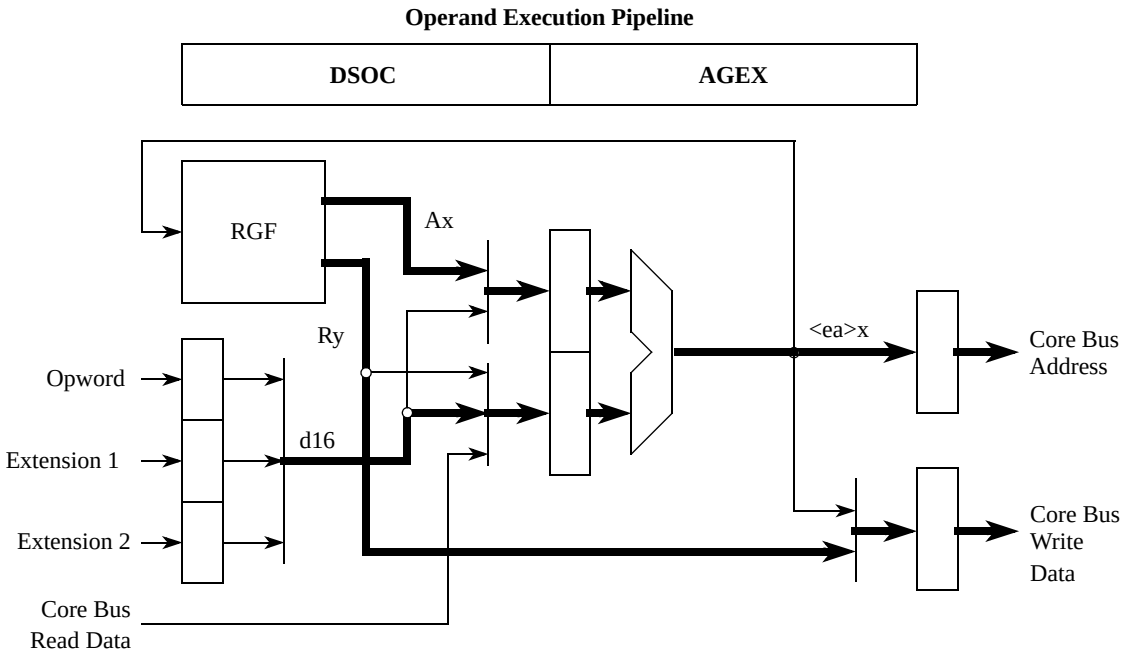


Figure 3-14. V2 OEP Register-to-Memory

The pipeline timing diagrams of Figure 3-15 depict the execution templates for these three classes of instructions. In these diagrams, the x-axis represents time, and the various instruction operations are shown progressing down the operand execution pipeline.

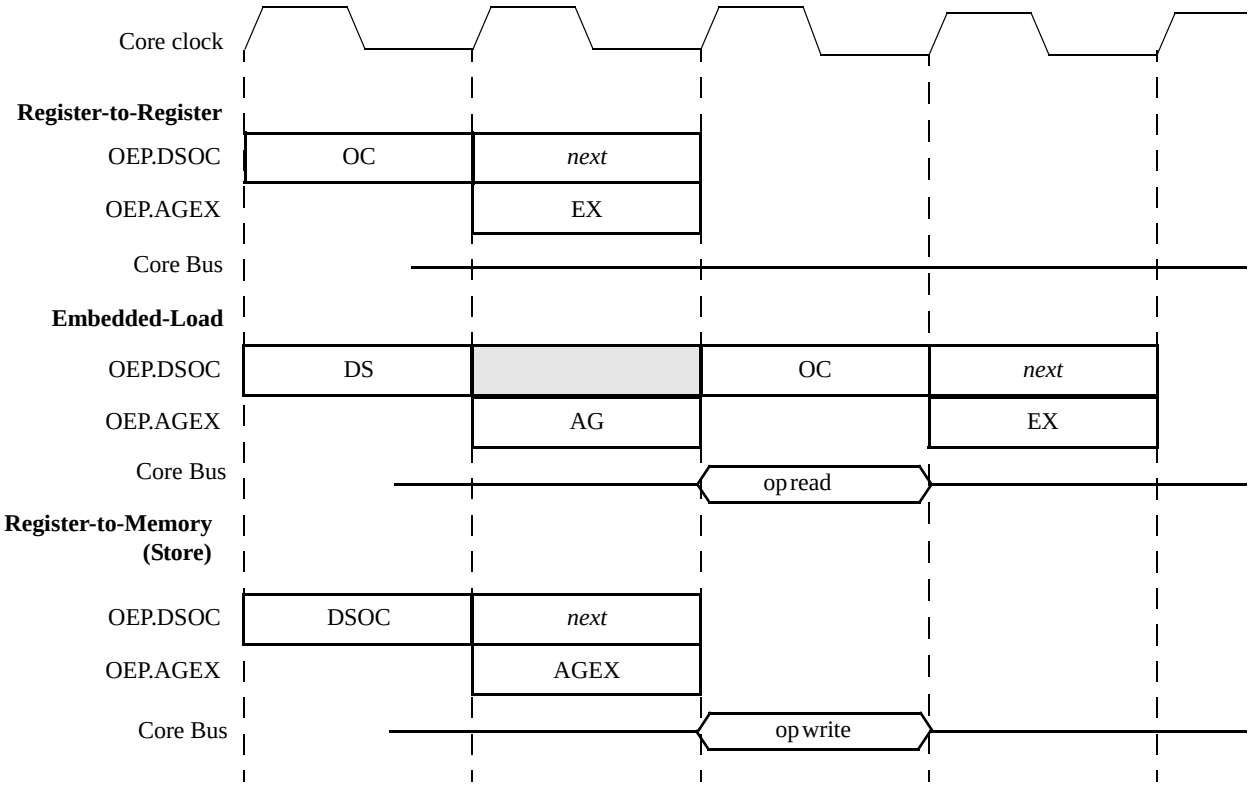


Figure 3-15. V2 OEP Pipeline Execution Templates

3.3.2 Instruction Set Architecture (ISA_A+)

The original ColdFire instruction set architecture (ISA_A) was derived from the M68000 family opcodes based on extensive analysis of embedded application code. The ISA was optimized for code compiled from high-level languages where the dominant operand size was the 32-bit integer declaration. This approach minimized processor complexity and cost, while providing excellent performance for compiled applications.

After the initial ColdFire compilers were created, developers noted there were certain ISA additions that would enhance code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they found certain frequently-used instruction sequences that could be improved by the creation of additional instructions.

The original ISA definition minimized support for instructions referencing byte- and word-sized operands. Full support for the move byte and move word instructions was provided, but the only other opcodes supporting these data types are CLR (clear) and TST (test). A set of instruction enhancements has been implemented in subsequent ISA revisions, ISA_B and ISA_C. The added opcodes primarily addressed three areas:

1. Enhanced support for byte and word-sized operands
2. Enhanced support for position-independent code
3. Miscellaneous instruction additions to address new functionality

Table 3-4 summarizes the instructions added to revision ISA_A to form revision ISA_A+. For more details see the *ColdFire Family Programmer's Reference Manual*.

Table 3-4. Instruction Enhancements over Revision ISA_A

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; new Dn[31] equals old Dn[0], new Dn[30] equals old Dn[1],..., new Dn[0] equals old Dn[31].
BYTEREV	The contents of the destination data register are byte-reversed; new Dn[31:24] equals old Dn[7:0],..., new Dn[7:0] equals old Dn[31:24].
FF1	The data register, Dn, is scanned, beginning from the most-significant bit (Dn[31]) and ending with the least-significant bit (Dn[0]), searching for the first set bit. The data register is then loaded with the offset count from bit 31 where the first set bit appears.
Move from USP	USP → Destination register
Move to USP	Source register → USP
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

3.3.3 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. The ColdFire processors differ from the M68000 family because they include:

- A simplified exception vector table
- Reduced relocation capabilities using the vector-base register
- A single exception stack frame format

All ColdFire processors use an instruction restart exception model. However, Version 2 ColdFire processors require more software support to recover from certain access errors. See Section 3.3.4.1, "Access Error Exception" for details.

Exception processing includes all actions from fault condition detection to the initiation of fetch for first handler instruction. Exception processing is comprised of four major steps:

1. The processor makes an internal copy of the SR and then enters supervisor mode by setting the S bit and disabling trace mode by clearing the T bit. The interrupt exception also forces the M bit to be cleared and the interrupt priority mask to set to current interrupt request level.

2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from the interrupt controller. The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address.

3. The processor saves the current context by creating an exception stack frame on the system stack. The exception stack frame is created at a 0-modulo-4 address on top of the system stack pointed to by the supervisor stack pointer (SSP). As shown in Figure 3-16, the processor uses a simplified fixed-length stack frame for all exceptions. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).
4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 MB boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 × vector number). After the exception vector has been fetched, the vector contents determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has initiated, exception processing terminates and normal instruction processing continues in the handler.

All ColdFire processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see Table 3-5).

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. See Chapter 15, “Interrupt Controller Module” for details on the device-specific interrupt sources.

Table 3-5. Exception Vector Assignments

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented line-F opcode
12	0x030	Next	Debug interrupt
13	0x034	—	Reserved
14	0x038	Fault	Format error
15–23	0x03C–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	—	Reserved

Table 3-5. Exception Vector Assignments (continued)

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–63	0x0C0–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	Device-specific interrupts

1 Fault refers to the PC of the instruction that caused the exception. Next refers to the PC of the instruction that follows the instruction that caused the fault.

All ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to disable interrupts effectively, if necessary, by raising the interrupt mask level contained in the status register. In addition, the ISA_A+ architecture includes an instruction (STLDSR) that stores the current interrupt mask level and loads a value into the SR. This instruction is specifically intended for use as the first instruction of an interrupt service routine that services multiple interrupt requests with different interrupt levels. For more details, see *ColdFire Family Programmer's Reference Manual*.

3.3.3.1 Exception Stack Frame Definition

Figure 3-16 shows exception stack frame. The first longword contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.

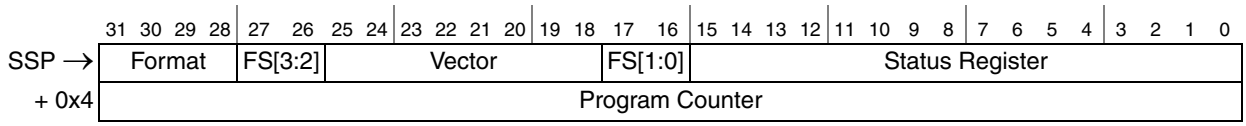


Figure 3-16. Exception Stack Frame Form

The 16-bit format/vector word contains three unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of 4, 5, 6, or 7 by the processor, indicating a two-longword frame format. See Table 3-6.

Table 3-6. Format Field Encodings

Original SSP @ Time of Exception, Bits 1:0	SSP @ 1st Instruction of Handler	Format Field
00	Original SSP - 8	0100
01	Original SSP - 9	0101
10	Original SSP - 10	0110
11	Original SSP - 11	0111

- There is a 4-bit fault status field, FS[3:0], at the top of the system stack. This field is defined for access and address errors only and written as zeros for all other exceptions. See Table 3-7.

Table 3-7. Fault Status Encodings

FS[3:0]	Definition
00*xx*	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the interrupt controller in case of an interrupt. See Table 3-5.

3.3.4 Processor Exceptions

3.3.4.1 Access Error Exception

The exact processor response to an access error depends on the memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults during instruction prefetches followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction is aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, (for example, (An)+,-(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during a MOVEM instruction loading from memory, any registers already updated before the fault occurs contain the operands from memory.

The V2 ColdFire processor uses an imprecise reporting mechanism for access errors on operand writes. Because the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its

execution until all previous operations, including all pending write operations, are complete. If any previous write terminates with an access error, it is guaranteed to be reported on the NOP instruction.

3.3.4.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (if bit 0 of the target address is set) results in an address error exception.

Any attempted use of a word-sized index register (Xn.w) or a scale factor of eight on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode, which is defined by bit 8 of extension word 1 being set.

If an address error occurs on a JSR instruction, the Version 2 ColdFire processor calculates the target address then the return address is pushed onto the stack. If an address error occurs on an RTS instruction, the Version 2 ColdFire processor overwrites the faulting return PC with the address error stack frame.

3.3.4.3 Illegal Instruction Exception

The ColdFire variable-length instruction set architecture supports three instruction sizes: 16, 32, or 48 bits. The first instruction word is known as the operation word (or opword), while the optional words are known as extension word 1 and extension word 2. The opword is further subdivided into three sections: the upper four bits segment the entire ISA into 16 instruction lines, the next 6 bits define the operation mode (opmode), and the low-order 6 bits define the effective address. See Figure 3-17. The opword line definition is shown in Table 3-8.

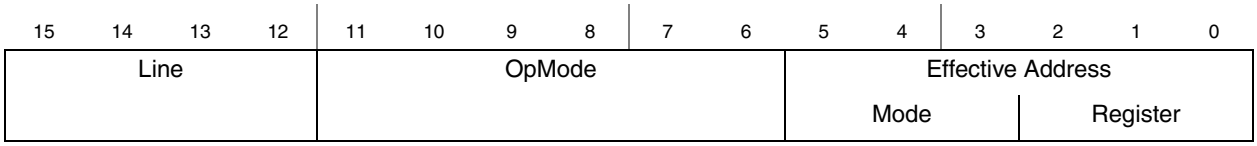


Figure 3-17. ColdFire Instruction Operation Word (Opword) Format

Table 3-8. ColdFire Opword Line Definition

Opword[Line]	Instruction Class
0x0	Bit manipulation, Arithmetic and Logical Immediate
0x1	Move Byte
0x2	Move Long
0x3	Move Word
0x4	Miscellaneous
0x5	Add (ADDQ) and Subtract Quick (SUBQ), Set according to Condition Codes (Scc)
0x6	PC-relative change-of-flow instructions Conditional (Bcc) and unconditional (BRA) branches, subroutine calls (BSR)
0x7	Move Quick (MOVEQ), Move with sign extension (MVS) and zero fill (MVZ)
0x8	Logical OR (OR)
0x9	Subtract (SUB), Subtract Extended (SUBX)

Table 3-8. ColdFire Opword Line Definition (continued)

Opword[Line]	Instruction Class
0xA	EMAC, Move 3-bit Quick (MOV3Q)
0xB	Compare (CMP), Exclusive-OR (EOR)
0xC	Logical AND (AND), Multiply Word (MUL)
0xD	Add (ADD), Add Extended (ADDX)
0xE	Arithmetic and logical shifts (ASL, ASR, LSL, LSR)
0xF	Cache Push (CPUSHL), Write DDATA (WDDATA), Write Debug (WDEBUG)

In the original M68000 ISA definition, lines A and F were effectively reserved for user-defined operations (line A) and co-processor instructions (line F). Accordingly, there are two unique exception vectors associated with illegal opwords in these two lines.

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any non-MAC line-A and most line-F opcodes generate their unique exception types, vector numbers 10 and 11, respectively. ColdFire cores do not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

3.3.4.4 Divide-By-Zero

Attempting to divide by zero causes an exception (vector 5, offset equal 0x014).

3.3.4.5 Privilege Violation

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception. See *ColdFire Programmer's Reference Manual* for a list of supervisor-mode instructions.

There is one special case involving the HALT instruction. Normally, this opcode is a supervisor mode instruction, but if the debug module's CSR[UHE] is set, then this instruction can be also be executed in user mode for debugging purposes.

3.3.4.6 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by setting of the SR[T] bit, the completion of an instruction execution (for all but the stop instruction) signals a trace exception. This functionality allows a debugger to monitor program execution.

The stop instruction has the following effects:

1. The instruction before the stop executes and then generates a trace exception. In the exception stack frame, the PC points to the stop opcode.
2. When the trace handler is exited, the stop instruction executes, loading the SR with the immediate operand from the instruction.

3. The processor then generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in the previous step.

If the processor is not in trace mode and executes a stop instruction where the immediate operand sets SR[T], hardware loads the SR and generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in step 2.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider a TRAP instruction execution while in trace mode. The processor initiates the trap exception and then passes control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the trap exception handler to check for this condition (SR[T] in the exception stack frame set) and pass control to the trace handler before returning from the original exception.

3.3.4.7 Unimplemented Line-A Opcode

A line-A opcode is defined when bits 15-12 of the opword are 0b1010. This exception is generated by the attempted execution of an undefined line-A opcode.

3.3.4.8 Unimplemented Line-F Opcode

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated when attempting to execute an undefined line-F opcode.

3.3.4.9 Debug Interrupt

See Chapter 31, “Debug Module,” for a detailed explanation of this exception, which is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12). Additionally, SR[M,I] are unaffected by the interrupt.

3.3.4.10 RTE and Format Error Exception

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire core, any attempted RTE execution (where the format is not equal to {4,5,6,7}) generates a format error. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from M68000 applications. On M68000 family processors, the SR was located at the top of the stack. On those processors, bit 30 of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted using this old format, it generates a format error on a ColdFire processor.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the second longword operand within the stack frame.

3.3.4.11 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls. The TRAP instruction may be used to change from user to supervisor mode.

3.3.4.12 Unsupported Instruction Exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor, an unsupported instruction exception is generated. The instruction functionality can then be emulated in the exception handler, if desired.

All ColdFire cores record the processor hardware configuration in the D0 register immediately after the negation of $\overline{\text{RESET}}$. See Section 3.3.4.15, “Reset Exception,” for details.

3.3.4.13 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle. See Chapter 15, “Interrupt Controller Module,” for details on the interrupt controller.

3.3.4.14 Fault-on-Fault Halt

If a ColdFire processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic fault-on-fault condition. A reset is required to to exit this state.

3.3.4.15 Reset Exception

Asserting the reset input signal ($\overline{\text{RESET}}$) to the processor causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when the reset input is recognized. Processing cannot be recovered.

The reset exception places the processor in the supervisor mode by setting the SR[S] bit and disables tracing by clearing the SR[T] bit. This exception also clears the SR[M] bit and sets the processor’s SR[I] field to the highest level (level 7, 0b111). Next, the VBR is initialized to zero (0x0000_0000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

NOTE

Other implementation-specific registers are also affected. Refer to each module in this reference manual for details on these registers.

After the processor is granted the bus, it performs two longword read-bus cycles. The first longword at address 0x0000_0000 is loaded into the supervisor stack pointer and the second longword at address 0x0000_0004 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault state.

ColdFire processors load hardware configuration information into the D0 and D1 general-purpose registers after system reset. The hardware configuration information is loaded immediately after the reset-in signal is negated. This allows an emulator to read out the contents of these registers via the BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in Figure 3-18.

BDM: Load: 0x080 (D0)
Store: 0x180 (D0)

Access: User read-only
BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PF								VER				REV			
W																
Reset	1	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	EMAC	FPU	0	0	0	0	ISA				DEBUG			
W																
Reset	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	1

Figure 3-18. D0 Hardware Configuration Info

Table 3-9. D0 Hardware Configuration Info Field Description

Field	Description
31–24 PF	Processor family. This field is fixed to a hex value of 0xCF indicating a ColdFire core is present.
23–20 VER	ColdFire core version number. Defines the hardware microarchitecture version of ColdFire core. 0001 V1 ColdFire core 0010 V2 ColdFire core (This is the value used for this device.) 0011 V3 ColdFire core 0100 V4 ColdFire core 0101 V5 ColdFire core Else Reserved for future use
19–16 REV	Processor revision number. The default is 0b0000.
15 MAC	MAC present. This bit signals if the optional multiply-accumulate (MAC) execution engine is present in processor core. 0 MAC execute engine not present in core. (This is the value used for this device.) 1 MAC execute engine is present in core.
14 DIV	Divide present. This bit signals if the hardware divider (DIV) is present in the processor core. 0 Divide execute engine not present in core. 1 Divide execute engine is present in core. (This is the value used for this device.)
13 EMAC	EMAC present. This bit signals if the optional enhanced multiply-accumulate (EMAC) execution engine is present in processor core. 0 EMAC execute engine not present in core. 1 EMAC execute engine is present in core. (This is the value used for this device.)
12 FPU	FPU present. This bit signals if the optional floating-point (FPU) execution engine is present in processor core. 0 FPU execute engine not present in core. (This is the value used for this device.) 1 FPU execute engine is present in core.

Table 3-9. D0 Hardware Configuration Info Field Description (continued)

Field	Description
11–8	Reserved.
7–4 ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0000 ISA_A 0001 ISA_B 0010 ISA_C 1000 ISA_A+ (This is the value used for this device.) Else Reserved
3–0 DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 0000 DEBUG_A 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 1001 DEBUG_B+ (This is the value used for this device.) 1011 DEBUG_D+ 1111 DEBUG_D+PST Buffer Else Reserved

Information loaded into D1 defines the local memory hardware configuration as shown in the figure below.

BDM: Load: 0x1 (D1)
Store: 0x1 (D1)

Access: User read-only
BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CLSZ		CCAS		CCSZ				FLASHSZ					0	0	0
W																
Reset	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0

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

Figure 3-19. D1 Hardware Configuration Info

Table 3-10. D1 Hardware Configuration Information Field Description

Field	Description
31–30 CLSZ	Cache line size. This field is fixed to a hex value of 0x0 indicating a 16-byte cache line size.
29–28 CCAS	Configurable cache associativity. 00 Four-way 01 Direct mapped (This is the value used for this device) Else Reserved for future use

Table 3-10. D1 Hardware Configuration Information Field Description (continued)

Field	Description
27–24 CCSZ	Configurable cache size. Indicates the amount of instruction/data cache.The cache configuration options available are 50% instruction/50% data, 100% instruction, or 100% data, and are specified in the CACR register. 0000 No configurable cache (This is the value used for this device) 0001 512 B configurable cache 0010 1 KB configurable cache 0011 2 KB configurable cache 0100 4 KB configurable cache 0101 8 KB configurable cache 0110 16 KB configurable cache 0111 32 KB configurable cache Else Reserved
23–19 FLASHSZ	Flash bank size. 00000-01110 No flash 10000 64 KB flash 10010 128 KB flash 10011 96 KB flash 10100 256 KB flash (This is the value used for this device) 10110 512 KB flash Else Reserved for future use
18–16	Reserved
15–14 MBSZ	Bus size. Defines the width of the ColdFire master bus datapath. 00 32-bit system bus datapath (This is the value used for this device) 01 64-bit system bus datapath Else Reserved
13–8	Reserved, resets to 0b01_0000
7–3 SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01111 24 KB 01110 32 KB (This is the value used for this device) 10000 64 KB 10010 128 KB Else Reserved for future use
2–0	Reserved.

3.3.5 Instruction Execution Timing

This section presents processor instruction execution times in terms of processor-core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of processor clock cycles. Each timing entry is presented as C(R/W) where:

- C is the number of processor clock cycles, including all applicable operand fetches and writes, and all internal core cycles required to complete the instruction execution.
- R/W is the number of operand reads (R) and writes (W) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes the assumptions concerning the timing values and the execution time details.

3.3.5.1 Timing Assumptions

For the timing data presented in this section, these assumptions apply:

1. The OEP is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the IFP to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. The most common example of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as busy for two clock cycles after the final decode and select/operand fetch cycle (DSOC) of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it is stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is two cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.
3. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
4. All operand data accesses are aligned on the same byte boundary as the operand size; for example, 16-bit operands aligned on 0-modulo-2 addresses, 32-bit operands aligned on 0-modulo-4 addresses.

   The processor core decomposes misaligned operand references into a series of aligned accesses as shown in Table 3-11.

Table 3-11. Misaligned Operand References

address[1:0]	Size	Bus Operations	Additional C(R/W)
01 or 11	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
01 or 11	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

3.3.5.2 MOVE Instruction Execution Times

Table 3-12 lists execution times for MOVE.{B,W} instructions; Table 3-13 lists timings for MOVE.L.

NOTE

For all tables in this section, the execution time of any instruction using the PC-relative effective addressing modes is the same for the comparable An-relative mode.

ET with {<ea> = (d16,PC)}	equals ET with {<ea> = (d16,An)}
ET with {<ea> = (d8,PC,Xi*SF)}	equals ET with {<ea> = (d8,An,Xi*SF)}

The nomenclature xxx.wl refers to both forms of absolute addressing, xxx.w and xxx.l.

Table 3-12. MOVE Byte and Word Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
(Ay)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
-(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
(d16,Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,Ay,Xi*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d16,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,PC,Xi*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1))	—	—	—
#xxx	1(0/0)	3(0/1)	3(0/1)	3(0/1)	—	—	—

Table 3-13. MOVE Long Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—

Table 3-13. MOVE Long Execution Times (continued)

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(d8,Ay,Xi*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#xxx	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

3.3.5.3 Standard One Operand Instruction Execution Times

Table 3-14. One Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BITREV	Dx	1(0/0)	—	—	—	—	—	—	—
BYTEREV	Dx	1(0/0)	—	—	—	—	—	—	—
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
EXT.W	Dx	1(0/0)	—	—	—	—	—	—	—
EXT.L	Dx	1(0/0)	—	—	—	—	—	—	—
EXTB.L	Dx	1(0/0)	—	—	—	—	—	—	—
FF1	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TST.B	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.W	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.L	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

3.3.5.4 Standard Two Operand Instruction Execution Times

Table 3-15. Two Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
ADD.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
ADD.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ADDQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
AND.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
AND.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ANDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ASL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
BCHG	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCHG	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCLR	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BSET	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
BTST	#imm,<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	—	—	—
CMP.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
DIVS.W	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
DIVU.W	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
DIVS.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
DIVU.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
EOR.L	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
EORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
LEA	<ea>,Ax	—	1(0/0)	—	—	1(0/0)	2(0/0)	1(0/0)	—
LSL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
LSR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVEQ.L	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
OR.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
OR.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—

Table 3-15. Two Operand Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
REMS.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
REMU.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
SUB.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
SUB.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUBQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

3.3.5.5 Miscellaneous Instruction Execution Times

Table 3-16. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
CPUSHL	(Ax)	—	11(0/1)	—	—	—	—	—	—
LINK.W	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
MOVE.L	Ay,USP	3(0/0)	—	—	—	—	—	—	—
MOVE.L	USP,Ax	3(0/0)	—	—	—	—	—	—	—
MOVE.W	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,CCR	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.W	SR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) [2]
MOVEC	Ry,Rc	9(0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>, and list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
NOP		3(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	2(0/1)	—	—	2(0/1) [4]	3(0/1) [5]	2(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
STOP	#imm	—	—	—	—	—	—	—	3(0/0) [3]
TRAP	#imm	—	—	—	—	—	—	—	15(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPF.L		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2(1/0)	—	—	—	—	—	—	—

Table 3-16. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
WDDATA	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
WDEBUG	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

[1]The n is the number of registers moved by the MOVEM opcode.
[2]If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).
[3]The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.
[4]PEA execution times are the same for (d16,PC).
[5]PEA execution times are the same for (d8,PC,Xn*SF).

3.3.5.6 EMAC Instruction Execution Times

Table 3-17. EMAC Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.L	Ry, Rx, <ea>, Rw, Raccx	—	(1/0)	(1/0)	(1/0)	(1/0)[1]	—	—	—
MAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.W	Ry, Rx, <ea>, Rw, Raccx	—	(1/0)	(1/0)	(1/0)	(1/0)[1]	—	—	—
MOVE.L	<ea>y, Raccx	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccy, Raccx	1(0/0)	—	—	—	—	—	—	—
MOVE.L	<ea>y, MACSR	5(0/0)	—	—	—	—	—	—	5(0/0)
MOVE.L	<ea>y, Rmask	4(0/0)	—	—	—	—	—	—	4(0/0)
MOVE.L	<ea>y,Raccext01	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	<ea>y,Raccext23	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccx, <ea>x	1(0/0)[2]	—	—	—	—	—	—	—
MOVE.L	MACSR, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext01,<ea.x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext23,<ea>x	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MSAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, <ea>, Rw, Raccx	—	(1/0)	(1/0)	(1/0)	(1/0)[1]	—	—	—
MSAC.W	Ry, Rx, <ea>, Rw, Raccx	—	(1/0)	(1/0)	(1/0)	(1/0)[1]	—	—	—

Table 3-17. EMAC Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MULS.L	<ea>y, Dx	4(0/0)	(1/0)	(1/0)	(1/0)	(1/0)	—	—	—
MULS.W	<ea>y, Dx	4(0/0)	(1/0)	(1/0)	(1/0)	(1/0)	(1/0)	(1/0)	4(0/0)
MULU.L	<ea>y, Dx	4(0/0)	(1/0)	(1/0)	(1/0)	(1/0)	—	—	—
MULU.W	<ea>y, Dx	4(0/0)	(1/0)	(1/0)	(1/0)	(1/0)	(1/0)	(1/0)	4(0/0)

1 Effective address of (d16,PC) not supported

2 Storing an accumulator requires one additional processor clock cycle when saturation is enabled, or fractional rounding is performed (MACSR[7:4] equals 1---, -11-, --11)

NOTE

The execution times for moving the contents of the Racc, Raccext[01,23], MACSR, or Rmask into a destination location <ea>x shown in this table represent the best-case scenario when the store instruction is executed and there are no load or M{S}AC instructions in the EMAC execution pipeline. In general, these store operations require only a single cycle for execution, but if preceded immediately by a load, MAC, or MSAC instruction, the depth of the EMAC pipeline is exposed and the execution time is four cycles.

3.3.5.7 Branch Instruction Execution Times

Table 3-18. General Branch Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	2(0/1)	—	—	—
BSR		—	—	—	—	3(0/1)	—	—	—
JMP	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
JSR	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
RTE		—	—	10(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

Table 3-19. Bcc Instruction Execution Times

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

Chapter 4
Enhanced Multiply-Accumulate Unit (EMAC)

4.1 Introduction

This chapter describes the functionality, microarchitecture, and performance of the enhanced multiply-accumulate (EMAC) unit in the ColdFire family of processors.

4.1.1 Overview

The EMAC design provides a set of DSP operations that can improve the performance of embedded code while supporting the integer multiply instructions of the baseline ColdFire architecture.

The MAC provides functionality in three related areas:

1. Signed and unsigned integer multiplication
2. Multiply-accumulate operations supporting signed and unsigned integer operands as well as signed, fixed-point, and fractional operands
3. Miscellaneous register operations

The ColdFire family supports two MAC implementations with different performance levels and capabilities. The original MAC features a three-stage execution pipeline optimized for 16-bit operands, with a 16x16 multiply array and a single 32-bit accumulator. The EMAC features a four-stage pipeline optimized for 32-bit operands, with a fully pipelined 32 × 32 multiply array and four 48-bit accumulators.

The first ColdFire MAC supported signed and unsigned integer operands and was optimized for 16x16 operations, such as those found in applications including servo control and image compression. As ColdFire-based systems proliferated, the desire for more precision on input operands increased. The result was an improved ColdFire MAC with user-programmable control to optionally enable use of fractional input operands.

EMAC improvements target three primary areas:

- Improved performance of 32 × 32 multiply operation.
- Addition of three more accumulators to minimize MAC pipeline stalls caused by exchanges between the accumulator and the pipeline’s general-purpose registers
- A 48-bit accumulation data path to allow a 40-bit product, plus 8 extension bits increase the dynamic number range when implementing signal processing algorithms

The three areas of functionality are addressed in detail in following sections. The logic required to support this functionality is contained in a MAC module (Figure 4-1).

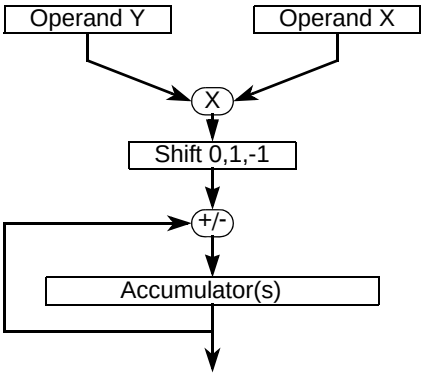


Figure 4-1. Multiply-Accumulate Functionality Diagram

4.1.1.1 Introduction to the MAC

The MAC is an extension of the basic multiplier in most microprocessors. It is typically implemented in hardware within an architecture and supports rapid execution of signal processing algorithms in fewer cycles than comparable non-MAC architectures. For example, small digital filters can tolerate some variance in an algorithm's execution time, but larger, more complicated algorithms such as orthogonal transforms may have more demanding speed requirements beyond scope of any processor architecture and may require full DSP implementation.

To balance speed, size, and functionality, the ColdFire MAC is optimized for a small set of operations that involve multiplication and cumulative additions. Specifically, the multiplier array is optimized for single-cycle pipelined operations with a possible accumulation after product generation. This functionality is common in many signal processing applications. The ColdFire core architecture is also modified to allow an operand to be fetched in parallel with a multiply, increasing overall performance for certain DSP operations.

Consider a typical filtering operation where the filter is defined as in Equation 4-1.

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k) \qquad \textit{Eqn. 4-1}$$

Here, the output y(i) is determined by past output values and past input values. This is the general form of an infinite impulse response (IIR) filter. A finite impulse response (FIR) filter can be obtained by setting coefficients a(k) to zero. In either case, the operations involved in computing such a filter are multiplies and product summing. To show this point, reduce Equation 4-1 to a simple, four-tap FIR filter, shown in Equation 4-2, in which the accumulated sum is a past data values and coefficients sum.

$$y(i) = \sum_{k=0}^{3} b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3) \qquad \textit{Eqn. 4-2}$$

4.2 Memory Map/Register Definition

The following table and sections explain the MAC registers:

Table 4-1. EMAC Memory Map

BDM[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	4.2.1/4-4
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	4.2.2/4-6
0x806	MAC Accumulator 0 (ACC0)	32	R/W	Undefined	4.2.3/4-8
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	4.2.4/4-8
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	4.2.4/4-8
0x809	MAC Accumulator 1 (ACC1)	32	R/W	Undefined	4.2.3/4-8
0x80A	MAC Accumulator 2 (ACC2)	32	R/W	Undefined	4.2.3/4-8
0x80B	MAC Accumulator 3 (ACC3)	32	R/W	Undefined	4.2.3/4-8

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 31, “Debug Module.”

4.2.1 MAC Status Register (MACSR)

The MAC status register (MACSR) contains a 4-bit operational mode field and condition flags. Operational mode bits control whether operands are signed or unsigned and whether they are treated as integers or fractions. These bits also control the overflow/saturation mode and the way in which rounding is performed. Negative, zero, and multiple overflow condition flags are also provided.

BDM: 0x804 (MACSR)

Access: Supervisor read/write
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAV*n*				OMC	S/U	F/I	R/T	N	Z	V	EV
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-2. MAC Status Register (MACSR)

Table 4-2. MACSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11–8 PAV*n*	Product/accumulation overflow flags. Contains four flags, one per accumulator, that indicate if past MAC or MSAC instructions generated an overflow during product calculation or the 48-bit accumulation. When a MAC or MSAC instruction is executed, the PAV*n* flag associated with the destination accumulator forms the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a `move.l, MACSR` instruction or the accumulator is loaded directly. Bit 11: Accumulator 3 ... Bit 8: Accumulator 0
7 OMC	Overflow saturation mode. Enables or disables saturation mode on overflow. If set, the accumulator is set to the appropriate constant (see S/U field description) on any operation that overflows the accumulator. After saturation, the accumulator remains unaffected by any other MAC or MSAC instructions until the overflow bit is cleared or the accumulator is directly loaded.
6 S/U	Signed/unsigned operations. **In integer mode:** S/U determines whether operations performed are signed or unsigned. It also determines the accumulator value during saturation, if enabled. 0 Signed numbers. On overflow, if OMC is enabled, an accumulator saturates to the most positive (0x7FFF_FFFF) or the most negative (0x8000_0000) number, depending on the instruction and the product value that overflowed. 1 Unsigned numbers. On overflow, if OMC is enabled, an accumulator saturates to the smallest value (0x0000_0000) or the largest value (0xFFFF_FFFF), depending on the instruction. **In fractional mode:** S/U controls rounding while storing an accumulator to a general-purpose register. 0 Move accumulator without rounding to a 16-bit value. Accumulator is moved to a general-purpose register as a 32-bit value. 1 The accumulator is rounded to a 16-bit value using the round-to-nearest (even) method when moved to a general-purpose register. See Section 4.3.1.1, “Rounding”. The resulting 16-bit value is stored in the lower word of the destination register. The upper word is zero-filled. This rounding procedure does not affect the accumulator value.
5 F/I	Fractional/integer mode. Determines whether input operands are treated as fractions or integers. 0 Integers can be represented in signed or unsigned notation, depending on the value of S/U. 1 Fractions are represented in signed, fixed-point, two’s complement notation. Values range from -1 to 1 - 2^{-15} for 16-bit fractions and -1 to 1 - 2^{-31} for 32-bit fractions. See Section 4.3.4, “Data Representation.”
4 R/T	Round/truncate mode. Controls rounding procedure for `move.l ACCx,Rx`, or MSAC.L instructions when in fractional mode. 0 Truncate. The product’s lsbs are dropped before it is combined with the accumulator. Additionally, when a store accumulator instruction is executed (`move.l ACCx,Rx`), the 8 lsbs of the 48-bit accumulator logic are truncated. 1 Round-to-nearest (even). The 64-bit product of two 32-bit, fractional operands is rounded to the nearest 40-bit value. If the low-order 24 bits equal 0x80_0000, the upper 40 bits are rounded to the nearest even (lsb = 0) value. See Section 4.3.1.1, “Rounding”. Additionally, when a store accumulator instruction is executed (`move.l ACCx,Rx`), the lsbs of the 48-bit accumulator logic round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared and MACSR[R/T] is set, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.

Table 4-2. MACSR Field Descriptions (continued)

Field	Description
3 N	Negative. Set if the msb of the result is set, otherwise cleared. N is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
2 Z	Zero. Set if the result equals zero, otherwise cleared. This bit is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
1 V	Overflow. Set if an arithmetic overflow occurs on a MAC or MSAC instruction, indicating that the result cannot be represented in the limited width of the EMAC. V is set only if a product overflow occurs or the accumulation overflows the 48-bit structure. V is evaluated on each MAC or MSAC operation and uses the appropriate PAV*n* flag in the next-state V evaluation.
0 EV	Extension overflow. Signals that the last MAC or MSAC instruction overflowed the 32 lsbs in integer mode or the 40 lsbs in fractional mode of the destination accumulator. However, the result remains accurately represented in the combined 48-bit accumulator structure. Although an overflow has occurred, the correct result, sign, and magnitude are contained in the 48-bit accumulator. Subsequent MAC or MSAC operations may return the accumulator to a valid 32/40-bit result.

Table 4-3 summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

Table 4-3. Summary of S/U, F/I, and R/T Control Bits

S/U	F/I	R/T	Operational Modes
0	0	x	Signed, integer
0	1	0	Signed, fractional Truncate on MAC.L and MSAC.L No round on accumulator stores
0	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-32-bits on accumulator stores
1	0	x	Unsigned, integer
1	1	0	Signed, fractional Truncate on MAC.L and MSAC.L Round-to-16-bits on accumulator stores
1	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-16-bits on accumulator stores

4.2.2 Mask Register (MASK)

The 32-bit MASK implements the low-order 16 bits to minimize the alignment complications involved with loading and storing only 16 bits. When the MASK is loaded, the low-order 16 bits of the source operand are actually loaded into the register. When it is stored, the upper 16 bits are all forced to ones.

This register performs a simple AND with the operand address for MAC instructions. The processor calculates the normal operand address and, if enabled, that address is then ANDed with {0xFFFF, MASK[15:0]} to form the final address. Therefore, with certain MASK bits cleared, the operand address

can be constrained to a certain memory region. This is used primarily to implement circular queues with the (An)+ addressing mode.

This minimizes the addressing support required for filtering, convolution, or any routine that implements a data array as a circular queue. For MAC + MOVE operations, the MASK contents can optionally be included in all memory effective address calculations. The syntax is as follows:

```
mac.sz  Ry,RxSF,<ea>y&,Rw
```

The & operator enables the MASK use and causes bit 5 of the extension word to be set. The exact algorithm for the use of MASK is:

```
if extension word, bit [5] = 1, the MASK bit, then
        if <ea> = (An)
                oa  =  An & {0xFFFF, MASK}

        if <ea> = (An)+
                oa  =  An
                An  = (An + 4) & {0xFFFF, MASK}

        if <ea> =-(An)
                oa  = (An - 4) & {0xFFFF, MASK}
                An  = (An - 4) & {0xFFFF, MASK}

        if <ea> = (d16,An)
                oa  = (An + se_d16) & {0xFFFF0x, MASK}
```

Here, *oa* is the calculated operand address and *se_d16* is a sign-extended 16-bit displacement. For auto-addressing modes of post-increment and pre-decrement, the updated An value calculation is also shown.

Use of the post-increment addressing mode, {(An)+} with the MASK is suggested for circular queue implementations.

BDM: 0x805 (MASK)

Access: User read/write
BDM read/write

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

Figure 4-3. Mask Register (MASK)

Table 4-4. MASK Field Descriptions

Field	Description
31–16	Reserved, must be set.
15–0 MASK	Performs a simple AND with the operand address for MAC instructions.

4.2.3 Accumulator Registers (ACC0–3)

The accumulator registers store 32-bits of the MAC operation result. The accumulator extension registers form the entire 48-bit result.

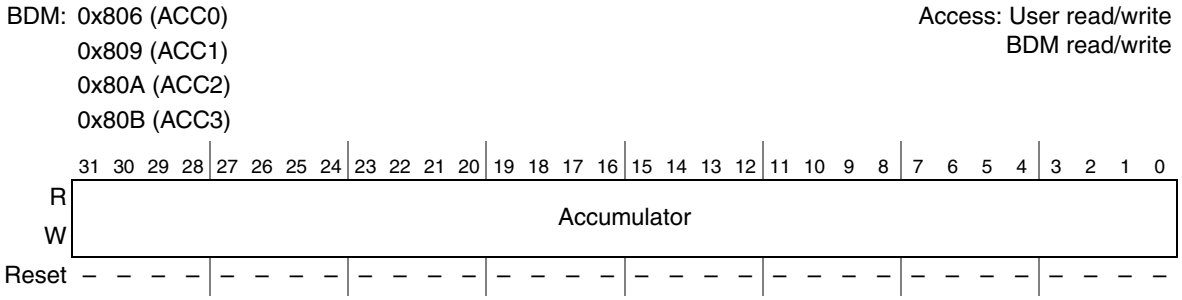


Figure 4-4. Accumulator Registers (ACC0–3)

Table 4-5. ACC0–3 Field Descriptions

Field	Description
31–0 Accumulator	Store 32-bits of the result of the MAC operation.

4.2.4 Accumulator Extension Registers (ACCext01, ACCext23)

Each pair of 8-bit accumulator extension fields are concatenated with the corresponding 32-bit accumulator register to form the 48-bit accumulator. For more information, see Section 4.3, “Functional Description.”

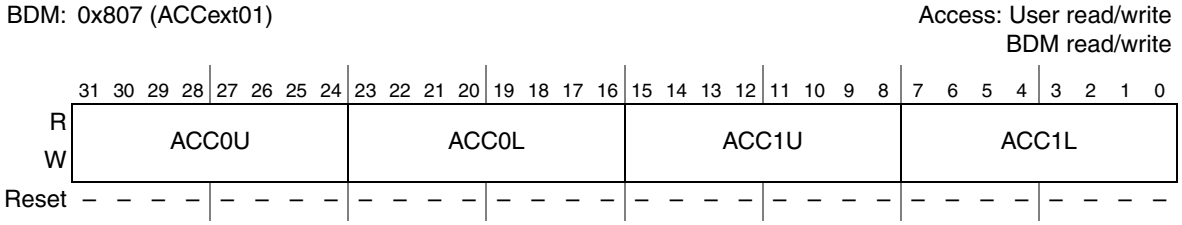


Figure 4-5. Accumulator Extension Register (ACCext01)

Table 4-6. ACCext01 Field Descriptions

Field	Description
31–24 ACC0U	Accumulator 0 upper extension byte
23–16 ACC0L	Accumulator 0 lower extension byte
15–8 ACC1U	Accumulator 1 upper extension byte
7–0 ACC1L	Accumulator 1 lower extension byte

BDM: 0x808 (ACCext23)

Access: User read/write
BDM read/write

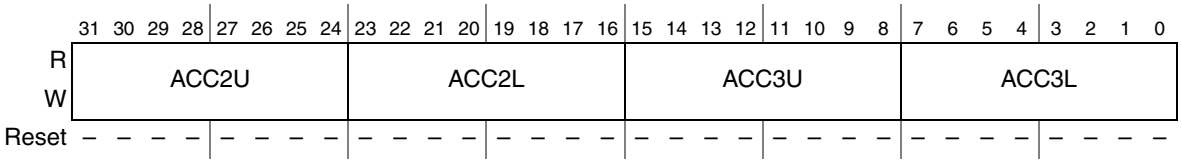


Figure 4-6. Accumulator Extension Register (ACCext23)

Table 4-7. ACCext23 Field Descriptions

Field	Description
31–24 ACC2U	Accumulator 2 upper extension byte
23–16 ACC2L	Accumulator 2 lower extension byte
15–8 ACC3U	Accumulator 3 upper extension byte
7–0 ACC3L	Accumulator 3 lower extension byte

4.3 Functional Description

The MAC speeds execution of ColdFire integer-multiply instructions (MULS and MULU) and provides additional functionality for multiply-accumulate operations. By executing MULS and MULU in the MAC, execution times are minimized and deterministic compared to the 2-bit/cycle algorithm with early termination that the OEP normally uses if no MAC hardware is present.

The added MAC instructions to the ColdFire ISA provide for the multiplication of two numbers, followed by the addition or subtraction of the product to or from the value in an accumulator. Optionally, the product may be shifted left or right by 1 bit before addition or subtraction. Hardware support for saturation arithmetic can be enabled to minimize software overhead when dealing with potential overflow conditions. Multiply-accumulate operations support 16- or 32-bit input operands in these formats:

- Signed integers
- Unsigned integers
- Signed, fixed-point, fractional numbers

The EMAC is optimized for single-cycle, pipelined 32 × 32 multiplications. For word- and longword-sized integer input operands, the low-order 40 bits of the product are formed and used with the destination accumulator. For fractional operands, the entire 64-bit product is calculated and truncated or rounded to the most-significant 40-bit result using the round-to-nearest (even) method before it is combined with the destination accumulator.

For all operations, the resulting 40-bit product is extended to a 48-bit value (using sign-extension for signed integer and fractional operands, zero-fill for unsigned integer operands) before being combined with the 48-bit destination accumulator.

Figure 4-7 and Figure 4-8 show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.

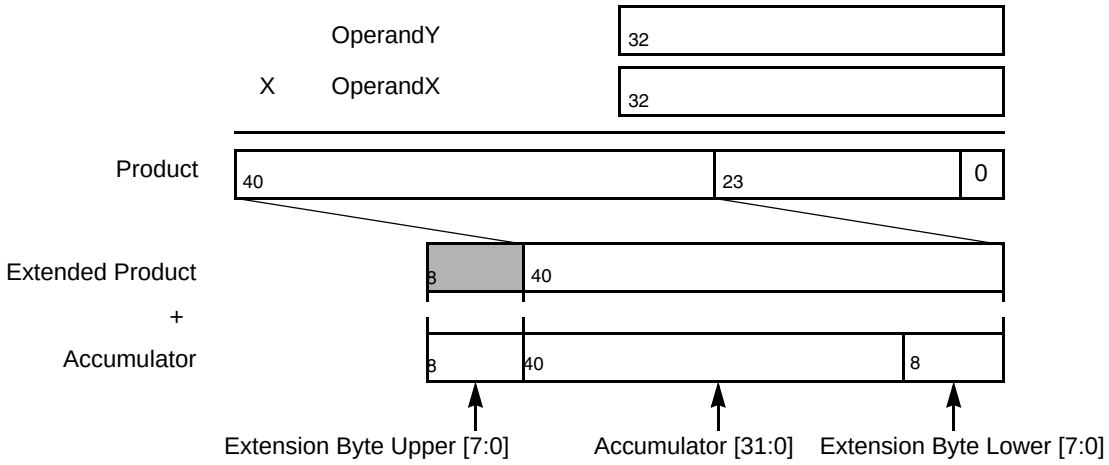


Figure 4-7. Fractional Alignment

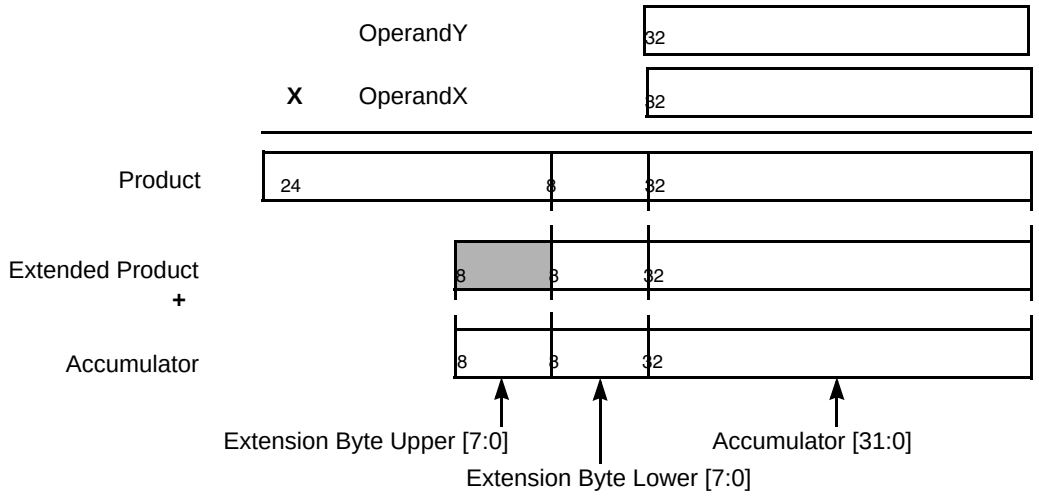


Figure 4-8. Signed and Unsigned Integer Alignment

Therefore, the 48-bit accumulator definition is a function of the EMAC operating mode. Given that each 48-bit accumulator is the concatenation of 16-bit accumulator extension register (ACCext*n*) contents and 32-bit ACC*n* contents, the specific definitions are:

```
if MACSR[6:5] == 00       /* signed integer mode */
      Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
if MACSR[6:5] == 01 or 11 /* signed fractional mode */
      Complete Accumulator [47:0] = {ACCextn[15:8], ACCn[31:0], ACCextn[7:0]}
if MACSR[6:5] == 10       /* unsigned integer mode */
      Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
```

The four accumulators are represented as an array, ACC*n*, where *n* selects the register.

Although the multiplier array is implemented in a four-stage pipeline, all arithmetic MAC instructions have an effective issue rate of 1 cycle, regardless of input operand size or type.

All arithmetic operations use register-based input operands, and summed values are stored in an accumulator. Therefore, an additional MOVE instruction is needed to store data in a general-purpose register. One new feature in EMAC instructions is the ability to choose the upper or lower word of a register as a 16-bit input operand. This is useful in filtering operations if one data register is loaded with the input data and another is loaded with the coefficient. Two 16-bit multiply accumulates can be performed without fetching additional operands between instructions by alternating word choice during calculations.

The EMAC has four accumulator registers versus the MAC's single accumulator. The additional registers improve the performance of some algorithms by minimizing pipeline stalls needed to store an accumulator value back to general-purpose registers. Many algorithms require multiple calculations on a given data set. By applying different accumulators to these calculations, it is often possible to store one accumulator without any stalls while performing operations involving a different destination accumulator.

The need to move large amounts of data presents an obstacle to obtaining high throughput rates in DSP engines. Existing ColdFire instructions can accommodate these requirements. A MOVEM instruction can efficiently move large data blocks by generating line-sized burst references. The ability to load an operand simultaneously from memory into a register and execute a MAC instruction makes some DSP operations such as filtering and convolution more manageable.

The programming model includes a mask register (MASK), which can optionally be used to generate an operand address during MAC + MOVE instructions. The register application with auto-increment addressing mode supports efficient implementation of circular data queues for memory operands.

4.3.1 Fractional Operation Mode

This section describes behavior when the fractional mode is used (MACSR[F/I] is set).

4.3.1.1 Rounding

When the processor is in fractional mode, there are two operations during which rounding can occur:

1. Execution of a store accumulator instruction (`move.l ACCx,Rx`). The lsbs of the 48-bit accumulator logic are used to round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared, the low-order 8 bits round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.
2. Execution of a MAC (or MSAC) instruction with 32-bit operands. If MACSR[R/T] is zero, multiplying two 32-bit numbers creates a 64-bit product truncated to the upper 40 bits; otherwise, it is rounded using round-to-nearest (even) method.

To understand the round-to-nearest-even method, consider the following example involving the rounding of a 32-bit number, R0, to a 16-bit number. Using this method, the 32-bit number is rounded to the closest 16-bit number possible. Let the high-order 16 bits of R0 be named R0.U and the low-order 16 bits be R0.L.

- If R0.L is less than 0x8000, the result is truncated to the value of R0.U.
- If R0.L is greater than 0x8000, the upper word is incremented (rounded up).

- If R0.L is 0x8000, R0 is half-way between two 16-bit numbers. In this case, rounding is based on the lsb of R0.U, so the result is always even (lsb = 0).
  - If the lsb of R0.U equals 1 and R0.L equals 0x8000, the number is rounded up.
  - If the lsb of R0.U equals 0 and R0.L equals 0x8000, the number is rounded down.

This method minimizes rounding bias and creates as statistically correct an answer as possible.

The rounding algorithm is summarized in the following pseudocode:

```
if R0.L < 0x8000
        then Result = R0.U
else if R0.L > 0x8000
        then Result = R0.U + 1
else if lsb of R0.U = 0             /* R0.L = 0x8000 */
        then Result = R0.U
else Result = R0.U + 1
```

The round-to-nearest-even technique is also known as convergent rounding.

4.3.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the EMAC output datapath requires special care during the EMAC's save/restore process. In particular, any result rounding modes must be disabled during the save/restore process so the exact bit-wise contents of the EMAC registers are accessed. Consider the memory structure containing the EMAC programming model:

```
struct   macState {
        int acc0;
        int acc1;
        int acc2;
        int acc3;
        int accext01;
        int accext02;
        int mask;
        int macsr;
} macState;
```

The following assembly language routine shows the proper sequence for a correct EMAC state save. This code assumes all Dn and An registers are available for use, and the memory location of the state save is defined by A7.

```
EMAC_state_save:
        move.l  macsr,d7          ; save the macsr
        clr.l   d0                ; zero the register to ...
        move.l  d0,macsr          ; disable rounding in the macsr
        move.l  acc0,d0           ; save the accumulators
        move.l  acc1,d1
        move.l  acc2,d2
        move.l  acc3,d3
        move.l  accext01,d4       ; save the accumulator extensions
        move.l  accext23,d5
        move.l  mask,d6           ; save the address mask
        movem.l #0x00ff,(a7)      ; move the state to memory
```

This code performs the EMAC state restore:

```
EMAC_state_restore:
```

```
movem.l (a7),#0x00ff      ; restore the state from memory
move.l  #0,macsr          ; disable rounding in the macsr
move.l  d0,acc0           ; restore the accumulators
move.l  d1,acc1
move.l  d2,acc2
move.l  d3,acc3
move.l  d4,accext01       ; restore the accumulator extensions
move.l  d5,accext23
move.l  d6,mask           ; restore the address mask
move.l  d7,macsr          ; restore the macsr
```

Executing this sequence type can correctly save and restore the exact state of the EMAC programming model.

4.3.1.3 MULS/MULU

MULS and MULU are unaffected by fractional-mode operation; operands remain assumed to be integers.

4.3.1.4 Scale Factor in MAC or MSAC Instructions

The scale factor is ignored while the MAC is in fractional mode.

4.3.2 EMAC Instruction Set Summary

Table 4-8 summarizes EMAC unit instructions.

Table 4-8. EMAC Instruction Summary

Command	Mnemonic	Description
Multiply Signed	`muls <ea>y,Dx`	Multiplies two signed operands yielding a signed result
Multiply Unsigned	`mulu <ea>y,Dx`	Multiplies two unsigned operands yielding an unsigned result
Multiply Accumulate	`mac Ry,RxSF,ACCx` `msac Ry,RxSF,ACCx`	Multiplies two operands and adds/subtracts the product to/from an accumulator
Multiply Accumulate with Load	`mac Ry,Rx,<ea>y,Rw,ACCx` `msac Ry,Rx,<ea>y,Rw,ACCx`	Multiplies two operands and combines the product to an accumulator while loading a register with the memory operand
Load Accumulator	`move.l {Ry,#imm},ACCx`	Loads an accumulator with a 32-bit operand
Store Accumulator	`move.l ACCx,Rx`	Writes the contents of an accumulator to a CPU register
Copy Accumulator	`move.l ACCy,ACCx`	Copies a 48-bit accumulator
Load MACSR	`move.l {Ry,#imm},MACSR`	Writes a value to MACSR
Store MACSR	`move.l MACSR,Rx`	Write the contents of MACSR to a CPU register
Store MACSR to CCR	`move.l MACSR,CCR`	Write the contents of MACSR to the CCR
Load MAC Mask Reg	`move.l {Ry,#imm},MASK`	Writes a value to the MASK register
Store MAC Mask Reg	`move.l MASK,Rx`	Writes the contents of the MASK to a CPU register
Load Accumulator Extensions 01	`move.l {Ry,#imm},ACCext01`	Loads the accumulator 0,1 extension bytes with a 32-bit operand

Table 4-8. EMAC Instruction Summary (continued)

Command	Mnemonic	Description
Load Accumulator Extensions 23	`move.l {Ry,#imm},ACCext23`	Loads the accumulator 2,3 extension bytes with a 32-bit operand
Store Accumulator Extensions 01	`move.l ACCext01,Rx`	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store Accumulator Extensions 23	`move.l ACCext23,Rx`	Writes the contents of accumulator 2,3 extension bytes into a CPU register

4.3.3 EMAC Instruction Execution Times

The instruction execution times for the EMAC can be found in Section 3.3.5.6, “EMAC Instruction Execution Times”.

The EMAC execution pipeline overlaps the AGEX stage of the OEP (the first stage of the EMAC pipeline is the last stage of the basic OEP). EMAC units are designed for sustained, fully-pipelined operation on accumulator load, copy, and multiply-accumulate instructions. However, instructions that store contents of the multiply-accumulate programming model can generate OEP stalls that expose the EMAC execution pipeline depth:

```
mac.w    Ry, Rx, Acc0
move.l   Acc0, Rz
```

The MOVE.L instruction that stores the accumulator to an integer register (Rz) stalls until the program-visible copy of the accumulator is available. Figure 4-9 shows EMAC timing.

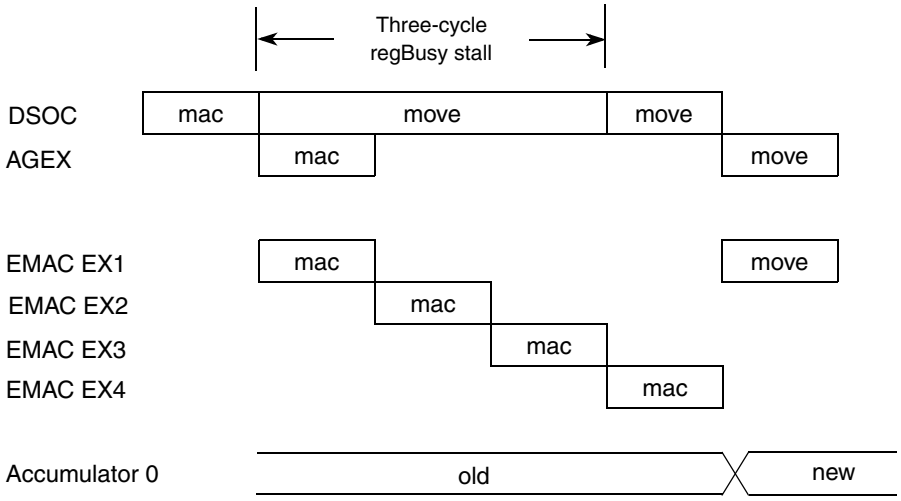


Figure 4-9. EMAC-Specific OEP Sequence Stall

In Figure 4-9, the OEP stalls the store-accumulator instruction for three cycles: the EMAC pipeline depth minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the AGEX stage. As the store-accumulator instruction reaches the AGEX stage where the operation is performed, the recently-updated accumulator 0 value is available.

As with change or use stalls between accumulators and general-purpose registers, introducing intervening instructions that do not reference the busy register can reduce or eliminate sequence-related store-MAC instruction stalls. A major benefit of the EMAC is the addition of three accumulators to minimize stalls caused by exchanges between accumulator(s) and general-purpose registers.

4.3.4 Data Representation

MACSR[S/U,F/I] selects one of the following three modes, where each mode defines a unique operand type:

1. Two's complement signed integer: In this format, an N-bit operand value lies in the range $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$. The binary point is right of the lsb.
2. Unsigned integer: In this format, an N-bit operand value lies in the range $0 \leq \text{operand} \leq 2^{N} - 1$. The binary point is right of the lsb.
3. Two's complement, signed fractional: In an N-bit number, the first bit is the sign bit. The remaining bits signify the first N-1 bits after the binary point. Given an N-bit number, $a_{N-1}a_{N-2}a_{N-3}\ldots a_2a_1a_0$, its value is given by the equation in Equation 4-3.

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{-(i+1-N)} \cdot ai \qquad \textit{Eqn. 4-3}$$

This format can represent numbers in the range $-1 \leq \text{operand} \leq 1 - 2^{-(N-1)}$.

For words and longwords, the largest negative number that can be represented is -1, whose internal representation is 0x8000 and 0x8000_0000, respectively. The largest positive word is 0x7FFF or $(1 - 2^{-15})$; the most positive longword is 0x7FFF_FFFF or $(1 - 2^{-31})$. Thus, the number range for these signed fractional numbers is [-1.0, ..., 1.0].

4.3.5 MAC Opcodes

MAC opcodes are described in the *ColdFire Programmer's Reference Manual*.

Remember the following:

- Unless otherwise noted, the value of MACSR[N,Z] is based on the result of the final operation that involves the product and the accumulator.
- The overflow (V) flag is managed differently. It is set if the complete product cannot be represented as a 40-bit value (this applies to 32 × 32 integer operations only) or if the combination of the product with an accumulator cannot be represented in the given number of bits. The EMAC design includes an additional product/accumulation overflow bit for each accumulator that are treated as sticky indicators and are used to calculate the V bit on each MAC or MSAC instruction. See Section 4.2.1, "MAC Status Register (MACSR)".

- For the MAC design, the assembler syntax of the MAC (multiply and add to accumulator) and MSAC (multiply and subtract from accumulator) instructions does not include a reference to the single accumulator. For the EMAC, assemblers support this syntax and no explicit reference to an accumulator is interpreted as a reference to ACC0. Assemblers also support syntaxes where the destination accumulator is explicitly defined.
- The optional 1-bit shift of the product is specified using the notation {<< | >>} SF, where <<1 indicates a left shift and >>1 indicates a right shift. The shift is performed before the product is added to or subtracted from the accumulator. Without this operator, the product is not shifted. If the EMAC is in fractional mode (MACSR[F/I] is set), SF is ignored and no shift is performed. Because a product can overflow, the following guidelines are implemented:
  - For unsigned word and longword operations, a zero is shifted into the product on right shifts.
  - For signed, word operations, the sign bit is shifted into the product on right shifts unless the product is zero. For signed, longword operations, the sign bit is shifted into the product unless an overflow occurs or the product is zero, in which case a zero is shifted in.
  - For all left shifts, a zero is inserted into the lsb position.

The following pseudocode explains basic MAC or MSAC instruction functionality. This example is presented as a case statement covering the three basic operating modes with signed integers, unsigned integers, and signed fractionals. Throughout this example, a comma-separated list in curly brackets, {}, indicates a concatenation operation.

```
switch (MACSR[6:5])     /* MACSR[S/U, F/I] */
{
   case 0:              /* signed integers */
     if (MACSR.OMC == 0 || MACSR.PAVn == 0)
         then {
               MACSR.PAVn = 0
               /* select the input operands */
               if (sz == word)
                  then {if (U/Ly == 1)
                       then operandY[31:0] = {sign-extended Ry[31], Ry[31:16]}
                        else operandY[31:0] = {sign-extended Ry[15], Ry[15:0]}
                        if (U/Lx == 1)
                       then operandX[31:0] = {sign-extended Rx[31], Rx[31:16]}
                        else operandX[31:0] = {sign-extended Rx[15], Rx[15:0]}
                  }
                  else {operandY[31:0] = Ry[31:0]
                        operandX[31:0] = Rx[31:0]
                  }

               /* perform the multiply */
               product[63:0] = operandY[31:0] * operandX[31:0]

               /* check for product overflow */
       if ((product[63:39] != 0x0000_00_0) && (product[63:39] != 0xffff_ff_1))
                  then {        /* product overflow */
                        MACSR.PAVn = 1
                        MACSR.V = 1
                        if (inst == MSAC && MACSR.OMC == 1)
                           then if (product[63] == 1)
                                   then result[47:0] = 0x0000_7fff_ffff
                                   else result[47:0] = 0xffff_8000_0000
```

```
                        else if (MACSR.OMC == 1)
                                then /* overflowed MAC,
                                        saturationMode enabled */
                                     if (product[63] == 1)
                                       then result[47:0] = 0xffff_8000_0000
                                       else result[47:0] = 0x0000_7fff_ffff
                   }

               /* sign-extend to 48 bits before performing any scaling */
                        product[47:40] = {8{product[39]}}    /* sign-extend */

               /* scale product before combining with accumulator */
               switch (SF)     /* 2-bit scale factor */
               {
                    case 0:     /* no scaling specified */
                       break;
                    case 1:     /* SF = “<< 1” */
                       product[40:0] = {product[39:0], 0}
                       break;
                    case 2:     /* reserved encoding */
                       break;
                    case 3:     /* SF = “>> 1” */
                       product[39:0] = {product[39], product[39:1]}
                       break;
               }

               if (MACSR.PAVn == 0)
                  then {if (inst == MSAC)
                           then result[47:0] = ACCx[47:0] - product[47:0]
                           else result[47:0] = ACCx[47:0] + product[47:0]
                  }

               /* check for accumulation overflow */
               if (accumulationOverflow == 1)
                  then {MACSR.PAVn = 1
                        MACSR.V = 1
                        if (MACSR.OMC == 1)
                           then /* accumulation overflow,
                                   saturationMode enabled */
                                if (result[47] == 1)
                                   then result[47:0] = 0x0000_7fff_ffff
                                   else result[47:0] = 0xffff_8000_0000
                  }
               /* transfer the result to the accumulator */
               ACCx[47:0] = result[47:0]
         }
         MACSR.V = MACSR.PAVn
         MACSR.N = ACCx[47]
         if (ACCx[47:0] == 0x0000_0000_0000)
            then MACSR.Z = 1
            else MACSR.Z = 0
         if ((ACCx[47:31] == 0x0000_0) || (ACCx[47:31] == 0xffff_1))
            then MACSR.EV = 0
            else MACSR.EV = 1
   break;
      case 1,3:              /* signed fractionals */
      if (MACSR.OMC == 0 || MACSR.PAVn == 0)
```

```
      then {
            MACSR.PAVn = 0
            if (sz == word)
               then {if (U/Ly == 1)
                        then operandY[31:0] = {Ry[31:16], 0x0000}
                        else operandY[31:0] = {Ry[15:0],  0x0000}
                     if (U/Lx == 1)
                        then operandX[31:0] = {Rx[31:16], 0x0000}
                        else operandX[31:0] = {Rx[15:0],  0x0000}
               }
               else {operandY[31:0] = Ry[31:0]
                     operandX[31:0] = Rx[31:0]
               }
            /* perform the multiply */
            product[63:0] = (operandY[31:0] * operandX[31:0]) << 1
            /* check for product rounding */
            if (MACSR.R/T == 1)
               then { /* perform convergent rounding */
                     if (product[23:0] > 0x80_0000)
                        then product[63:24] = product[63:24] + 1
               else if ((product[23:0] == 0x80_0000) && (product[24] == 1))
                                then product[63:24] = product[63:24] + 1
               }
            /* sign-extend to 48 bits and combine with accumulator */
            /* check for the -1 * -1 overflow case */
    if ((operandY[31:0] == 0x8000_0000) && (operandX[31:0] == 0x8000_0000))
               then product[71:64] = 0x00                /* zero-fill */
               else product[71:64] = {8{product[63]}}    /* sign-extend */
            if (inst == MSAC)
               then result[47:0] = ACCx[47:0] - product[71:24]
               else result[47:0] = ACCx[47:0] + product[71:24]
            /* check for accumulation overflow */
            if (accumulationOverflow == 1)
               then {MACSR.PAVn = 1
                     MACSR.V = 1
                     if (MACSR.OMC == 1)
                        then /* accumulation overflow,
                                saturationMode enabled */
                             if (result[47] == 1)
                                then result[47:0] = 0x007f_ffff_ff00
                                else result[47:0] = 0xff80_0000_0000
               }
            /* transfer the result to the accumulator */
            ACCx[47:0] = result[47:0]
              }
      MACSR.V = MACSR.PAVn
      MACSR.N = ACCx[47]
      if (ACCx[47:0] == 0x0000_0000_0000)
          then MACSR.Z = 1
          else MACSR.Z = 0
      if ((ACCx[47:39] == 0x00_0) || (ACCx[47:39] == 0xff_1))
          then MACSR.EV = 0
          else MACSR.EV = 1
break;
case 2:              /* unsigned integers */
   if (MACSR.OMC == 0 || MACSR.PAVn == 0)
      then {
```

```
MACSR.PAVn = 0
/* select the input operands */
if (sz == word)
   then {if (U/Ly == 1)
            then operandY[31:0] = {0x0000, Ry[31:16]}
            else operandY[31:0] = {0x0000, Ry[15:0]}
         if (U/Lx == 1)
            then operandX[31:0] = {0x0000, Rx[31:16]}
            else operandX[31:0] = {0x0000, Rx[15:0]}
   }
   else {operandY[31:0] = Ry[31:0]
         operandX[31:0] = Rx[31:0]
   }

/* perform the multiply */
product[63:0] = operandY[31:0] * operandX[31:0]

/* check for product overflow */
if (product[63:40] != 0x0000_00)
   then {        /* product overflow */
         MACSR.PAVn = 1
         MACSR.V = 1
         if (inst == MSAC && MACSR.OMC == 1)
            then result[47:0] = 0x0000_0000_0000
            else if (MACSR.OMC == 1)
                    then /* overflowed MAC,
                            saturationMode enabled */
                         result[47:0] = 0xffff_ffff_ffff
   }

/* zero-fill to 48 bits before performing any scaling */
         product[47:40] = 0    /* zero-fill upper byte */

/* scale product before combining with accumulator */
switch (SF)     /* 2-bit scale factor */
{
    case 0:     /* no scaling specified */
       break;
    case 1:     /* SF = “<< 1” */
       product[40:0] = {product[39:0], 0}
       break;
    case 2:     /* reserved encoding */
       break;
    case 3:     /* SF = “>> 1” */
       product[39:0] = {0, product[39:1]}
       break;
}

/* combine with accumulator */
if (MACSR.PAVn == 0)
   then {if (inst == MSAC)
            then result[47:0] = ACCx[47:0] - product[47:0]
            else result[47:0] = ACCx[47:0] + product[47:0]
   }

/* check for accumulation overflow */
if (accumulationOverflow == 1)
```

```
                then {MACSR.PAVn = 1
                      MACSR.V = 1
                      if (inst == MSAC && MACSR.OMC == 1)
                         then result[47:0] = 0x0000_0000_0000
                         else if (MACSR.OMC == 1)
                              then /* overflowed MAC,
                                      saturationMode enabled */
                                   result[47:0] = 0xffff_ffff_ffff
                }

              /* transfer the result to the accumulator */
              ACCx[47:0] = result[47:0]
        }
        MACSR.V = MACSR.PAVn
        MACSR.N = ACCx[47]
        if (ACCx[47:0] == 0x0000_0000_0000)
           then MACSR.Z = 1
           else MACSR.Z = 0
        if (ACCx[47:32] == 0x0000)
           then MACSR.EV = 0
           else MACSR.EV = 1
   break;
}
```

Chapter 5
Cryptographic Acceleration Unit (CAU)

5.1 Introduction

The cryptographic acceleration unit (CAU) is a ColdFire coprocessor implementing a set of specialized operations in hardware to increase the throughput of software-based encryption and hashing functions.

5.1.1 Block Diagram

Figure 5-1 shows a simplified block diagram of the CAU.

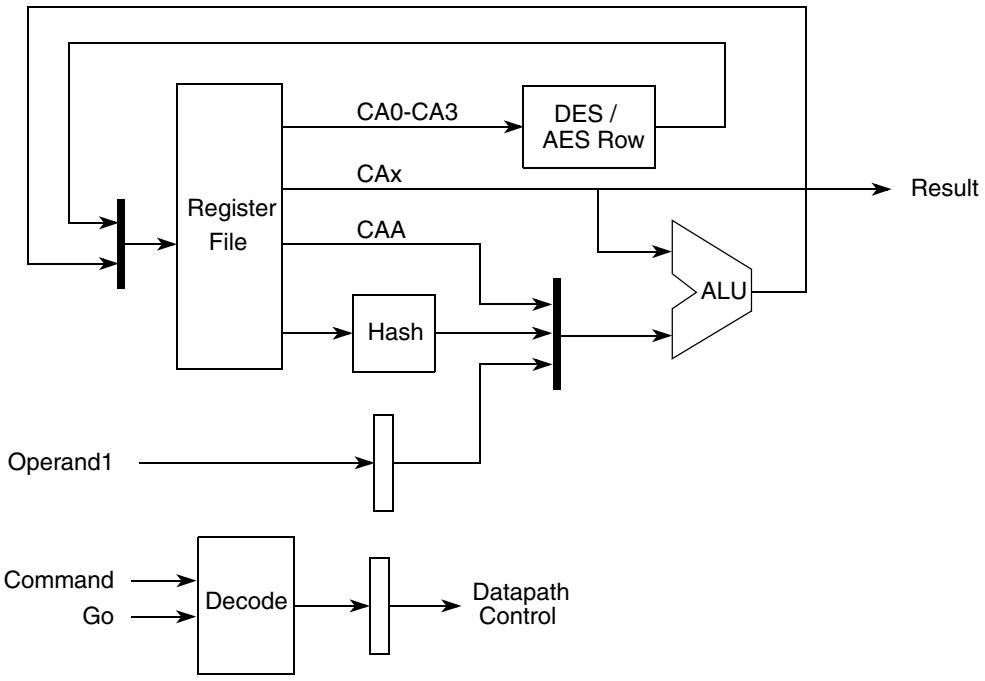


Figure 5-1. Top Level CAU Block Diagram

5.1.2 Overview

The CAU supports acceleration of the following algorithms:

- DES
- 3DES
- AES
- MD5

- SHA-1

This selection of algorithms provides excellent support for network security standards (SSL, IPsec). Additionally, using the CAU efficiently permits the implementation of any higher level functions or modes of operation (HMAC, CBC, etc.) based on the supported algorithm.

The CAU is an instruction-level ColdFire coprocessor. The cryptographic algorithms are implemented partially in software with only functions critical to increasing performance implemented in hardware. The ColdFire coprocessor allows for efficient, fine-grained partitioning of functions between hardware and software.

- Implement the innermost round functions by using the coprocessor instructions
- Implement higher-level functions in software by using the standard ColdFire instructions

This partitioning of functions is key to minimizing size of the CAU while maintaining a high level of throughput. Using software for some functions also simplifies the CAU design. The CAU implements a set of 22 coprocessor commands that operate on a register file of eight 32-bit registers. It is tightly coupled to the ColdFire core and there is no local memory or external interface.

5.1.3 Features

The CAU includes these distinctive features:

- Supports DES, 3DES, AES, MD5, SHA-1 algorithms
- Simple, flexible programming model

5.2 Memory Map/Register Definition

The CAU only supports longword operations and register accesses. All registers support read, write, and ALU operations. However, only bits 1–0 of the CASR are writeable. Bits 31–2 of the CASR must be written as 0 for compatibility with future versions of the CAU.

Table 5-1. CAU Memory Map

Code	Register	DES	AES	SHA-1	MD5	Access	Reset Value	Section/Page
0	CAU status register (CASR)	—	—	—	—	R/W	0x1000_0000	5.2.1/5-4
1	CAU accumulator (CAA)	—	—	T	a	R	0x0000_0000	5.2.2/5-4
2	General purpose register 0 (CA0)	C	W0	A	—	R	0x0000_0000	5.2.3/5-5
3	General purpose register 1 (CA1)	D	W1	B	b	R	0x0000_0000	5.2.3/5-5
4	General purpose register 2 (CA2)	L	W2	C	c	R	0x0000_0000	5.2.3/5-5
5	General purpose register 3 (CA3)	R	W3	D	d	R	0x0000_0000	5.2.3/5-5
6	General purpose register 4 (CA4)	—	—	E	—	R	0x0000_0000	5.2.3/5-5
7	General purpose register 5 (CA5)	—	—	W	—	R	0x0000_0000	5.2.3/5-5

5.2.1 CAU Status Register (CASR)

CASR contains the status and configuration for the CAU.

Register code: 0x0 (CASR) — Access: Read/write via CAU commands

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

Figure 5-2. CAU Status Register (CASR)

Table 5-2. CASR Field Descriptions

Field	Description
31–28 VER	CAU version. Indicates CAU version 0x1 Initial CAU version (This is the value on this device) 0x2 Second version, added support for SHA-256 algorithm
27–2	Reserved, must be cleared.
1 DPE	DES parity error. 0 No error detected 1 DES key parity error detected
0 IC	Illegal command. Indicates an illegal instruction not found in Section 5.3.3, “CAU Commands,” has been executed. 0 No illegal commands issued 1 Illegal coprocessor command issued

5.2.2 CAU Accumulator (CAA)

CAU commands use the CAU accumulator for storage of results and as an operand for the cryptographic algorithms.

Register code: 0x1 (CAA) — Access: Read/write via CAU commands

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

Figure 5-3. CAU Accumulator Register (CAA)

Table 5-3. CAA Field Descriptions

Field	Description
31–0 ACC	Accumulator. Stores results of various CAU commands.

5.2.3 CAU General Purpose Registers (CA*n*)

The nine CAU general purpose registers are used in the CAU commands for storage of results and as operands for the various cryptographic algorithms.

Register code: 0x2 (CA0), 0x3 (CA1), 0x4 (CA2), 0x5 (CA3), 0x6 (CA4), 0x7 (CA5)

Access: Read/write via CAU commands

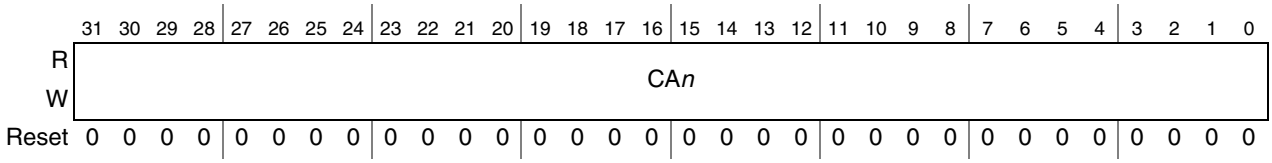


Figure 5-4. CAU General Purpose Registers (CA*n*)

Table 5-4. CA*n* Field Descriptions

Field	Description
31–0 CA*n*	General purpose registers. Used by the CAU commands. Some cryptographic operations work with specific registers.

5.3 Functional Description

5.3.1 Programming Model

The CAU is an instruction-level coprocessor. It has a dedicated register file, a specialized ALU, and specialized units for performing cryptographic operations. The CAU design uses a simple, flexible accumulator-based architecture. Most commands, including load and store, can specify any register in the register file. Some cryptographic operations work with specific registers.

5.3.2 Coprocessor Instructions

Operation of the CAU is controlled via standard ColdFire coprocessor load (cp0ld) and store (cp0st) instructions. The CAU has a dedicated register file accessed using these instructions. The load instruction loads CAU registers and specifies CAU operations. The store instruction stores CAU registers. The example assembler syntax for the CAU is:

```
cp0ld.l         <ea>,<CMD>      ; coprocessor load
cp0st.l         <ea>,<CMD>      ; coprocessor store
```

The <ea> field specifies the source operand (operand1) for load instructions and destination (result) for store instructions. The basic ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)} are supported for this field. The <CMD> field is a 9-bit value that specifies the CAU command for an instruction. Table 5-5 shows how the CAU supports a single command (STR) for store instructions and 21 commands for the load instructions. The CAU only supports longword operations. A CAU command can be issued every clock cycle.

5.3.3 CAU Commands

The CAU supports the commands shown in Table 5-5. All other encodings are reserved. The CASR[IC] bit is set if an undefined command is issued. A specific illegal command (ILL) is defined to allow software self-checking. Reserved commands should not be issued to ensure compatibility with future implementations.

The CMD field specifies the CAU command for the instruction.

Table 5-5. CAU Commands

Inst Type	Command Name	Description	CMD									Operation
			8	7	6	5	4	3	2	1	0	
cp0ld	CNOP	No Operation	0x000									—
cp0ld	LDR	Load Reg	0x01					CAx				Op1 → CAx
cp0st	STR	Store Reg	0x02					CAx				CAx → Result
cp0ld	ADR	Add	0x03					CAx				CAx + Op1 → CAx
cp0ld	RADR	Reverse and Add	0x04					CAx				CAx + ByteRev(Op1) → CAx
cp0ld	ADRA	Add Reg to Acc	0x05					CAx				CAx + CAA → CAA
cp0ld	XOR	Exclusive Or	0x06					CAx				CAx ^ Op1 → CAx
cp0ld	ROTL	Rotate Left	0x07					CAx				CAx <<< Op1 → CAx
cp0ld	MVRA	Move Reg to Acc	0x08					CAx				CAx → CAA
cp0ld	MVAR	Move Acc to Reg	0x09					CAx				CAA → CAx
cp0ld	AESS	AES Sub Bytes	0x0A					CAx				SubBytes(CAx) → CAx
cp0ld	AESIS	AES Inv Sub Bytes	0x0B					CAx				InvSubBytes(CAx) → CAx
cp0ld	AESC	AES Column Op	0x0C					CAx				MixColumns(CAx)^Op1→ CAx
cp0ld	AESIC	AES Inv Column Op	0x0D					CAx				InvMixColumns(CAx^Op1) → CAx
cp0ld	AESR	AES Shift Rows	0x0E0									ShiftRows(CA0-CA3) → CA0-CA3
cp0ld	AESIR	AES Inv Shift Rows	0x0F0									InvShiftRows(CA0-CA3)→ CA0-CA3
cp0ld	DESR	DES Round	0x10					IP	FP	KS[1:0]		DES Round(CA0-CA3)→CA0-CA3
cp0ld	DESK	DES Key Setup	0x11					0	0	CP	DC	DES Key Op(CA0-CA1)→CA0-CA1 Key Parity Error & CP → CASR[1]
cp0ld	HASH	Hash Function	0x12					0	HF[2:0]			Hash Func(CA1-CA3)+CAA→CAA
cp0ld	SHS	Secure Hash Shift	0x130									CAA <<< 5→ CAA, CAA→CA0, CA0→CA1, CA1 <<< 30 → CA2, CA2→CA3, CA3→CA4
cp0ld	MDS	Message Digest Shift	0x140									CA3→CAA, CAA→CA1, CA1→CA2, CA2→CA3,
cp0ld	ILL	Illegal Command	0x1F0									0x1→CASR[0]

Section 5.4.2, "Assembler Equate Values," contains a set of assembly constants used in the command descriptions here. If supported by the assembler, macros can also be created for each instruction. The value CA*x* should be interpreted as any CAU register (CASR, CAA, CA*n*) and the <ea> field as one of the supported ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)}. For example, the instruction to add the value from the core register D1 to the CAU register CA0 is:

```
cp0ld.l         %d1,#ADR+CA0        ; CA0=CA0+d1
```

5.3.3.1 Coprocessor No Operation (CNOP)

```
cp0ld.l   #CNOP
```

The CNOP command is the coprocessor no-op defined by the ColdFire coprocessor definition for synchronization. It is not actually issued to the coprocessor from the core.

5.3.3.2 Load Register (LDR)

```
cp0ld.l   <ea>,#LDR+CAx
```

The LDR command loads CAx with the source data specified by <ea>.

5.3.3.3 Store Register (STR)

```
cp0st.l   <ea>,#STR+CAx
```

The STR command stores the value from CAx to the destination specified by <ea>.

5.3.3.4 Add to Register (ADR)

```
cp0ld.l   <ea>,#ADR+CAx
```

The ADR command adds the source operand specified by <ea> to CAx and stores the result in CAx.

5.3.3.5 Reverse and Add to Register (RADR)

```
cp0ld.l   <ea>,#RADR+CAx
```

The RADR command performs a byte reverse on the source operand specified by <ea>, adds that value to CAx, and stores the result in CAx. Table 5-6 shows an example.

Table 5-6. RADR Command Example

Operand	CAx Before	CAx After
0x0102_0304	0xA0B0_C0D0	0xA4B3_C2D1

5.3.3.6 Add Register to Accumulator (ADRA)

```
cp0ld.l   #ADRA+CAx
```

The ADRA command adds CAx to CAA and stores the result in CAA.

5.3.3.7 Exclusive Or (XOR)

```
cp0ld.l   <ea>,#XOR+CAx
```

The XOR command performs an exclusive-or of the source operand specified by <ea> with CAx and stores the result in CAx.

5.3.3.8 Rotate Left (ROTL)

```
cp0ld.l   <ea>,#ROTL+CAx
```

ROTL rotates the CAx bits to the left with the result stored back to CAx. The number of bits to rotate is the value specified by <ea> modulo 32.

5.3.3.9 Move Register to Accumulator (MVRA)

```
cp0ld.l   #MVRA+CAx
```

The MVRA command moves the value from the source register CAx to the destination register CAA.

5.3.3.10 Move Accumulator to Register (MVAR)

```
cp0ld.l   #MVAR+CAx
```

The MVAR command moves the value from source register CAA to the destination register CAx.

5.3.3.11 AES Substitution (AESS)

```
cp0ld.l   #AESS+CAx
```

The AESS command performs the AES byte substitution operation on CAx and stores the result back to CAx.

5.3.3.12 AES Inverse Substitution (AESIS)

```
cp0ld.l   #AESIS+CAx
```

The AESIS command performs the AES inverse byte substitution operation on CAx and stores the result back to CAx.

5.3.3.13 AES Column Operation (AESC)

```
cp0ld.l   <ea>,#AESC+CAx
```

The AESC command performs the AES column operation on the contents of CAx then performs an exclusive-or of that result with the source operand specified by <ea> and stores the result in CAx.

5.3.3.14 AES Inverse Column Operation (AESIC)

```
cp0ld.l   <ea>,#AESIC+CAx
```

The AESIC command performs an exclusive-or operation of the source operand specified by <ea> on the contents of CAx followed by the AES inverse mix column operation on that result and stores the result back in CAx.

5.3.3.15 AES Shift Rows (AESR)

```
cp0ld.l   #AESR
```

The AESR command performs the AES shift rows operation on registers CA0, CA1, CA2, and CA3. Table 5-7 shows an example.

Table 5-7. AESR Command Example

Register	Before	After
CA0	0x0102_0304	0x0106_0B00
CA1	0x0506_0708	0x050A_0F04
CA2	0x090A_0B0C	0x090E_0308
CA3	0x0D0E_0F00	0x0D02_070C

5.3.3.16 AES Inverse Shift Rows (AESIR)

```
cp0ld.l   #AESIR
```

The AESIR command performs the AES inverse shift rows operation on registers CA0, CA1, CA2 and CA3. Table 5-8 has an example.

Table 5-8. AESIR Command Example

Register	Before	After
CA0	01060B00	01020304
CA1	050A0F04	05060708
CA2	090E0308	090A0B0C
CA3	0D02070C	0D0E0F00

5.3.3.17 DES Round (DESR)

```
cp0ld.l   #DESR+{IP}+{FP}+{KSx}
```

The DESR command performs a round of the DES algorithm and a key schedule update with the following source and destination designations: CA0=C, CA1=D, CA2=L, CA3=R. If the IP bit is set, DES initial permutation performs on CA2 and CA3 before the round operation. If the FP bit is set, DES final permutation (inverse initial permutation) performs on CA2 and CA3 after the round operation. The round operation uses the source values from registers CA0 and CA1 for the key addition operation. The KSx field specifies the shift for the key schedule operation to update the values in CA0 and CA1. Table 5-9 defines the specific shift function performed based on the KSx field.

Table 5-9. Key Shift Function Codes

KSx Code	KSx Define	Shift Function
0	KSL1	Left 1
1	KSL2	Left 2

Table 5-9. Key Shift Function Codes (continued)

KSx Code	KSx Define	Shift Function
2	KSR1	Right 1
3	KSR2	Right 2

5.3.3.18 DES Key Setup (DESK)

```
cp0ld.l   #DESK+{CP}+{DC}
```

The DESK command performs the initial key transformation (permuted choice 1) defined by the DES algorithm on CA0 and CA1 with CA0 containing bits 1–32 of the key and CA1 containing bits 33–64 of the key[1]. If the DC bit is set, no shift operation performs and the values C_0 and D_0 store back to CA0 and CA1 respectively. The DC bit should be set for decrypt operations. If the DC bit is not set, a left shift by one also occurs and the values C_1 and D_1 store back to CA0 and CA1 respectively. The DC bit should be cleared for encrypt operations. If the CP bit is set and a key parity error is detected, CASR[DPE] bit is set; otherwise, it is cleared.

5.3.3.19 Hash Function (HASH)

```
cp0ld.l   #HASH+HFx
```

The HASH command performs a hashing operation on a set of registers and adds that result to the value in CAA and stores the result in CAA. The specific hash function performed is based on the HFx field as defined in Table 5-10.

Table 5-10. Hash Function Codes

HFx Code	HFx Define	Hash Function	Hash Logic
0	HFF	MD5 F()	(CA1 & CA2) | ($\overline{CA1}$ & CA3)
1	HFG	MD5 G()	(CA1 & CA3) | (CA2 & $\overline{CA3}$)
2	HFH	MD5 H(), SHA Parity()	CA1 ^ CA2 ^ CA3
3	HFI	MD5 I()	CA2 ^ (CA1 | $\overline{CA3}$)
4	HFC	SHA Ch()	(CA1 & CA2) ^ ($\overline{CA1}$ & CA3)
5	HFM	SHA Maj()	(CA1 & CA2) ^ (CA1 & CA3) ^ (CA2 & CA3)

5.3.3.20 Secure Hash Shift (SHS)

```
cp0ld.l   #SHS
```

The SHS command does a set of parallel register-to-register move and shift operations for implementing SHA-1. The following source and destination assignments are made: CAA=CAA<<<5, CA0=CAA, CA1=CA0, CA2=CA1<<<30, CA3=CA2, CA4=CA3.

1.The DES algorithm numbers the most significant bit of a block as bit 1 and the least significant as bit 64.

5.3.3.21 Message Digest Shift (MDS)

```
cp0ld.l   #MDS
```

The MDS command does a set of parallel register-to-register move operations for implementing MD5. The following source and destination assignments are made: CAA=CA3, CA1=CAA, CA2=CA1, CA3=CA2.

5.3.3.22 Illegal Command (ILL)

```
cp0ld.l   #ILL
```

The ILL command is a specific illegal command that sets CASR[IC]. All other illegal commands are reserved for use in future implementations.

5.4 Application/Initialization Information

5.4.1 Code Example

A code fragment is shown below as an example of how the CAU is used. This example shows the round function of the AES algorithm. Core register A0 is pointing to the key schedule.

```
cp0ld.l   #AESS+CA0         ; sub bytes w0
cp0ld.l   #AESS+CA1         ; sub bytes w1
cp0ld.l   #AESS+CA2         ; sub bytes w2
cp0ld.l   #AESS+CA3         ; sub bytes w3
cp0ld.l   #AESR             ; shift rows
cp0ld.l   (%a0)+,#AESC+CA0  ; mix col, add key w0
cp0ld.l   (%a0)+,#AESC+CA1  ; mix col, add key w1
cp0ld.l   (%a0)+,#AESC+CA2  ; mix col, add key w2
cp0ld.l   (%a0)+,#AESC+CA3  ; mix col, add key w3
```

5.4.2 Assembler Equate Values

The following equates ease programming of the CAU.

```
; CAU Registers (CAx)
        .set    CASR,0x0
        .set    CAA,0x1
        .set    CA0,0x2
        .set    CA1,0x3
        .set    CA2,0x4
        .set    CA3,0x5
        .set    CA4,0x6
        .set    CA5,0x7

; CAU Commands
        .set    CNOP,0x000
        .set    LDR,0x010
        .set    STR,0x020
        .set    ADR,0x030
        .set    RADR,0x040
        .set    ADRA,0x050
        .set    XOR,0x060
        .set    ROTL,0x070
        .set    MVRA,0x080
```

```
        .set    MVAR,0x090
        .set    AESS,0x0A0
        .set    AESIS,0x0B0
        .set    AESC,0x0C0
        .set    AESIC,0x0D0
        .set    AESR,0x0E0
        .set    AESIR,0x0F0
        .set    DESR,0x100
        .set    DESK,0x110
        .set    HASH,0x120
        .set    SHS,0x130
        .set    MDS,0x140
        .set    ILL,0x1F0

; DESR  Fields
        .set    IP,0x08          ; initial permutation
        .set    FP,0x04          ; final permutation
        .set    KSL1,0x00        ; key schedule left 1 bit
        .set    KSL2,0x01        ; key schedule left 2 bits
        .set    KSR1,0x02        ; key schedule right 1 bit
        .set    KSR2,0x03        ; key schedule right 2 bits

; DESK Field
        .set    DC,0x01          ; decrypt key schedule
        .set    CP,0x02          ; check parity

; HASH Functions Codes
        .set    HFF,0x0          ; MD5 F() CA1&CA2 | ~CA1&CA3
        .set    HFG,0x1          ; MD5 G() CA1&CA3 | CA2&~CA3
        .set    HFH,0x2          ; MD5 H(), SHA Parity() CA1^CA2^CA3
        .set    HFI,0x3          ; MD5 I()  CA2^(CA1|~CA3)
        .set    HFC,0x4          ; SHA Ch() CA1&CA2 ^ ~CA1&CA3
        .set    HFM,0x5          ; SHA Maj() CA1&CA2 ^ CA1&CA3 ^ CA2&CA3
```

Chapter 6
Random Number Generator (RNG)

6.1 Introduction

This chapter describes the random number generator (RNG), including a programming model, functional description, and application information.

6.1.1 Overview

The random number generator (RNG) module is capable of generating 32-bit random numbers. It complies with Federal Information Processing Standard (FIPS) 140 standards for randomness and non-determinism. The random bits generate by clocking shift registers with clocks derived from ring oscillators. The configuration of the shift registers ensures statistically good data (data that looks random). The oscillators with their unknown frequencies provide the required entropy needed to create random data.

CAUTION

There is no known cryptographic proof showing that this is a secure method of generating random data. In fact, there may be an attack against the random number generator if its output is used directly in a cryptographic application (the attack is based on the linearity of the internal shift registers). In light of this, it is highly recommended to use the random data produced by this module as an input seed to a NIST-approved (based on DES or SHA-1) or cryptographically-secure (RSA generator or BBS generator) random number generation algorithm.

It is also recommended to use other sources of entropy along with the RNG to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed the better. The following is a list of sources that can be easily combined with the output of this module.

– Current time using highest precision possible
– Mouse and keyboard motions (or equivalent if being used on a cell phone or PDA)
– Other entropy supplied directly by the user

NOTE

See Appendix D of the NIST Special Publication 800-90 “Recommendation for Random Number Generation Using Deterministic Random Bit Generators” for more information: http://csrc.nist.gov

6.2 Memory Map/Register Definition

Table 6-1 shows the address map for the RNG module. Detailed register descriptions are found in the following section.

Table 6-1. RNG Block Memory Map

IPSBAR Offset	Register	Width (bits)	Access	Reset Value	Section/Page
0x1F_0000	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	6.2.1/6-2
0x1F_0004	RNG Status Register (RNGSR)	32	R	0x0010_0000	6.2.2/6-3
0x1F_0008	RNG Entropy Register (RNGER)	32	W	0x0000_0000	6.2.3/6-4
0x1F_000C	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	6.2.4/6-4

6.2.1 RNG Control Register (RNGCR)

Immediately following reset, the RNG begins generating entropy (random data) in its internal shift registers. Random data is not pushed to the output FIFO until after the RNGCR[GO] bit is set. After this, a random 32-bit word is pushed to RNGOUT every 256 cycles.

IPSBAR Offset: 0x1F_0000 (RNGCR) — Access: User read/write

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

Figure 6-1. RNG Control Register (RNGCR)

Table 6-2. RNGCR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 SLM	Sleep mode. The RNGA can be placed in low power mode by setting this bit. When this bit is set, the oscillators are disabled.Clearing this bit causes the RNGA to exit sleep mode. RNGOUT is not pushed while the RNGA is in sleep mode. 0 RNGA is not in sleep mode. 1 RNGA is in sleep mode.
3 CI	Clear interrupt. Writing a 1 to this bit clears the error interrupt and RNGSR[EI]. This bit is self-clearing, 0 Do not clear error interrupt. 1 Clear error interrupt.
2 IM	Interrupt mask. 0 Error interrupt enabled. 1 Error interrupt masked.

Table 6-2. RNGCR Field Descriptions (continued)

Field	Description
1 HA	High assurance. Notifies core when RNGOUT underflow has occurred (RNGOUT is read while empty). Enables the security violation bit in the RNGSR. Bit is sticky and only cleared by hardware reset. 0 Disable security violation notification. 1 Enable security violation notification.
0 GO	Go bit. Starts/stops random data from being generated. Bit is sticky and only cleared by hardware reset. 0 RNGOUT not loaded with random data. 1 RNGOUT loaded with random data.

6.2.2 RNG Status Register (RNGSR)

The RNGSR, shown in Figure 6-2, is a read only register which reflects the internal status of the RNG.

IPSBAR Offset: 0x1F_0004 (RNGSR) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	OFS								OFL								0	0	0	SLP	EI	FUF	LRS	SV
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-2. RNG Status Register (RNGSR)

Table 6-3. RNGSR Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–16 OFS	Output FIFO size. Indicates size of the output FIFO (1 word) and maximum possible value of RNGR[OFL].
15–8 OFL	Output FIFO level. Indicates current number of random words in the output FIFO. Determines if valid random data is available for reading from the FIFO without causing an underflow condition. On this device, the maximum value for this field is 0x01.
7–5	Reserved, must be cleared.
4 SLP	Sleep. This bit reflects whether the RNG is in sleep mode. When this bit is set, the RNGA is in sleep mode and the oscillator clocks are inactive. While in this mode, RNGOUT is not loaded. 0 RNGA is not in sleep mode. 1 RNGA is in sleep mode.
3 EI	Error interrupt. Signals a FIFO underflow. Reset by a write to RNGCR[CI] and not masked by RNGCR[IM]. 0 RNGOUT not read while empty. 1 RNGOUT read while empty.
2 FUF	FIFO underflow. Signals FIFO underflow. Reset by reading RNGSR. 0 RNGOUT not read while empty since last read of RNGSR. 1 RNGOUT read while empty since last read of RNGSR.

Table 6-3. RNGSR Field Descriptions (continued)

Field	Description
1 LRS	Last read status. Reflects status of most recent read of RNGOUT. 0 During last read, RNGOUT was not empty. 1 During last read, RNGOUT was empty (underflow condition).
0 SV	Security violation. When enabled by RNGCR[HA], signals that a RNGOUT underflow has occurred. Bit is sticky and is only cleared by hardware reset. 0 No violation occurred or RNGCR[HA] is cleared. 1 Security violation (RNGOUT underflow) has occurred.

6.2.3 RNG Entropy Register (RNGER)

The RNGER is a write-only register which allows the user to insert entropy into the RNG. This register allows an external user to continually seed the RNG with externally generated random data. Although use of this register is recommended, it is optional. The RNGER can be written at any time during operation.

Each time the RNGER is written, the value updates the internal state of the RNG. The update is performed in such a way that the entropy in the RNG's internal state is preserved. Use of the RNGER can increase the entropy but never decrease it.

IPSBAR Offset: 0x1F_0008 (RNGER) Access: User write-only

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

Figure 6-3. RNG Entropy Register (RNGER)

6.2.4 RNG Output FIFO (RNGOUT)

The RNGOUT provides temporary storage for random data generated by the RNG. As long as RNGOUT is not empty, a read of this address returns 32 bits of random data. If RNGOUT is read when it is empty, RNGSR[EI, FUF, LRS] are set. If the interrupt is enabled in RNGCR, an interrupt is triggered to the interrupt controller. The RNGSR[OFL], described in Section 6.2.2, "RNG Status Register (RNGSR)," can be polled to monitor if data is currently resident in RNGOUT. A new random word pushes into the FIFO every 256 clock cycles (as long as RNGOUT is not full). It is very important to poll RNGSR[OFL] to make sure random values are present before reading from RNGOUT.

IPSBAR Offset: 0x1F_000C (RNGOUT) Access: User read-only

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

Figure 6-4. RNGOUT

6.3 Functional Description

Figure 6-5 shows the RNG has three functional blocks: output FIFO, internal bus interface, and the RNG core/control logic blocks. The following sections describe these blocks in more detail.

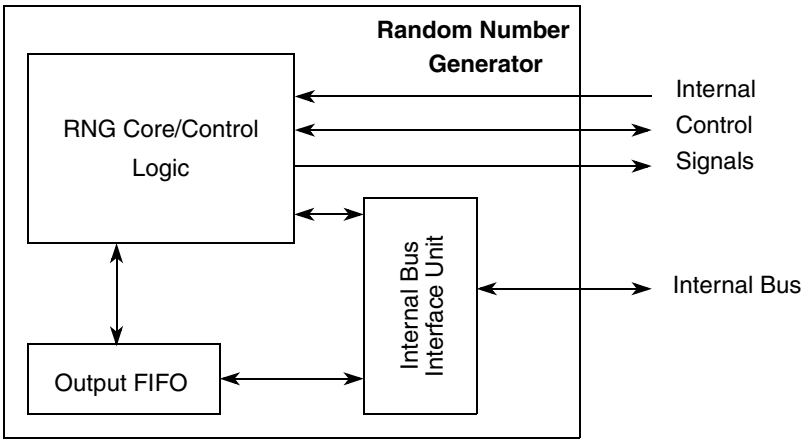


Figure 6-5. RNG Block Diagram

6.3.1 Output FIFO

RNGOUT stores the 32 bits of random data generated by the RNG core/control block. The RNGSR allows the user to check if random data has been written to RNGOUT, through the output FIFO level field. If the user reads from the FIFO when it is empty and the interrupt is enabled, the RNG drives an interrupt request to the interrupt controller. It is very important to poll RNGSR[OFL] to make sure random values are present before reading from RNGOUT.

6.3.2 RNG Core/Control Logic Block

This block contains the RNG's control logic as well as its core engine that generates random data.

6.3.2.1 RNG Control Block

The control block contains the address decoder, all addressable registers, and control state machines for the RNG. This block is responsible for communication with the peripheral interface and the FIFO interface. The block also controls the core engine to generate random data. The general functionality of the block is as follows. After reset, entropy generates and stores in the RNG's shift registers. After RNGCR[GO] is set, RNGOUT is loaded with a random word every 256 cycles.

6.3.2.2 RNG Core Engine

The core engine block contains the logic that generates random data. The logic within the core engine contains the internal shift registers, as well as the logic that generates the two oscillator based clocks. This logic is brainless and must be controlled by the control block. The control block controls how the shift registers are configured and when the oscillator clocks are turned on.

6.4 Initialization/Application Information

The intended general operation of the RNG is as follows:

1. Reset/initialize.
2. Write to the RNG entropy register (optional).
3. Write to the RNG control register and set the interrupt mask, high assurance, and GO bits.
4. Poll RNGSR[OFL] to check for random data in RNGOUT.
5. Read available random data from RNGOUT.
6. Repeat steps 3 and 4 as needed.

Chapter 7
Clock Module

7.1 Introduction

The clock module allows the MCF52235 to be configured for one of several clocking methods. Clocking modes include internal phase-locked loop (PLL) clocking with an external clock reference or an external crystal reference supported by an internal crystal amplifier. The PLL can also be disabled and an external oscillator can be used to clock the device directly. The device always comes out of reset running in external crystal mode (although this mode also supports an external clock source) with the PLL disabled. After out of reset, it is not possible to change the input clock source, although it is possible to enable the PLL and switch between the PLL clock and the oscillator clock as the source of the system clock.The clock module contains the following:

- Crystal amplifier and oscillator (OSC)
- Phase-locked loop (PLL)
- Reduced frequency divider (RFD)
- Status and control registers
- Control logic

7.2 Features

Features of the clock module include the following:

- Crystal input
- On-chip PLL
- Provides clock for integrated EPHY

7.3 Modes of Operation

The clock module can be operated in normal PLL mode or external clock mode (PLL disabled).

7.3.1 Normal PLL Mode

In normal PLL mode, the PLL is fully programmable. It can synthesize frequencies ranging from 4x to 18x the reference frequency and has a post divider capable of reducing this synthesized frequency without disturbing the PLL. The PLL reference can be a crystal oscillator or an external clock.

7.3.2 External Clock Mode

In external clock mode, the PLL is bypassed, and the external clock is applied to EXTAL. The resulting operating frequency is equal to the external clock frequency.

7.4 Low-power Mode Operation

This subsection describes the operation of the clock module in low-power and halted modes of operation. Low-power modes are described in Chapter 9, "Power Management." Table 7-1 shows the clock module operation in low-power modes.

Table 7-1. Clock Module Operation in Low-power Modes

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Exit not caused by clock module, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Exit not caused by clock module

In wait and doze modes, the system clocks to the peripherals are enabled and the clocks to the CPU and memory are stopped. Each module can disable its clock locally at the module level.

In stop mode, all system clocks are disabled. There are several options for enabling or disabling the PLL in stop mode, compromising between stop mode current and wakeup recovery time. The PLL can be disabled in stop mode, but requires a wakeup period before it can relock.

There is also a fast wakeup option for quickly enabling the system clocks during stop recovery. This eliminates the wakeup recovery time but at the risk of sending a potentially unstable clock to the system. To prevent a non-locked PLL frequency overshoot when using the fast wakeup option, change the RFD divisor to the current RFD value plus one before entering stop mode.

In external clock mode, there are no wakeup periods for oscillator startup or PLL lock.

7.5 Block Diagram

Figure 7-1 shows a block diagram of the entire clock module. The PLL block in this diagram is expanded in detail in Figure 7-2.

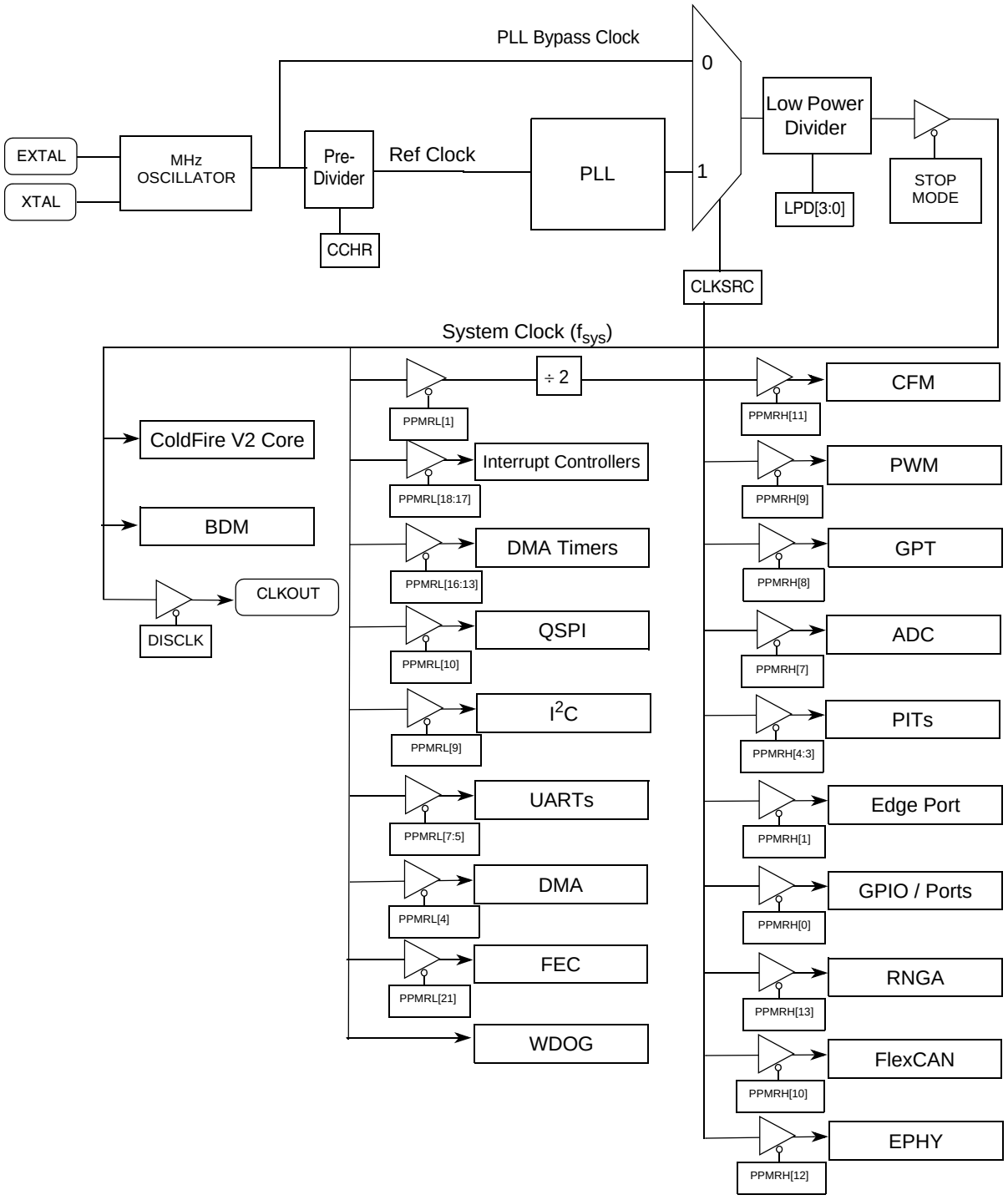


Figure 7-1. Clock Module Block Diagram

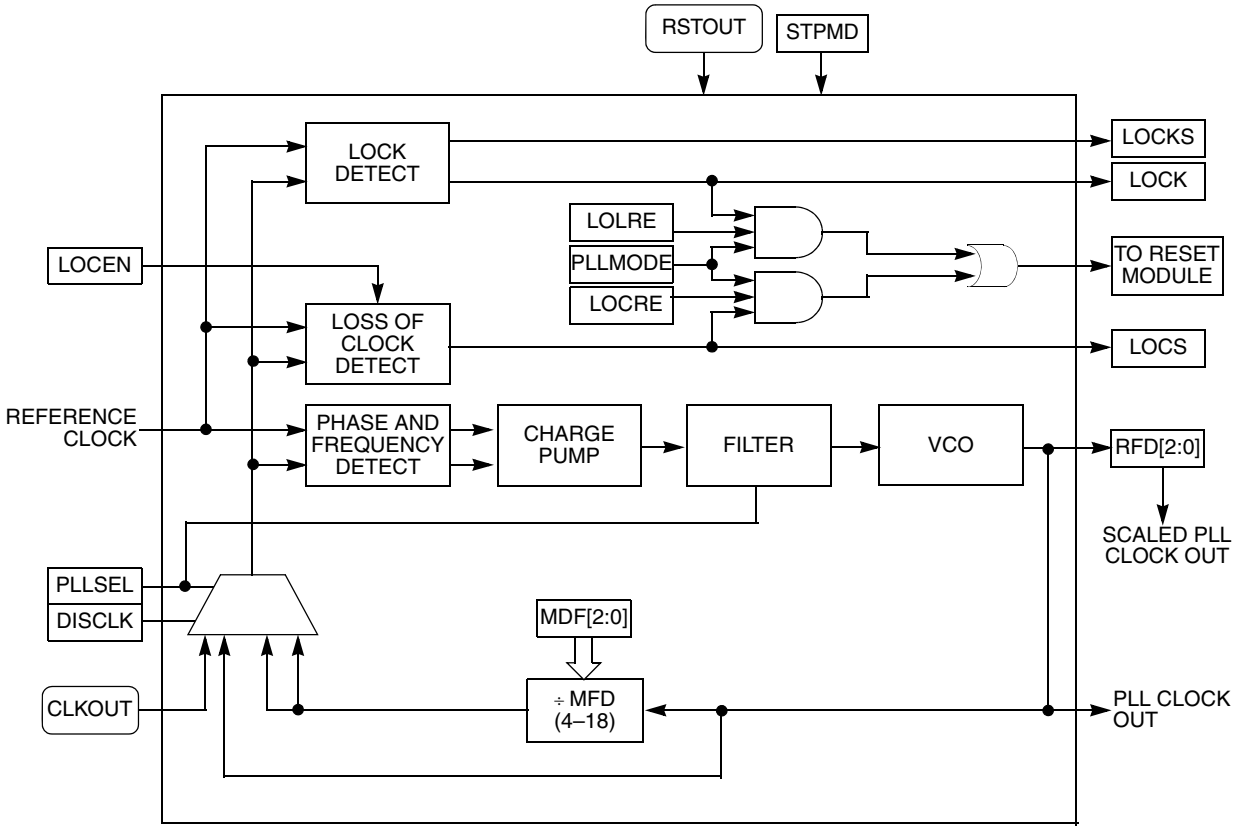


Figure 7-2. PLL Block Diagram

7.6 Signal Descriptions

The clock module signals are summarized in Table 7-2 and a brief description follows. For more detailed information, refer to Chapter 2, “Signal Descriptions.”

Table 7-2. Signal Properties

Name	Function
EXTAL	Oscillator or clock input
XTAL	Oscillator output
CLKOUT	System clock output
$\overline{\text{RSTO}}$	Reset signal from reset controller

7.6.1 EXTAL

This input is driven by an external clock except when used as a connection to the external crystal when using the internal oscillator.

7.6.2 XTAL

This output is an internal oscillator connection to the external crystal.

7.6.3 CLKOUT

This output reflects the internal system clock.

7.6.4 $\overline{\text{RSTO}}$

The $\overline{\text{RSTO}}$ pin is asserted by one of the following:

- Internal system reset signal
- FRCRSTOUT bit in the reset control status register (RCR); see Section 10.5.1, “Reset Control Register (RCR).”

7.7 Memory Map and Registers

The clock module programming model shown in Table 7-3 consists of registers that define clock operation and status as well as additional peripheral power management registers.

Table 7-3. Clock Module Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Mode Access Only					
0x0012_0000	Synthesizer Control Register (SYNCR)	16	R/W	0x1002	7.7.1.1/7-6
0x0012_0002	Synthesizer Status Register (SYNSR)	8	R	0x00	7.7.1.2/7-8
0x0012_0007	Low Power Control Register (LPCR)	8	R/W	0x00	7.7.1.3/7-9
0x0012_0008	Clock Control High Register (CCHR)	8	R/W	0x04	7.7.1.4/7-10
0x0012_000C	Real Time Clock Divide Register (RTCDR)	32	R/W	0x00000000	7.7.1.5/7-11
0x0000_000C	Peripheral Power Management Register High (PPMRH)[2]	32	R/W	0x00000000	9.2.1/9-2
0x0000_0008	Peripheral Power Management Register Low (PPMRL)[2]	32	R/W	0x00000001	9.2.1/9-2

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 See Section 9.2.1, “Peripheral Power Management Registers (PPMRH, PPMRL).”

7.7.1 Register Descriptions

This subsection provides a description of the clock module registers.

7.7.1.1 Synthesizer Control Register (SYNCR)

IPSBAR
Offset: 0x12_0000 (SYNCR)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8
R W	LOLRE	MFD2	MFD1	MFD0	LOCRE	RFD2	RFD1	RFD0
Reset	0	0	0	1	0	0	0	0

	7	6	5	4	3	2	1	0
R W	LOCEN	DISCLK	FWKUP	—	—	CLKSRC[1]	PLLMODE	PLLEN[1]
Reset	0	0	0	0	0	0	1	0

Figure 7-3. Synthesizer Control Register (SYNCR)

[1] The reset values of PLLEN and CLKSRC are zero, as the PLL is not enabled when the device emerges from reset).

Table 7-4. SYNCR Field Descriptions

Field	Description
15 LOLRE	Loss-of-lock reset enable. Determines how the system handles a loss-of-lock indication. When operating in normal mode, the PLL must be locked before setting the LOLRE bit. Otherwise, reset is immediately asserted. To prevent an immediate reset, the LOLRE bit must be cleared before writing the MFD[2:0] bits or entering stop mode with the PLL disabled. 0 No reset on loss of lock 1 Reset on loss of lock **Note:** In external clock mode, the LOLRE bit has no effect.

Table 7-4. SYNCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>14–12
MFD</td><td>Multiplication Factor Divider. Contain the binary value of the divider in the PLL feedback loop. The MFD[2:0] value is the multiplication factor applied to the reference frequency. When MFD[2:0] are changed or the PLL is disabled in stop mode, the PLL loses lock.
Note: In external clock mode, the MFD[2:0] bits have no effect.

The following table shows the system frequency multiplier of the reference frequency[1] in normal PLL mode.
<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="8">MFD[2:0]</th></tr>
<tr><th>000[2] (4x)</th><th>001 (6x)</th><th>010 (8x)[(3)]</th><th>011 (10x)</th><th>100 (12x)</th><th>101 (14x)</th><th>110 (16x)</th><th>111 (18x)</th></tr>
<tr><td rowspan="8">RFD[2:0]</td><td>000 (÷ 1)</td><td>4</td><td>6</td><td>8</td><td>10</td><td>12</td><td>14</td><td>16</td><td>18</td></tr>
<tr><td>001 (÷ 2)[3]</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td>010 (÷ 4)</td><td>1</td><td>3/2</td><td>2</td><td>5/2</td><td>3</td><td>7/2</td><td>4</td><td>9/2</td></tr>
<tr><td>011 (÷ 8)</td><td>1/2</td><td>3/4</td><td>1</td><td>5/4</td><td>3/2</td><td>7/4</td><td>2</td><td>9/4</td></tr>
<tr><td>100 (÷ 16)</td><td>1/4</td><td>3/8</td><td>1/2</td><td>5/8</td><td>3/4</td><td>7/8</td><td>1</td><td>9/8</td></tr>
<tr><td>101 (÷ 32)</td><td>1/8</td><td>3/16</td><td>1/4</td><td>5/16</td><td>3/8</td><td>7/16</td><td>1/2</td><td>9/16</td></tr>
<tr><td>110 (÷ 64)</td><td>1/16</td><td>3/32</td><td>1/8</td><td>5/32</td><td>3/16</td><td>7/32</td><td>1/4</td><td>9/32</td></tr>
<tr><td>111 (÷ 128)</td><td>1/32</td><td>3/64</td><td>1/16</td><td>5/64</td><td>3/32</td><td>7/64</td><td>1/8</td><td>9/64</td></tr>
</table>
[1] $f_{sys} = f_{ref}$ x 2(MFD + 2)/(2 exp RFD); f_{ref} x 2(MFD + 2) ≤ 60 MHz, f_{sys} ≤ MHz
[2] MFD = 000 not valid for f_{ref} < 3 MHz
[3] Default value out of reset</td></tr>
<tr><td>11
LOCRE</td><td>Loss-of-clock reset enable. Determines how the system handles a loss-of-clock condition. When the LOCEN bit is clear, LOCRE has no effect. If the LOCS flag in SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes an immediate reset. To prevent an immediate reset, the LOCRE bit must be cleared before entering stop mode with the PLL disabled.
0 No reset on loss-of-clock
1 Reset on loss-of-clock
Note: In external clock mode, the LOCRE bit has no effect.</td></tr>
<tr><td>10–8
RFD</td><td>Reduced frequency divider field. The binary value written to RFD[2:0] is the PLL frequency divisor; see table in MFD bit description. Changes in clock frequency are synchronized to the next falling edge of the current system clock.
Note: Changing RFD may cause a glitch in the PLL clock. To ensure correct operation of the PLL:
1. Bypass the PLL
2. Update RFD.
3. Select the PLL clock again.
The LPD (Low Power Divider) register can also be used to change system clock frequency.</td></tr>
<tr><td>7
LOCEN</td><td>Enables the loss-of-clock function. LOCEN does not affect the loss-of-lock function.
0 Loss-of-clock function disabled
1 Loss-of-clock function enabled
Note: In external clock mode, the LOCEN bit has no effect.</td></tr>
<tr><td>6
DISCLK</td><td>Disable CLKOUT determines whether CLKOUT is driven. Setting the DISCLK bit holds CLKOUT low.
0 CLKOUT enabled
1 CLKOUT disabled</td></tr>
</table>

Table 7-4. SYNCR Field Descriptions (continued)

Field	Description
5 FWKUP	Fast wakeup. Determines when the system clocks are enabled during wakeup from stop mode. 0 System clocks enabled only when PLL is locked or operating normally 1 System clocks enabled on wakeup regardless of PLL lock status **Note:** When FWKUP equals 0, if the PLL or oscillator is enabled and unintentionally lost in stop mode, the PLL wakes up in self-clocked mode or reference clock mode depending on the clock that was lost. In external clock mode, the FWKUP bit has no effect on the wakeup sequence.
4–3 —	Reserved, should be cleared.
2 CLKSRC	Clock Source. Determines whether the PLL output clock or the PLL reference clock is to drive the system clock. This bit is ignored when the PLL is disabled, in which case the PLL reference clock drives the system clock. Having this separate bit allows the PLL to first be enabled, and then the system clock can be switched to the PLL output clock only after the PLL has locked. When disabling the PLL, the clock can be switched before disabling the PLL so that a smooth transfer is ensured. 0) PLLreference clock (input clock) drives the system clock. 1) PLL output clock drives the system clock (provided the PLL is enabled).
1 PLLMODE	Determines the operating mode of the PLL. This bit should only be changed after reset with the PLL disabled. 0) Reserved, should be cleared. 1) PLL operates in normal mode
0 PLLEN	Enables and disables the PLL. If the PLL is enabled out of reset the chip does not leave the reset state until the PLL is locked and the system clock is driven by the PLL output clock. Use the CLKSRC control bit to switch the system clock between the PLL output clock and PLL bypass clock after the PLL is enabled. 0) PLL is disabled 1) PLL is enabled

7.7.1.2 Synthesizer Status Register (SYNSR)

The SYNSR is a read-only register that can be read at any time. Writing to the SYNSR has no effect and terminates the cycle normally.

IPSBAR Access: Supervisor read/write

Offset: 0x12_0002 (SYNSR)

	7	6	5	4	3	2	1	0
R	EXTOSC	—	—	LOCKS	LOCK	LOCS	—	—
W								
Reset:	1	0	0	See note 1	See note 1	0	0	0

Note: 1. See the LOCKS and LOCK bit descriptions.

Figure 7-4. Synthesizer Status Register (SYNSR)

Table 7-5. SYNSR Field Descriptions

Field	Description
7 EXTOSC	Indicates if an external oscillator is providing the reference clock source 0) Reference clock is not external oscillator 1 Reference clock is external oscillator
6–5	Reserved, should be cleared.

Table 7-5. SYNSR Field Descriptions (continued)

Field	Description
4 LOCKS	Sticky indication of PLL lock status. 0 PLL loss of lock since last system reset or MFD change or currently not locked due to exit from STOP with FWKUP set 1 No unintentional PLL loss of lock since last system reset or MFD change The lock detect function sets the LOCKS bit when the PLL achieves lock after: • A system reset • A write to SYNCR that changes the MFD[2:0] bits When the PLL loses lock, LOCKS is cleared. When the PLL relocks, LOCKS remains cleared until one of the two listed events occurs. In stop mode, if the PLL is intentionally disabled, then the LOCKS bit reflects the value prior to entering stop mode. However, if FWKUP is set, then LOCKS is cleared until the PLL regains lock. After lock is regained, the LOCKS bit reflects the value prior to entering stop mode. Furthermore, reading the LOCKS bit at the same time that the PLL loses lock does not return the current loss of lock condition. In external clock mode, LOCKS remains cleared after reset. In normal PLL mode, LOCKS is set after reset.
3 LOCK	Set when the PLL is locked. PLL lock occurs when the synthesized frequency is within approximately 0.75% of the programmed frequency. The PLL loses lock when a frequency deviation of greater than approximately 1.5% occurs. Reading the LOCK flag at the same time that the PLL loses lock or acquires lock does not return the current condition of the PLL. The power-on reset circuit uses the LOCK bit as a condition for releasing reset. If operating in external clock mode, LOCK remains cleared after reset. 0 PLL not locked 1 PLL locked
2 LOCS	Sticky indication of whether a loss-of-clock condition has occurred at any time since exiting reset in normal PLL mode. • LOCS equals 0 when the system clocks are operating normally. • LOCS equals 1 when system clocks have failed due to a reference failure or PLL failure. After entering stop mode with FWKUP set and the PLL and oscillator intentionally disabled (STPMD[1:0] = 11), the PLL exits stop mode in the SCM while the oscillator starts up. During this time, LOCS is temporarily set regardless of LOCEN. It is cleared after the oscillator comes up and the PLL is attempting to lock. If a read of the LOCS flag and a loss-of-clock condition occur simultaneously, the flag does not reflect the current loss-of-clock condition. A loss-of-clock condition can be detected only if LOCEN equals 1 or the oscillator has not yet returned from exit from stop mode with FWKUP equaling 1. 0 Loss-of-clock not detected since exiting reset 1 Loss-of-clock detected since exiting reset or oscillator not yet recovered from exit from stop mode with FWKUP equaling 1 **Note:** The LOCS flag is always 0 in external clock mode.
1–0	Reserved, should be cleared.

7.7.1.3 Low Power Control Register (LPCR)

The low power control register (LPCR) controls the low-power divider. It contains a 4-bit field that divides down the system clock (regardless if the reference clock or PLL clock is driving the system clock) by a factor of 2^n (where n is a number from 0 to 15 represented by the 4 bit field). The clock change takes effect with the next rising edge of the system clock.

IPSBAR Access: Supervisor read/write
Offset: 0x12_0007 (LPCR)

	7	6	5	4	3	2	1	0
R	—	—	—	—	LPD3	LPD2	LPD1	LPD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 7-5. Low Power Control Register (LPCR)

Table 7-6. LPCR Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 LPD	Low-power divider factor (divides the system clock by a factor of 2^{LPD}).

7.7.1.4 Clock Control High Register (CCHR)

The CCHR sets the pre-division factor, which divides down the PLL input clock by 1 (CCHR[2:0] = 000) to 8 (CCHR[2:0] =111). This allows an external oscillator or crystal of more than 10 MHz to be used with the PLL. The division factor should be set to generate an input clock for the PLL, refer to the device datasheet for crystal oscillator frequency range in normal PLL mode. When CCHR[2:0] are changed or the PLL is disabled in stop mode, the PLL loses lock.

NOTE

The CCHR can be written at any time. However, changes take effect only after the PLL is disabled and re-enabled.

IPSBAR Access: Supervisor read/write
Offset: 0x12_0008 (CCHR)

	7	6	5	4	3	2	1	0
R	—	—	—	—	—	CCHR2	CCHR1	CCHR0
W								
Reset:	0	0	0	0	0	1	0	0

Figure 7-6. Clock Control High Register (CCHR)

Table 7-7. CCHR Field Descriptions

Field	Description
7–3	Reserved, should be cleared.
2–0 CCHR	Clock control pre-division factor (divides the PLL input clock by a factor of CCHR+1).

7.7.1.5 Real Time Clock Divide Register (RTCDR)

The Real Time Clock Divide Register is a 32 bit read/write register that divides down the oscillator clock to a 1 Hz clock for the Real Time Clock module. If this register is programmed with zero then the clock to the Real Time Clock module is disabled, otherwise the oscillator clock is divided by one more than the value written to the register field (between 2 and 4,294,967,296).

IPSBAR
Offset: 0x0011_000C (RTCDR)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24
R W	RTCDF (31:24)							
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R W	RTCDF (23:16)							
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R W	RTCDF (15:8)							
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R W	RTCDF (7:0)							
Reset	0	0	0	0	0	0	0	0

Figure 7-7. Real Time Clock Divide Register (RTCDR)

Table 7-8. RTCDR Field Descriptions

Field	Description
31–0 RTCDF	Real-time clock divide factor (divides the oscillator clock by a factor of RTCDF+1). **Note:** If RTCDF equals 0x0, the clock to the RTC module is disabled.

7.8 Functional Description

This section provides a functional description of the clock module.

7.8.1 Clock Operation During Reset

The PLL is always disabled as the part emerges from Reset, with a default configuration of external crystal mode (although this mode also supports an external clock source). After out of reset, it is not possible to change the input clock source, although it is possible to enable the PLL and switch between the PLL clock and the oscillator clock as the source of the system clock.

7.8.2 System Clock Generation

In normal PLL clock mode, the default system frequency is two times the reference frequency (i.e., clock frequency divided by the pre-division factor specified by CCHR) after reset. The RFD[2:0] and MFD[2:0] bits in the SYNCR select the frequency multiplier. The LPD[3:0] field in the LPCR register provides additional settings for dividing down the system clock (including when the PLL is disabled) for low power operation.

When programming the PLL, do not exceed the maximum system clock frequency listed in the electrical specifications. Use this procedure to accommodate the frequency overshoot that occurs when the MFD bits are changed:

1. Determine the appropriate value for the MFD and RFD fields in the SYNCR. The amount of jitter in the system clocks can be minimized by selecting the maximum MFD factor that can be paired with an RFD factor to provide the required frequency.
2. Write a value of 1 + RFD (from step 1) to the RFD field of the SYNCR.
3. Write the MFD value from step 1 to the SYNCR.
4. Monitor the LOCK flag in SYNSR. When the PLL achieves lock, write the RFD value from step 1 to the RFD field of the SYNCR. This changes the system clocks frequency to the required frequency.

NOTE

Keep the maximum system clock frequency below the limit given in the electrical characteristics.

7.8.3 PLL Operation

In PLL mode, the PLL synthesizes the system clocks. The PLL can multiply the reference clock frequency by 4x to 18x, provided that the system clock frequency remains within the range listed in electrical specifications. For example, if the reference frequency is 2 MHz, the PLL can synthesize frequencies of 8 MHz to 36 MHz. In addition, the RFD can reduce the system frequency by dividing the output of the PLL.

Figure 7-8 shows the external support circuitry for the crystal oscillator with example component values. Actual component values depend on crystal specifications.

The following subsections describe each major block of the PLL. Refer to Figure 7-8 to see how these functional sub-blocks interact.

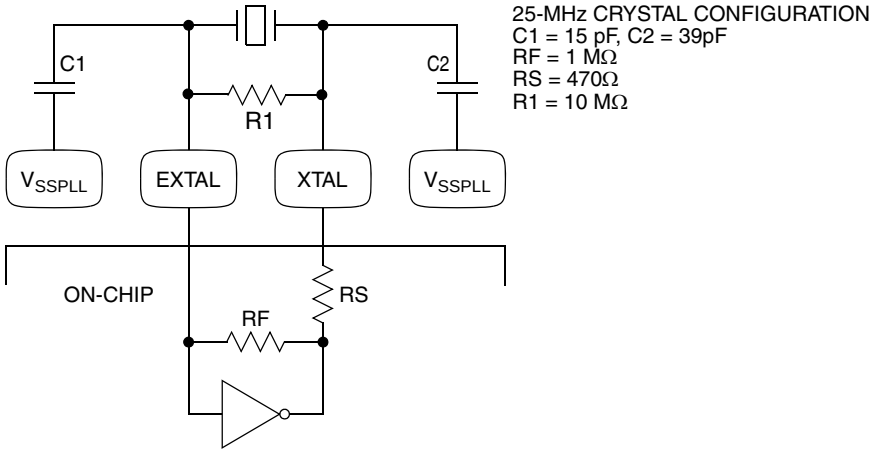


Figure 7-8. Crystal Oscillator Example

7.8.3.1 Phase and Frequency Detector (PFD)

The PFD is a dual-latch phase-frequency detector. It compares the phase and frequency of the reference and feedback clocks. The reference clock comes from the crystal oscillator or an external clock source.

- The feedback clock comes from the VCO output divided by the MFD in normal PLL mode.

When the frequency of the feedback clock equals the frequency of the reference clock, the PLL is frequency-locked. If the falling edge of the feedback clock lags the falling edge of the reference clock, the PFD pulses the UP signal. If the falling edge of the feedback clock leads the falling edge of the reference clock, the PFD pulses the DOWN signal. The width of these pulses relative to the reference clock depends on how much the two clocks lead or lag each other. After phase lock is achieved, the PFD continues to pulse the UP and DOWN signals for very short durations during each reference clock cycle. These short pulses continually update the PLL and prevent the frequency drift phenomenon known as dead-banding.

7.8.3.2 Charge Pump/Loop Filter

In normal mode the current magnitude of the charge pump varies with the MFD as shown in Table 7-9.

Table 7-9. Charge Pump Current and MFD in Normal Mode Operation

Charge Pump Current	MFD
1x	0 ≤ MFD < 2
2x	2 ≤ MFD < 6
4x	6 ≤ MFD

The UP and DOWN signals from the PFD control whether the charge pump applies or removes charge, respectively, from the loop filter. The filter is integrated on the chip.

7.8.3.3 Voltage Control Output (VCO)

The voltage across the loop filter controls the frequency of the VCO output. The frequency-to-voltage relationship (VCO gain) is positive, and the output frequency is four times the target system frequency.

7.8.3.4 Multiplication Factor Divider (MFD)

The MFD divides the output of the VCO and feeds it back to the PFD. The PFD controls the VCO frequency via the charge pump and loop filter such that the reference and feedback clocks have the same frequency and phase. Thus, the frequency of the input to the MFD, which is also the output of the VCO, is the reference frequency multiplied by the same amount that the MFD divides by. For example, if the MFD divides the VCO frequency by six, the PLL is frequency locked when the VCO frequency is six times the reference frequency. The presence of the MFD in the loop allows the PLL to perform frequency multiplication, or synthesis.

7.8.3.5 PLL Lock Detection

The lock detect logic monitors the reference frequency and the PLL feedback frequency to determine when frequency lock is achieved. Phase lock is inferred by the frequency relationship, but is not guaranteed. The LOCK flag in the SYNSR reflects the PLL lock status. A sticky lock flag, LOCKS, is also provided.

The lock detect function uses two counters: one is clocked by the reference, and the other is clocked by the PLL feedback. When the reference counter has counted N cycles, its count is compared to that of the feedback counter. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then, if the two counters continue to match, the lock criteria is relaxed by 1/2 and the system is notified that the PLL has achieved frequency lock.

After lock is detected, the lock circuit continues to monitor the reference and feedback frequencies using the alternate count and compare process. If the counters do not match at any comparison time, then the LOCK flag is cleared to indicate that the PLL has lost lock. At this point, the lock criteria is tightened and the lock detect process is repeated.

The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and non-locked status due to phase sensitivities. Figure 7-9 shows the sequence for detecting locked and non-locked conditions.

In external clock mode, the PLL is disabled and cannot lock.

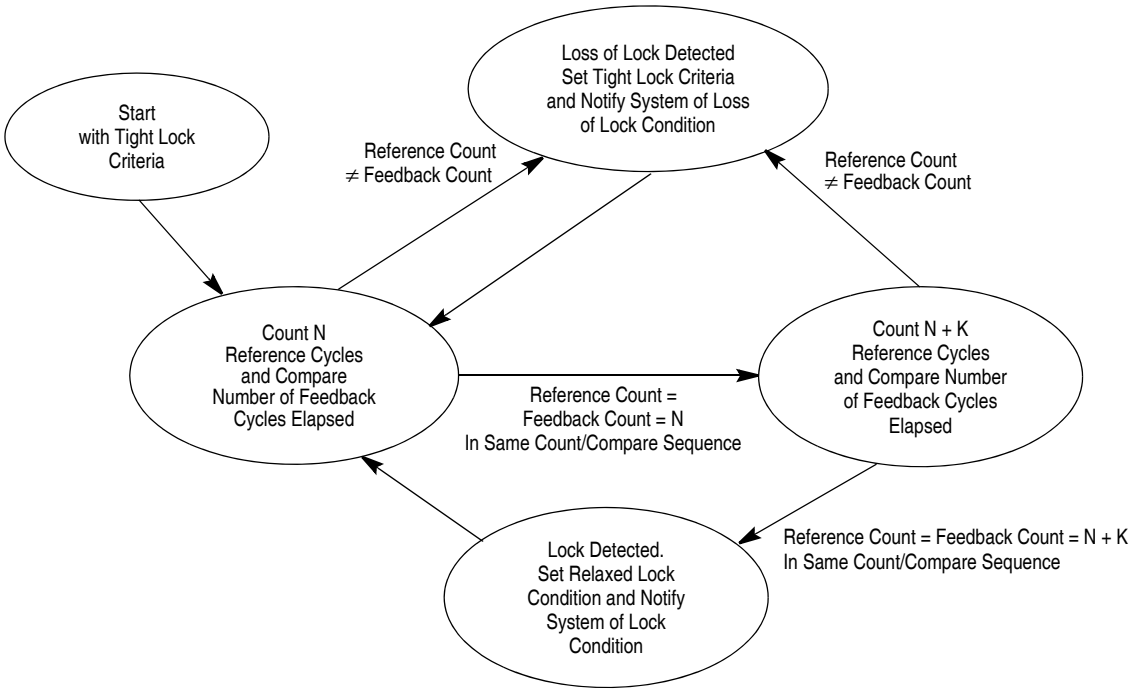


Figure 7-9. Lock Detect Sequence

7.8.3.6 PLL Loss of Lock Conditions

After the PLL acquires lock after reset, the LOCK and LOCKS flags are set. If the MFD is changed, or if an unexpected loss of lock condition occurs, the LOCK and LOCKS flags are negated. While the PLL is in the non-locked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to relock. Consequently, during the relocking process, the system clocks frequency is not well defined and may exceed the maximum system frequency, violating the system clock timing specifications.

However, after the PLL has relocked, the LOCK flag is set. The LOCKS flag remains cleared if the loss of lock was unexpected. The LOCKS flag is set when the loss of lock is caused by changing MFD. If the PLL is intentionally disabled during stop mode, then after exit from stop mode, the LOCKS flag reflects the value prior to entering stop mode after lock is regained.

7.8.3.7 PLL Loss of Lock Reset

If the LOLRE bit in the SYNCR is set, a loss of lock condition asserts reset. Reset reinitializes the LOCK and LOCKS flags. Therefore, software must read the LOL bit in the reset status register (RSR) to determine if a loss of lock caused the reset. See Section 10.5.2, “Reset Status Register (RSR).”

To exit reset in PLL mode, the reference must be present, and the PLL must achieve lock.

In external clock mode, the PLL cannot lock. Therefore, a loss of lock condition cannot occur, and the LOLRE bit has no effect.

7.8.3.8 Loss of Clock Detection

The LOCEN bit in the SYNCR enables the loss of clock detection circuit to monitor the input clocks to the phase and frequency detector (PFD). When the reference or feedback clock frequency falls below the minimum frequency, the loss of clock circuit sets the sticky LOCS flag in the SYNSR.

NOTE

In external clock mode, the loss of clock circuit is disabled.

7.8.3.9 Loss of Clock Reset

The clock module can assert a reset when a loss of clock or loss of lock occurs. When a loss-of-clock condition is recognized, reset is asserted if the LOCRE bit in SYNCR is set. The LOCS bit in SYNSR is cleared after reset. Therefore, the LOC bit must be read in RSR to determine that a loss of clock condition occurred. LOCRE has no effect in external clock mode.

To exit reset in PLL mode, the reference must be present, and the PLL must acquire lock.

Reset initializes the clock module registers to a known startup state as described in Section 7.7, “Memory Map and Registers.”

7.8.3.10 Alternate Clock Selection

Depending on which clock source fails, the loss-of-clock circuit switches the system clocks source to the remaining operational clock. The alternate clock source generates the system clocks until reset is asserted. As Table 7-10 shows, if the reference fails, the PLL goes out of lock and into self-clocked mode (SCM). The PLL remains in SCM until the next reset. When the PLL is operating in SCM, the system frequency depends on the value in the RFD field. The SCM system frequency stated in electrical specifications assumes that the RFD has been programmed to binary 000. If the loss-of-clock condition is due to PLL failure, the PLL reference becomes the system clocks source until the next reset, even if the PLL regains and relocks.

Table 7-10. Loss of Clock Summary

Clock Mode	System Clock Source Before Failure	Reference Failure Alternate Clock Selected by LOC Circuit[1] Until Reset	PLL Failure Alternate Clock Selected by LOC Circuit Until Reset
PLL	PLL	PLL self-clocked mode	PLL reference
External	External clock	None	NA

1 The LOC circuit monitors the reference and feedback inputs to the PFD. See Figure 7-8.

A special loss-of-clock condition occurs when the reference and the PLL fail. The failures may be simultaneous, or the PLL may fail first. In either case, the reference clock failure takes priority and the PLL attempts to operate in SCM. If successful, the PLL remains in SCM until the next reset. If the PLL cannot operate in SCM, the system remains static until the next reset. The reference and the PLL must be functioning properly to exit reset.

7.8.3.11 Loss of Clock in Stop Mode

Table 7-11 shows the resulting actions for a loss of clock in stop mode when the device is being clocked by the various clocking methods.

Table 7-11. Stop Mode Operation

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
EXT	X	X	X	X	X	X	—	—	EXT	0	0	0	
								Lose reference clock	Stuck	—	—	—	
NRM	0	0	0	Off	Off	0	Lose lock, f.b. clock, reference clock	Regain	NRM	LK	1	LC	
								No regain	Stuck	—	—	—	
NRM	X	0	0	Off	Off	1	Lose lock, f.b. clock, reference clock	Regain clocks, but don't regain lock	SCM→ unstable NRM	0→LK	0→1	1→LC	Block LOCS and LOCKS until clock and lock respectively regain; enter SCM regardless of LOCEN bit until reference regained
								No reference clock regain	SCM→	0→	0→	1→	Block LOCS and LOCKS until clock and lock respectively regain; enter SCM regardless of LOCEN bit
								No f.b. clock regain	Stuck	—	—	—	
NRM	0	0	0	Off	On	0	Lose lock	Regain	NRM	LK	1	LC	Block LOCKS from being cleared
								Lose reference clock or no lock regain	Stuck	—	—	—	
								Lose reference clock, regain	NRM	LK	1	LC	Block LOCKS from being cleared

Table 7-11. Stop Mode Operation (continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	0	0	0	Off	On	1	Lose lock	No lock regain	Unstable NRM	0→LK	0→1	LC	Block LOCKS until lock regained
								Lose reference clock or no f.b. clock regain	Stuck	—	—	—	
								Lose reference clock, regain	Unstable NRM	0→LK	0→1	LC	LOCS not set because LOCEN = 0
NRM	0	0	0	On	On	0	—	—	NRM	LK	1	LC	
								Lose lock or clock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	LC	
								Lose clock and lock, regain	NRM	0	1	LC	LOCS not set because LOCEN = 0
NRM	0	0	0	On	On	1	—	—	NRM	LK	1	LC	
								Lose lock	Unstable NRM	0	0→1	LC	
								Lose lock, regain	NRM	0	1	LC	
								Lose clock	Stuck	—	—	—	
								Lose clock, regain without lock	Unstable NRM	0	0→1	LC	
								Lose clock, regain with lock	NRM	0	1	LC	
NRM	X	X	1	Off	X	X	Lose lock, f.b. clock, reference clock	RESET	RESET	—	—	—	Reset immediately
NRM	0	0	1	On	On	X	—	—	NRM	LK	1	LC	
								Lose lock or clock	RESET	—	—	—	Reset immediately
NRM	1	0	0	Off	Off	0	Lose lock, f.b. clock, reference clock	Regain	NRM	LK	1	LC	REF not entered during stop; SCM entered during stop only during oscillator startup
								No regain	Stuck	—	—	—	

Table 7-11. Stop Mode Operation (continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	1	0	0	Off	On	0	Lose lock, f.b. clock	Regain	NRM	LK	1	LC	REF mode not entered during stop
								No f.b. clock or lock regain	Stuck	—	—	—	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
NRM	1	0	0	Off	On	1	Lose lock, f.b. clock	Regain f.b. clock	Unstable NRM	0→LK	0→1	LC	REF mode not entered during stop
								No f.b. clock regain	Stuck	—	—	—	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
NRM	1	0	0	On	On	0	—	—	NRM	LK	1	LC	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
								Lose f.b. clock	REF	0	X	1	Wakeup without lock
								Lose lock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	LC	
NRM	1	0	0	On	On	1	—	—	NRM	LK	1	LC	
								Lose reference clock	SCM	0	0	1	Wakeup without lock
								Lose f.b. clock	REF	0	X	1	Wakeup without lock
								Lose lock	Unstable NRM	0	0→1	LC	
NRM	1	0	1	On	On	X	—	—	NRM	LK	1	LC	
								Lose lock or clock	RESET	—	—	—	Reset immediately
NRM	1	1	X	Off	X	X	Lose lock, f.b. clock, reference clock	RESET	RESET	—	—	—	Reset immediately

Table 7-11. Stop Mode Operation (continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
NRM	1	1	0	On	On	0	—	—	NRM	LK	1	LC	
								Lose clock	RESET	—	—	—	Reset immediately
								Lose lock	Stuck	—	—	—	
								Lose lock, regain	NRM	0	1	LC	
NRM	1	1	0	On	On	1	—	—	NRM	LK	1	LC	
								Lose clock	RESET	—	—	—	Reset immediately
								Lose lock	Unstable NRM	0	0→1	LC	
								Lose lock, regain	NRM	0	1	LC	
NRM	1	1	1	On	On	X	—	—	NRM	LK	1	LC	
								Lose clock or lock	RESET	—	—	—	Reset immediately
REF	1	0	0	X	X	X	—	—	REF	0	X	1	
								Lose reference clock	Stuck	—	—	—	
SCM	1	0	0	Off	X	0	PLL disabled	Regain SCM	SCM	0	0	1	Wakeup without lock
SCM	1	0	0	Off	X	1	PLL disabled	Regain SCM	SCM	0	0	1	
SCM	1	0	0	On	On	0	—	—	SCM	0	0	1	Wakeup without lock
								Lose reference clock	SCM				

Table 7-11. Stop Mode Operation (continued)

MODE In	LOCEN	LOCRE	LOLRE	PLL	OSC	FWKUP	Expected PLL Action at Stop	PLL Action During Stop	MODE Out	LOCKSS	LOCK	LOCS	Comments
SCM	1	0	0	On	On	1	—	—	SCM	0	0	1	
								Lose reference clock	SCM				

Note:

PLL = PLL enabled during STOP mode. PLL = On when STPMD[1:0] = 00 or 01

OSC = oscillator enabled during STOP mode. Oscillator is on when STPMD[1:0] = 00, 01, or 10

MODES

NRM = normal PLL crystal clock reference or normal PLL external reference. During normal external reference mode, the oscillator is never enabled. Therefore, during these modes, refer to the OSC = On case regardless of STPMD values.

EXT = external clock mode

REF = PLL reference mode due to losing PLL clock or lock from NRM mode

SCM = PLL self-clocked mode due to losing reference clock from NRM mode

RESET = immediate reset

LOCKS

LK -= expecting previous value of LOCKS before entering stop

0→LK = current value is 0 until lock is regained which then is the previous value before entering stop

0→ = current value is 0 until lock is regained but lock is never expected to regain

LOCS

LC = expecting previous value of LOCS before entering stop

1→LC = current value is 1 until clock is regained which then is the previous value before entering stop

1→ = current value is 1 until clock is regained but CLK is never expected to regain

Chapter 8
Real-Time Clock

8.1 Introduction

This section discusses how to operate and program the real-time clock (RTC) module that maintains the system clock, provides stopwatch, alarm, and interrupt functions, and supports the following features.

8.1.1 Overview

Figure 8-1 is a block diagram of the Real-Time Clock (RTC) module. It consists of the following blocks:

- Time-of-day (TOD) clock counter
- Alarm
- Minute stopwatch
- Associated control and bus interface hardware

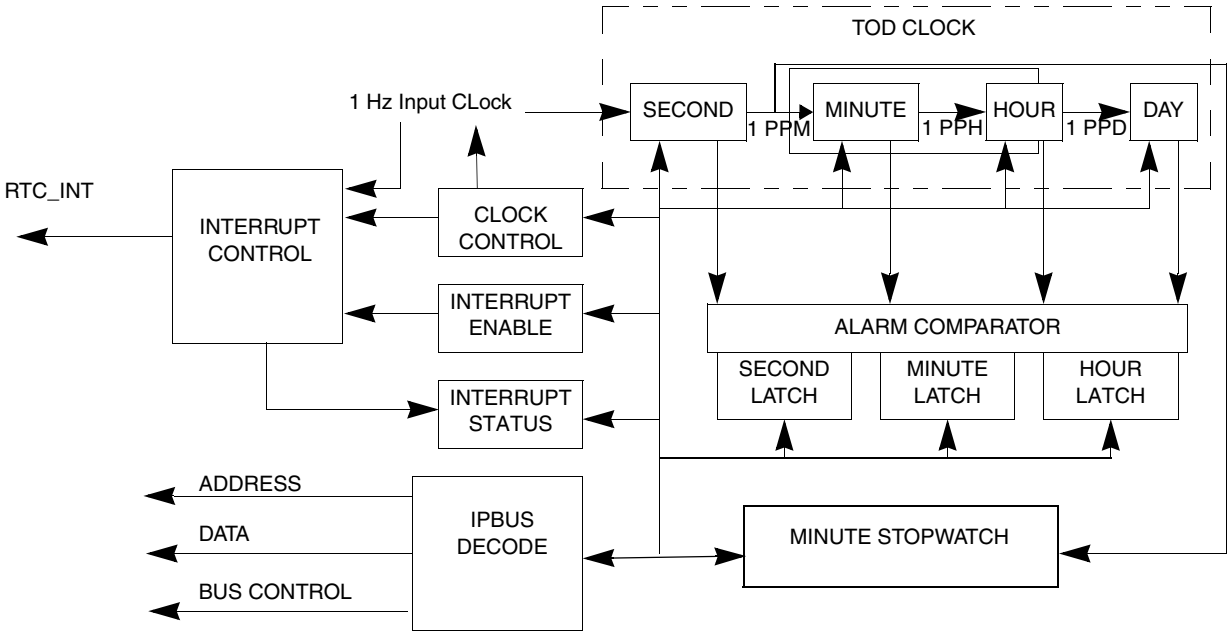


Figure 8-1. Real-Time Clock Block Diagram

8.1.2 Features

The RTC module includes the following features:

- Full clock—days, hours, minutes, seconds

- Minute countdown timer with interrupt
- Programmable daily alarm with interrupt
- Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts

8.1.3 Modes of Operation

The incoming 1 Hz signal is used to increment the seconds, minutes, hours, and days TOD counters. The alarm functions, when enabled, generate RTC interrupts when the TOD settings reach programmed values. The sampling timer generates fixed-frequency interrupts, and the minute stopwatch allows for efficient interrupts on minute boundaries.

- Counter

  The counter portion of the RTC module consists of four groups of counters that are physically located in three registers:

  — The 6-bit seconds counter is located in the SECONDS register

  — The 6-bit minutes counter and the 5-bit hours counter are located in the HOURMIN register

  — The 16-bit day counter is located in the DAYR register
- Alarm

  There are three alarm registers that mirror the three counter registers. An alarm is set by accessing the real-time clock alarm registers (ALRM_HM, ALRM_SEC, and DAYALARM) and loading the exact time that the alarm should generate an interrupt. When the TOD clock value and the alarm value coincide, an interrupt occurs one half second later.
- Minute Stopwatch

  The minute stopwatch performs a countdown with a one minute resolution. It can be used to generate an interrupt on a minute boundary.

8.2 Memory Map/Register Definition

The RTC module includes 10 32-bit registers. Table 8-1 summarizes these registers and their addresses.

Table 8-1. RTC Module Register Memory Map

IPSBAR Offset	Use	Access
0x03C0	RTC Hours and Minutes Counter Register (HOURMIN)	read/write
0x03C4	RTC Seconds Counter Register (SECONDS)	read/write
0x03C8	RTC Hours and Minutes Alarm Register (ALRM_HM)	read/write
0x03CC	RTC Seconds Alarm Register (ALRM_SEC)	read/write
0x03D0	RTC Control Register (RTCCTL)	read/write
0x03D4	RTC Interrupt Status Register (RTCISR)	read/write
0x03D8	RTC Interrupt Enable Register (RTCIENR)	read/write
0x03DC	Stopwatch Minutes Register (STPWCH)	read/write
0x03E0	RTC Days Counter Register (DAYS)	read/write

Table 8-1. RTC Module Register Memory Map (continued)

IPSBAR Offset	Use	Access
0x03E4	RTC Day Alarm Register (ALRM_DAY)	read/write
0x03F0	Reserved	—

8.2.1 RTC Hours and Minutes Counter Register (HOURMIN)

The real-time clock hours and minutes counter register (HOURMIN) is used to program the hours and minutes for the TOD clock. It can be read or written at any time. After a write, the time changes to the new value. A power-on reset (POR) sets the RTC to the reset values shown in Figure 8-2.

IPSBAR Access: User read/write
Offset: 0x03C0 (HOURMIN)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	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

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

1 After power-on reset (POR)

Figure 8-2. RTC Hours and Minutes Counter Register (HOURMIN)

Table 8-2. HOURMIN Field Descriptions

Field	Description
31–13	Reserved, should be cleared.
12–8 HOURS	Hour setting; can be set to any value between 0 and 23.
7–6	Reserved, should be cleared.
5–0 MINUTES	Minutes setting; can be set to any value between 0 and 59.

8.2.2 RTC Seconds Counter Register (SECONDS)

The real-time clock seconds register (SECONDS) is used to program the seconds for the TOD clock. It can be read or written at any time. After a write, the time changes to the new value. A power-on reset (POR) sets the RTC to the reset values shown in Figure 8-3.

IPSBAR
Offset: 0x03C4 (SECONDS)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	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

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

Figure 8-3. RTC Seconds Counter Register (SECONDS)

[1] After power-on reset (POR)

Table 8-3. SECONDS Field Descriptions

Field	Description
31–6	Reserved, should be cleared.
5–0 SECONDS	Seconds setting; can be set to any value between 0 and 59.

8.2.3 RTC Hours and Minutes Alarm Register (ALRM_HM)

The real-time clock hours and minutes alarm (ALRM_HM) register is used to configure the hours and minutes setting for the alarm. The alarm settings can be read or written at any time.

IPSBAR
Offset: 0x03C8 (ALRM_HM)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-4. RTC Hours and Minutes Alarm Register (ALRM_HM)

Table 8-4. ALRM_HM Field Descriptions

Field	Description
31–13	Reserved, should be cleared.
12–8 HOURS	Alarm hour setting; can be set to any value between 0 and 23.
7–6	Reserved, should be cleared.
5–0 MINUTES	Alarm minute setting; can be set to any value between 0 and 59.

8.2.4 RTC Seconds Alarm Register (ALRM_SEC)

The real-time clock seconds alarm (ALRM_SEC) register is used to configure the seconds setting for the alarm. The alarm settings can be read or written at any time.

IPSBAR
Offset: 0x03CC (ALRM_SEC)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-5. RTC Seconds Alarm Register (ALRM_SEC)

Table 8-5. ALRM_SEC Field Descriptions

Field	Description
31–6	Reserved, should be cleared.
5–0 SECONDS	Alarm seconds setting; can be set to any value between 0 and 59.

8.2.5 RTC Control Register (RTCCTL)

The real-time clock control (RTCCTL) register is used to enable the real-time clock module and specify the reference frequency information for the prescaler.

IPSBAR
Offset: 0x03D0 (RTCCTL)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-6. RTC Control Register (RTCCTL)

Table 8-6. RTCCTL Field Descriptions

Field	Description
31–8	Reserved, should be cleared.
7 EN	RTC Enables/Disable bit. This bit enables/disables the RTC. The software reset bit (SWR) has no effect on this bit. Bit description 0 Disable the real-time clock 1 Enable the real-time clock
6–1	Reserved, should be cleared.
0 SWR	Software Reset bit. This bit resets the RTC to its default state. However, a software reset has no effect on the EN bit. 0 No effect 1 Reset the module to its default state

8.2.6 RTC Interrupt Status Register (RTCISR)

The real-time clock interrupt status register (RTCISR) indicates the status of the various real-time clock interrupts. When an event of the types included in this register occurs, then the bit is set in this register regardless of its corresponding interrupt enable bit. These bits are cleared by writing a 1 to them; this also clears the interrupt. Interrupts may occur while the system clock is idle or in sleep mode.

IPSBAR
Offset: 0x03D4 (RTCISR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	HR	1HZ	DAY	ALM	MIN	SW
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-7. RTC Interrupt Status Register (RTCISR)

Table 8-7. RTCISR Field Descriptions

Field	Description
31–6	Reserved, should be cleared.
5 HR	Hour flag bit. This bit indicates whether the hour counter has incremented. If enabled, this bit is set on every increment of the RTC hour counter. 0 No 1-hour interrupt occurred 1 A 1-hour interrupt has occurred
4 1HZ	1 Hz flag bit. This bit indicates whether the second counter has incremented. If enabled, this bit is set on every increment of the RTC second counter. 0 No 1 Hz interrupt occurred 1 A 1 Hz interrupt has occurred
3 DAY	Day flag bit. This bit indicates whether the day counter has incremented. If enabled, this bit is set on every increment of the RTC day counter. 0 No 24-hour rollover interrupt occurred 1 A 24-hour rollover interrupt has occurred
2 ALM	Alarm flag bit. This bit indicates that the RTC time matches the value in the alarm registers. The alarm reoccurs every 65536 days. For a single alarm, clear the interrupt enable for this bit in the interrupt service routine. 0 No alarm interrupt occurred 1 An alarm interrupt has occurred
1 MIN	Minute flag bit. This bit indicates that the minute counter has incremented. If enabled, this bit is set on every increment of the RTC minute counter. 0 No 1-minute interrupt occurred 1 A 1-minute interrupt has occurred
0 SW	Stopwatch flag bit. This bit indicates that the stopwatch countdown has timed out. 0 The stopwatch did not time out. 1 The stopwatch timed out.

8.2.7 RTC Interrupt Enable Register (RTCIENR)

The real-time clock interrupt enable register (RTCIENR) is used to enable/disable the various real-time clock interrupts. Masking an interrupt bit has no effect on its corresponding status bit.

IPSBAR Offset: 0x03D8 (RTCIENR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	HR	1HZ	DAY	ALM	MIN	SW
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-8. RTC Interrupt Enable Register (RTCIENR)

Table 8-8. RTCIENR Field Descriptions

Field	Description
31-6	Reserved, should be cleared.
5 HR	Hour interrupt enable bit. This bit enables/disables an interrupt when the hour counter of the real-time clock increments. 0 The 1-hour interrupt id disabled. 1 The 1-hour interrupt is enabled.
4 1HZ	1 Hz interrupt enable bit. This bit enables/disables an interrupt when the second counter of the real-time clock increments. 0 The 1 Hz interrupt is disabled. 1 The 1 Hz interrupt is enabled.
3 DAY	Day interrupt enable bit. This bit enables/disables an interrupt when the hours counter rolls over from 23 to 0 (midnight rollover). 0 The 24-hour interrupt is disabled. 1 The 24-hour interrupt is enabled.
2 ALM	Alarm interrupt enable bit. This bit enables/disables the alarm interrupt. 0 The alarm interrupt is disabled. 1 The alarm interrupt is enabled.
1 MIN	Minute interrupt enable bit. This bit enables/disables an interrupt when the RTC minute counter increments. 0 The 1-minute interrupt is disabled. 1 The 1-minute interrupt is enabled.
0 SW	Stopwatch interrupt enable; enables/disables the stopwatch interrupt. The stopwatch counts down and remains at decimal -1 until it is reprogrammed. If this bit is enabled with -1 (decimal) in the STPWCH register, an interrupt is posted on the next minute tick. Bit description 1 = Stopwatch interrupt is enabled. 0 = Stopwatch interrupt is disabled.

8.2.8 RTC Stopwatch Minutes Register (STPWCH)

The stopwatch minutes (STPWCH) register contains the current stopwatch countdown value. When the minute counter of the TOD clock increments, the value in this register decrements.

IPSBAR
Offset: 0x03DC (STPWCH)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-9. RTC Stopwatch Minutes Register (STPWCH)

Table 8-9. STPWCH Field Descriptions

Field	Description
31–6	Reserved, should be cleared.
5–0 CNT	Stopwatch count. This field contains the stopwatch countdown value. **Note:** The stopwatch counter is decremented by the minute (MIN) tick output from the real-time clock, so the average tolerance of the count is 0.5 minutes. For better accuracy, enable the stopwatch by polling the MIN bit of the RTCISR register or by polling the minute interrupt service routine.

8.2.9 RTC Days Counter Register (DAYS)

The real-time clock days counter register (DAYS) is used to program the day for the TOD clock. When the HOUR field of the HOURMIN register rolls over from 23 to 00, the day counter increments. It can be read or written at any time. After a write, the time changes to the new value. This register cannot be reset because the real-time clock is always enabled at reset. Only 16-bit accesses to this register are allowed.

IPSBAR
Offset: 0x03E0 (DAYS)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-10. RTC Days Counter Register (DAYS)

Table 8-10. DAYS Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 DAYS	Day Setting. This field indicates the current day count, and can be set to any value between 0 and 65535.

8.2.10 RTC Day Alarm Register (ALRM_DAY)

The real-time clock day alarm (ALRM_DAY) register is used to configure the day for the alarm. The alarm settings can be read or written at any time.

IPSBAR
Offset: 0x03E4 (ALRM_DAY)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 8-11. RTC Day Alarm Register (ALRM_DAY)

Table 8-11. ALRM_DAY Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 DAYSAL	Day Setting of the Alarm. This field can be set to any value between 0 and 65535.

8.3 Functional Description

The RTC uses a supplied 1 Hz signal to increment the seconds, minutes, hours, and days TOD counters. The alarm functions, when enabled, generate RTC interrupts when the TOD settings reach programmed values. The minute stopwatch allows for efficient interrupts on minute boundaries.

8.3.1 Prescaler and Counter

A 1 Hz clock is supplied to the RTC. This 1 Hz clock drives the RTC's counters. The counter portion of the RTC module consists of four groups of counters that are physically located in three registers:

- The 6-bit seconds counter is located in the SECONDS register
- The 6-bit minutes counter and the 5-bit hours counter are located in the HOURMIN register
- The 16-bit day counter is located in the DAYR register

These counters cover a 24-hour clock over 65536 days. All three registers can be read or written at any time.

Interrupts signal when each of the four counters increments, and can be used to indicate when a counter rolls over. For example, each tick of the seconds counter causes the 1HZ interrupt flag to be set. When the seconds counter rolls from 59 to 00, the minute counter increments and the MIN interrupt flag is set. The same is true for the minute counter with the HR signal, and the hour counter with the DAY signal.

8.3.2 Alarm

There are three alarm registers that mirror the three counter registers. An alarm is set by accessing the real-time clock alarm registers (ALRM_HM, ALRM_SEC, and DAYALARM) and loading the exact time that the alarm should generate an interrupt. If RTCIENR[ALM] is set, when the TOD clock value and the alarm value coincide, an interrupt occurs one half second later. If the alarm is not disabled, it reoccurs every 65536 days. If a single alarm is desired, the alarm function must be disabled through RTCIENR.

8.3.3 Minute Stopwatch

The minute stopwatch performs a countdown with a one minute resolution. It can be used to generate an interrupt on a minute boundary. At each minute, the value in the stopwatch is decremented. When the stopwatch value reaches -1, the interrupt occurs. The value of the register does not change until it is reprogrammed. The actual delay includes the seconds from setting the stopwatch to the next minute tick.

8.4 Initialization/Application Information

8.4.1 RTC Register Access Protocol

1. Perform dummy read access to a non-RTC address location.
2. Perform dummy read to the targeted RTC address location.
3. Perform actual read or write access to the targeted RTC address location.

Steps 1 and 2 can be combined into one cycle that is a dummy read access to a non-RTC but targeted RTC address location. Then the protocol is:

1. Perform dummy read access to a non-RTC, targeted RTC address location.
2. Perform actual read or write access to targeted RTC address location.

8.4.2 Flow Chart of RTC Operation

Figure 8-12 shows the flow chart of a typical RTC operation.

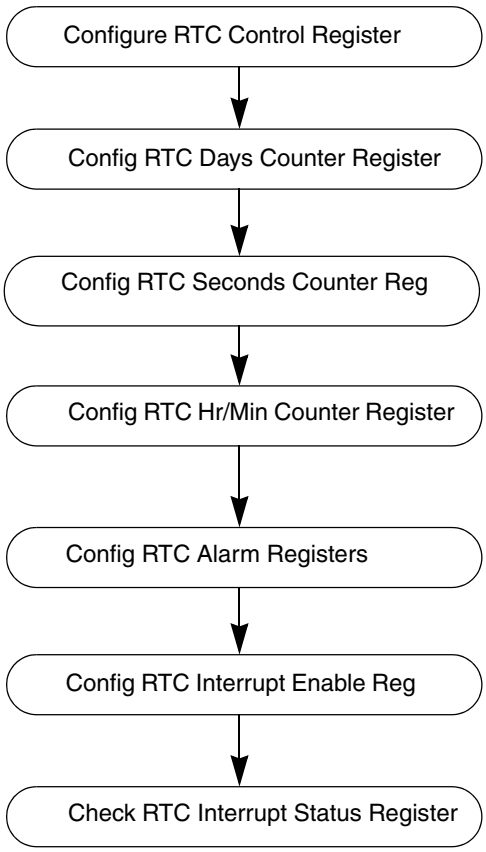


Figure 8-12. Flow Chart of RTC Operation

8.4.3 Code Example for Initializing the Real-Time Clock

Figure 8-13 shows sample code for initializing the RTC.

```
/* Set real-time clock freq */
MCF_CLOCK_RTCDR = 48000000;      /* Device-dependent */

MCF_RTC_HOURMIN = MCF_RTC_HOURMIN_HOURS(((uint32)time_temp % 24));
MCF_RTC_HOURMIN = MCF_RTC_HOURMIN_MINUTES(((uint32)time_temp % 60));
MCF_RTC_SECONDS = MCF_RTC_SECONDS_SECONDS(((uint32)time_temp % 60));
```

Figure 8-13. Code Example for Initializing the Real-Time Clock

Chapter 9
Power Management

9.1 Introduction

This chapter explains the low-power operation of the MCF52235.

9.1.1 Features

The following features support low-power operation.

- Four modes of operation: run, wait, doze, and stop
- Ability to shut down most peripherals independently
- Ability to shut down the external CLKOUT pin

9.2 Memory Map/Register Definition

The power management programming model consists of registers from the SCM and CCM memory space, as shown in Table 9-1.

Table 9-1. Power Management Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x11_0004	Chip Configuration Register (CCR)[2]	16	R	0x0001	12.3.3.1/12-3
0x11_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	9.2.4.1/9-8
0x00_000C	Peripheral Power Management Register High (PPMRH)	32	R/W	0x00000000	9.2.1/9-2
0x00_0010	Core Reset Status Register (CRSR)[3]	8	R/W		10.5.2/10-4
0x00_0011	Core Watchdog Control Register (CWCR)[3]	8	R/W	0x00	10.5.1/10-3
0x00_0012	Low-Power Interrupt Control Register (LPICR)	8	R/W	0x00	9.2.2/9-5
0x00_0013	Core Watchdog Service Register (CWSR)[3]	8	R/W		13.5.5/13-9
0x00_0018	Peripheral Power Management Register Low (PPMRL)	32	R/W	0x00000001	9.2.1.1/9-4
0x00_0021	Peripheral Power Management Set Register (PPMRS)	8	W	0x00	9.2.3/9-7
0x00_0022	Peripheral Power Management Clear Register (PPMRC)	32	R/W	0x00	9.2.4/9-8

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 The CCR is described in the Chip Configuration Module. It is shown here only to warn against accidental writes to this register when accessing the LPCR.

3 The CRSR, CWCR, and CWSR are described in the System Control Module. They are shown here only to warn against accidental writes to these registers when accessing the LPICR.

9.2.1 Peripheral Power Management Registers (PPMRH, PPMRL)

The PPMRH and PPMRL registers provide a bit map for controlling the generation of the module clocks for each decoded address space associated with the IPS controller. The PPMR*x* provides a unique control bit for each of these address spaces that defines whether the module clock for the given space is enabled or disabled.

NOTE

It is software's responsibility to appropriately disable module clocks using the PPMR*x* only when a module is completely unused or quiescent.

Because the operation of the IPS controller and the system control module (SCM) are fundamental to the operation of the system, the clocks for these three modules cannot be disabled.

The individual bits of the PPMR*x* can be modified using a read-modify-write to this register directly or indirectly through writes to the PPMRS and PPMRC registers to set/clear individual bits.

See Figure 9-1 and Table 9-2 for the PPMRH definition.

IPSBAR Offset: 0x00_000C (PPMRH) — Access: read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	CDRNGA	CDEPHY	CDCFM	CDFCAN	CDPWM	CDGPT
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	CDADC	0	0	CDPIT1	CDPIT0	0	CDEPORT	CDPORTS
W								
Reset	0	0	0	0	0	0	0	0

Figure 9-1. Peripheral Power Management Register High (PPMRH)

Table 9-2. PPMRH Field Descriptions

Field	Description
31–14	Reserved, should be cleared.
13 CDRNGA	Disable clock to the RNGA (Random Number Generator Accelerator Module) 0 RNGA module clock is enabled 1 RNGA module clock is disabled
12 CDEPHY	Disable clock to the EPHY (Ethernet PHY Module)) 0 EPHY module clock is enabled 1 EPHY module clock is disabled
11 CDCFM	Disable clock to the CFM (Common Flash Module) 0 CFM module clock is enabled 1 CFM module clock is disabled
10 CDFCAN	Disable clock to the FlexCAN module. 0 FlexCAN module clock is enabled 1 FlexCAN module clock is disabled
9 CDPWM	Disable clock to the PWM module. 0 PWM module clock is enabled 1 PWM module clock is disabled
8 CDGPT	Disable clock to the 16 bit general purpose timer module (GPT). 0 GPT module clock is enabled 1 GPT module clock is disabled
7 CDADC	Disable clock to the ADC module. 0 ADC module clock is enabled 1 ADC module clock is disabled
6–5	Reserved, should be cleared.
4 CDPIT1	Disable clock to the PIT1 module. 0 PIT0 module clock is enabled 1 PIT1 module clock is disabled
3 CDPIT0	Disable clock to the PIT0 module. 0 PIT0 module clock is enabled 1 PIT0 module clock is disabled
2	Reserved, should be cleared.
1 CDEPORT	Disable clock to both EPORT modules. 0 EPORT module clock is enabled 1 EPORT module clock is disabled
0 CDPORTS	Disable clock to the Ports module. 0 Ports module clock is enabled 1 Ports module clock is disabled

9.2.1.1 Peripheral Power Management Register Low (PPMRL)

IPSBAR Offset: 0x00_0018 (PPMRL) Access: read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	CDFEC0	0	0	CDINTC1	CDINTC0	CDTMR3
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	CDTMR2	CDTMR1	CDTMR0	CDRTC	0	CDQSPI	CDI2C	0
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	CDUART2	CDUART1	CDUART0	CDDMA	0	0	CDG	0
W								
Reset	0	0	0	0	1	0	0	0

Figure 9-2. Peripheral Power Management Register Low (PPMRL)

Table 9-3. PPMRL Field Descriptions

Field	Description
31–22	Reserved, should be cleared.
21 CDFEC0	Disable clock to the FEC (Fast Ethernet Controller) module. 0 FEC module clock is enabled 1 FEC module clock is disabled
20–19	Reserved, should be cleared.
18 CDINTC0	Disable clock to the INTC1 module. 0 INTC1 module clock is enabled 1 INTC1 module clock is disabled
17 CDINTC0	Disable clock to the INTC0 module. 0 INTC0 module clock is enabled 1 INTC0 module clock is disabled
16 CDTMR3	Disable clock to the DTIM3 module. 0 TMR3 module clock is enabled 1 TMR3 module clock is disabled
15 CDTMR2	Disable clock to the DTIM2 module. 0 TMR2 module clock is enabled 1 TMR2 module clock is disabled

Table 9-3. PPMRL Field Descriptions (continued)

Field	Description
14 CDTMR1	Disable clock to the DTIM1 module. 0 TMR1 module clock is enabled 1 TMR1 module clock is disabled
13 CDTMR0	Disable clock to the DTIM0 module. 0 TMR0 module clock is enabled 1 TMR0 module clock is disabled
12 CDRTC	Disable clock to the RTC (Real-Time Clock) module. 0 RTC module clock is enabled 1 RTC module clock is disabled
11	Reserved, should be cleared.
10 CDQSPI	Disable clock to the QSPI module. 0 QSPI module clock is enabled 1 QSPI module clock is disabled
9 CDI2C	Disable clock to the I2C module. 0 I2C module clock is enabled 1 I2C module clock is disabled
8 —	Reserved, should be cleared.
7 CDUART2	Disable clock to the UART2 module. 0 UART1 module clock is enabled 1 UART2 module clock is disabled
6 CDUART1	Disable clock to the UART1 module. 0 UART1 module clock is enabled 1 UART1 module clock is disabled
5 CDUART0	Disable clock to the UART0 module. 0 UART0 module clock is enabled 1 UART0 module clock is disabled
4 CDDMA	Disable clock to the DMA module. 0 DMA module clock is enabled 1 DMA module clock is disabled
3–2	Reserved, should be cleared.
1 CDG	Disable clock to the Global off-platform modules. 0 Global off-platform module clocks are enabled 1 Global off-platform module clocks are disabled
0	Reserved, should be cleared.

9.2.2 Low-Power Interrupt Control Register (LPICR)

Implementation of low-power stop mode and exit from a low-power mode via an interrupt require communication between the CPU and logic associated with the interrupt controller. The LPICR is an 8-bit register that enables entry into low-power stop mode, and includes the setting of the interrupt level needed to exit a low-power mode.

NOTE

The setting of the low-power mode select (LPMD) field in the power management module's low-power control register (LPCR) determines which low-power mode the device enters when a STOP instruction is issued.

If this field is set to enter stop mode, then the ENBSTOP bit in the LPICR must also be set.

The following is the sequence of operations needed to enable this functionality:

1. The LPICR is programmed, setting the ENBSTOP bit (if stop mode is the desired low-power mode) and loading the appropriate interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. After the processor has stopped execution, it asserts a specific Processor Status (PST) encoding. Issuing the STOP instruction when the LPICR[ENBSTOP] bit is set causes the SCM to enter stop mode.
3. The entry into a low-power mode is processed by the low-power mode control logic, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate an interrupt request with a priority level greater than the value programmed in LPICR[XLPM_IPL[2:0]].

NOTE

Only a fixed (external) interrupt can bring a device out of stop mode. To exit from other low-power modes, such as doze or wait, fixed or programmable interrupts may be used; however, the module generating the interrupt must be enabled in that particular low-power mode.

5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

IPSBAR Offset: 0x00_0012 (LPICR) Access: read/write

	7	6	5	4	3	2	1	0
R	ENBSTOP	XLPM_IPL[2:0]			0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-3. Low-Power Interrupt Control Register (LPICR)

Table 9-4. LPICR Field Description

Field	Description

Table 9-4. LPICR Field Description (continued)

7 ENBSTOP	Enable low-power stop mode. 0 Low-power stop mode disabled 1 Low-power stop mode enabled. After the core is stopped and the signal to enter stop mode is asserted, processor clocks can be disabled.
6–4 XLPM_IPL [2:0]	Exit low-power mode interrupt priority level. This field defines the interrupt priority level needed to exit the low-power mode.Refer to Table 9-5.
3–0 —	Reserved, should be cleared.

Table 9-5. XLPM_IPL Settings

XLPM_IPL[2:0]	Interrupts Level Needed to Exit Low-Power Mode
000	Any interrupt request exits low-power mode
001	Interrupt request levels 2–7 exit low-power mode
010	Interrupt request levels 3–7 exit low-power mode
011	Interrupt request levels 4–7 exit low-power mode
100	Interrupt request levels 5–7 exit low-power mode
101	Interrupt request levels 6–7 exit low-power mode
11x	Interrupt request level 7 exits low-power mode

9.2.3 Peripheral Power Management Set Register (PPMRS)

The PPMRS register provides a simple memory-mapped mechanism to set a given bit in the PPMR*x* registers to disable the clock for a given IPS module without the need to perform a read-modify-write on the PPMR. The data value on a register write causes the corresponding bit in the PPMR*x* register to be set. A data value of 64 to 127 provides a global set function, forcing the entire contents of the PPMR*x* to be set, disabling all IPS module clocks. Reads of this register return all zeroes. See Figure 9-4 and Table 9-6 for the PPMRS definition.

IPSBAR Offset: 0x00_0021 (PPMRS) Access: write-only

	7	6	5	4	3	2	1	0
R	0							
W		PPMRS						
Reset:	0	0	0	0	0	0	0	0

Figure 9-4. Peripheral Power Management Set Register (PPMRS)

Table 9-6. PPMRS Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRS	Set Module Clock Disable 0–63 Set corresponding bit in PPMR*x*, disabling the module clock 64–127 Set all bits in PPMR*x*, disabling all the module clocks

9.2.4 Peripheral Power Management Clear Register (PPMRC)

The PPMRC register provides a simple memory-mapped mechanism to clear a given bit in the PPMR*x* registers to *enable the clock* for a given IPS module without the need to perform a read-modify-write on the PPMR*x*. The data value on a register write causes the corresponding bit in the PPMR*x* register to be cleared. A data value of 64 to 127 provides a global clear function, forcing the entire contents of the PPMR*x* to be zeroed, enabling all IPS module clocks. In the event on simultaneous writes of the PPMRS and PPMRC, the write to the PPMRC takes priority. Reads of this register return all zeroes. See Figure 9-5 and Table 9-7 for the PPMRC definition.

IPSBAR Offset: 0x00_0022 (PPMRC) Access: write-only

	7	6	5	4	3	2	1	0
R	0							
W		PPMRC						
Reset:	0	0	0	0	0	0	0	0

Figure 9-5. Peripheral Power Management Clear Register (PPMRC)

Table 9-7. PPMRC Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRC	Clear Module Clock Disable 0–63 Clear corresponding bit in PPMR*x*, enabling the module clock 64–127 Clear all bits in PPMR*x*, enabling all the module clocks

9.2.4.1 Low-Power Control Register (LPCR)

The LPCR controls chip operation and module operation during low-power modes. The low-power control register (LPCR) specifies the low-power mode entered when the STOP instruction is issued, and controls clock activity in this low-power mode.

IPSBAR Offset: 0x11_0007 (LPCR) Access: read/write

	7	6	5	4	3	2	1	0
R	LPMD		0	STPMD		0	LVDSE	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-6. Low-Power Control Register (LPCR)

Table 9-8. LPCR Field Descriptions

Field	Description
7–6 LPMD	Low-power mode select. Used to select the low-power mode the chip enters after the ColdFire CPU executes the STOP instruction. These bits must be written prior to instruction execution for them to take effect. The LPMD[1:0] bits are readable and writable in all modes. The four different power modes that can be configured with the LPMD bit field are illustrated below: **LPMD[1:0]** / **Mode** 11 STOP 10 WAIT 01 DOZE 00 RUN **Note:** If LPCR[LPMD] is cleared, then the device stops executing code upon issue of a STOP instruction. However, no clocks is disabled.
5	Reserved, should be cleared.
4–3 STPMD	PLL/CLKOUT stop mode. Controls PLL and CLKOUT operation in stop mode as shown in Table 9-10
2	Reserved, should be cleared.
1 LVDSE	LDV standby enable. Controls whether the PMM enters VREG Standby Mode (LVD disabled) or VREG Pseudo-Standby (LVD enabled) mode when the PMM receives a power down request. This bit has no effect if the RCR[LVDE] bit is a logic 0. 1 VREG Pseudo-Standby mode (LVD enabled on power down request). 0 VREG Standby mode (LVD disabled on power down request).
0	Reserved, should be cleared.

9.3 IPS Bus Timeout Monitor

The IPS controller implements a bus timeout monitor to ensure that every IPS bus cycle is properly terminated within a programmed period of time. The monitor continually checks for termination of each IPS bus cycle and completes the cycle if there is no response when the programmed monitor cycle count is reached. The error termination is propagated onto the system bus and eventually back to the ColdFire Core.

The monitor can be programmed from 8–1024 system bus cycles under control of the IPS Bus Monitor Timeout Register (IPSBMT). The timeout value must be selected so that it is larger than the response time of the slowest IPS peripheral device. The bus timeout monitor begins counting on the initial assertion of any IPS module enable and continues to count until the bus cycle is terminated via the negation of **ips_xfr_wait**. If the programmed timeout value is reached before a termination, the bus monitor completes the cycle with an error termination. At reset, the IPSBMT is enabled with a maximum timeout value. See Figure 9-7 and Table 9-9 for the IPSBMT definition.

0x00_0023 (IPSBMT) Access: read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 9-7. IPS Bus Timeout Monitor (IPSBMT) Register

Table 9-9. IPSBMT Field Description

Field	Description
15–4	Reserved, should be cleared.
3 BME	Bus Timeout Monitor Enable 0 The bus timeout monitor is disabled. 1 The bus timeout monitor is enabled.
2–0 BMT[2:0]	Bus Monitor Timeout. This field selects the timeout period (measured in system bus clock cycles) for the bus monitor. 000 1024 cycles 001 512 cycles 010 256 cycles 011 128 cycles 100 64 cycles 101 32 cycles 110 16 cycles 111 8 cycles

9.4 Functional Description

Table 9-10. PLL/CLKOUT Stop Mode Operation

STPMD[1:0]	Operation During Stop Mode				
	System Clocks	CLKOUT	PLL	OSC	PMM
00	Disabled	Enabled	Enabled	Enabled	Enabled
01	Disabled	Disabled	Enabled	Enabled	Enabled
10	Disabled	Disabled	Disabled	Enabled	Enabled
11	Disabled	Disabled	Disabled	Disabled	Low-Power Option

The functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes are discussed in this section.

9.4.1 Low-Power Modes

The system enters a low-power mode by executing a STOP instruction. Which mode the device actually enters (stop, wait, or doze) depends on what is programmed in LPCR[LPMD]. Entry into any of these modes idles the CPU with no cycles active, powers down the system and stops all internal clocks appropriately. During stop mode, the system clock is stopped low.

For entry into stop mode, the LPICR[ENBSTOP] bit must be set before a STOP instruction is issued.

A wakeup event is required to exit a low-power mode and return to run mode. Wakeup events consist of any of these conditions:

- Any type of reset
- Any valid, enabled interrupt request

Exiting from low power mode via an interrupt request requires:

- An interrupt request whose priority is higher than the value programmed in the XLPM_IPL field of the LPICR.
- An interrupt request whose priority higher than the value programmed in the interrupt priority mask (I) field of the core’s status register.
- An interrupt request from a source which is not masked in the interrupt controller’s interrupt mask register.
- An interrupt request which has been enabled at the module of the interrupt’s origin.

9.4.1.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

9.4.1.2 Wait Mode

Wait mode is intended to be used to stop only the CPU and memory clocks until a wakeup event is detected. In this mode, peripherals may be programmed to continue operating and can generate interrupts, which cause the CPU to exit from wait mode.

9.4.1.3 Doze Mode

Doze mode affects the CPU in the same manner as wait mode, except that each peripheral defines individual operational characteristics in doze mode. Peripherals which continue to run and have the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Peripherals which are stopped restart operation on exit from doze mode as defined for each peripheral.

9.4.1.4 Stop Mode

Stop mode affects the CPU in the same manner as the wait and doze modes, except that all clocks to the system are stopped and the peripherals cease operation.

Stop mode must be entered in a controlled manner to ensure that any current operation is properly terminated. When exiting stop mode, most peripherals retain their pre-stop status and resume operation.

The following subsections specify the operation of each module while in and when exiting low-power modes.

9.4.1.5 Peripheral Shut Down

Most peripherals may be disabled by software to cease internal clock generation and remain in a static state. Each peripheral has its own specific disabling sequence (refer to each peripheral description for further details). A peripheral may be disabled at any time and remain disabled during any low-power mode of operation.

9.4.2 Peripheral Behavior in Low-Power Modes

9.4.2.1 ColdFire Core

The ColdFire core is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

9.4.2.2 Static Random-Access Memory (SRAM)

SRAM is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

9.4.2.3 Flash

The flash module is in a low-power state if not being accessed. No recovery time is required after exit from any low-power mode.

9.4.2.4 System Control Module (SCM)

The SCM's core watchdog timer can bring the device out of all low-power modes except stop mode. In stop mode, all clocks stop, and the core watchdog does not operate.

When enabled, the core watchdog can bring the device out of wait and doze modes via a core watchdog interrupt. This system setup must meet the conditions specified in Section 9.4.1, "Low-Power Modes" for the core watchdog interrupt to bring the part out of wait and doze modes.

9.4.2.5 DMA Controller (DMA0–DMA3)

In wait and doze modes, the DMA controller is capable of bringing the device out of a low-power mode by generating an interrupt upon completion of a transfer or upon an error condition. The completion of transfer interrupt is generated when DMA interrupts are enabled by the setting of the DCR[INT] bit, and an interrupt is generated when the DSR[DONE] bit is set. The interrupt upon error condition is generated when the DCR[INT] bit is set, and an interrupt is generated when the CE, BES, or BED bit in the DSR becomes set.

The DMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

9.4.2.6 UART Modules (UART0, UART1, and UART2)

In wait and doze modes, the UART may generate an interrupt to exit the low-power modes.

- Clearing the transmit enable bit (TE) or the receiver enable bit (RE) disables UART functions.
- The UARTs are unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the UARTs stop immediately and freeze their operation, register values, state machines, and external pins. During this mode, the UART clocks are shut down. Coming out of stop mode returns the UARTs to operation from the state prior to the low-power mode entry.

9.4.2.7 I^2C Module

When the I^2C Module is enabled by the setting of the I2CR[IEN] bit and when the device is not in stop mode, the I^2C module is operable and may generate an interrupt to bring the device out of a low-power mode. For an interrupt to occur, the I2CR[IIE] bit must be set to enable interrupts, and the setting of the I2SR[IIF] generates the interrupt signal to the CPU and interrupt controller. The setting of I2SR[IIF] signifies the completion of one byte transfer or the reception of a calling address matching its own specified address when in slave receive mode.

In stop mode, the I^2C Module stops immediately and freezes operation, register values, and external pins. Upon exiting stop mode, the I^2C resumes operation unless stop mode was exited by reset.

9.4.2.8 Queued Serial Peripheral Interface (QSPI)

In wait and doze modes, the queued serial peripheral interface (QSPI) may generate an interrupt to exit the low-power modes.

- Clearing the QSPI enable bit (SPE) disables the QSPI function.
- The QSPI is unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the QSPI stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the QSPI clocks are shut down. Coming out of stop mode returns the QSPI to operation from the state prior to the low-power mode entry.

9.4.2.9 DMA Timers (DTIM0–DTIM3)

In wait and doze modes, the DMA timers may generate an interrupt to exit a low-power mode. This interrupt can be generated when the DMA Timer is in input capture mode or reference compare mode.

In input capture mode, where the capture enable (CE) field of the timer mode register (DTMR) has a non-zero value and the DMA enable (DMAEN) bit of the DMA timer extended mode register (DTXMR) is cleared, an interrupt is issued upon a captured input. In reference compare mode, where the output reference request interrupt enable (ORRI) bit of DTMR is set and the DTXMR[DMAEN] bit is cleared, an interrupt is issued when the timer counter reaches the reference value.

DMA timer operation is disabled in stop mode, but the DMA timer is unaffected by the wait or doze modes and may generate an interrupt to exit these modes. Upon exiting stop mode, the timer resumes operation unless stop mode was exited by reset.

9.4.2.10 Interrupt Controllers (INTC0, INTC1)

The interrupt controller is not affected by any of the low-power modes. All logic between the input sources and generating the interrupt to the processor is combinational to allow the ability to wake up the CPU processor during low-power stop mode when all system clocks are stopped.

An interrupt request causes the CPU to exit a low-power mode only if that interrupt's priority level is at or above the level programmed in the interrupt priority mask field of the CPU's status register (SR). The interrupt must also be enabled in the interrupt controller's interrupt mask register as well as at the module from which the interrupt request would originate.

9.4.2.11 Fast Ethernet Controller (FEC)

In wait and doze modes, the FEC may generate an interrupt to exit the low-power modes.

- Clearing the ECNTRL[ETHER_EN] bit disables the FEC function.
- The FEC is unaffected by wait mode and may generate an interrupt to exit this mode.

In stop mode, the FEC stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the FEC clocks are shut down. Coming out of stop mode returns the FEC to operation from the state prior to the low-power mode entry.

9.4.2.12 I/O Ports

The I/O ports are unaffected by entry into a low-power mode. These pins may impact low-power current draw if they are configured as outputs and are sourcing current to an external load. If low-power mode is exited by a reset, the state of the I/O pins reverts to their default direction settings.

9.4.2.13 Reset Controller

A power-on reset (POR) always causes a chip reset and exit from any low-power mode.

In wait and doze modes, asserting the external $\overline{\text{RESET}}$ pin for at least four clocks causes an external reset that resets the chip and exit any low-power modes.

In stop mode, the $\overline{\text{RESET}}$ pin synchronization is disabled and asserting the external $\overline{\text{RESET}}$ pin asynchronously generates an internal reset and exit any low-power modes. Registers lose current values and must be reconfigured from reset state if needed.

If the phase lock loop (PLL) in the clock module is active and if the appropriate (LOCRE, LOLRE) bits in the synthesizer control register are set, any loss-of-clock or loss-of-lock resets the chip and exit any low-power modes.

The watchdog timer in the SCM is only able to request an interrupt, so a reset request must be performed in software.

When the CPU is inactive, a software reset cannot be generated to exit any low-power mode.

9.4.2.14 Chip Configuration Module

The Chip Configuration Module is unaffected by entry into a low-power mode. If low-power mode is exited by a reset, chip configuration may be executed if configured to do so.

9.4.2.15 Clock Module

In wait and doze modes, the clocks to the CPU, flash, and SRAM are stopped and the system clocks to the peripherals are enabled. Each module may disable the module clocks locally at the module level. In stop mode, all clocks to the system are stopped.

During stop mode, the PLL continues to run. The external CLKOUT signal may be enabled or disabled when the device enters stop mode, depending on the LPCR[STPMD] bit settings.The external CLKOUT output pin may be disabled to lower power consumption via the SYNCR[DISCLK] bit. The external CLKOUT pin function is enabled by default at reset.

9.4.2.16 Edge Port

In wait and doze modes, the edge port continues to operate normally and may be configured to generate interrupts (an edge transition or low level on an external pin) to exit the low-power modes.

In stop mode, there is no system clock available to perform the edge detect function. Thus, only the level detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit the stop mode.

9.4.2.17 Programmable Interrupt Timers (PIT0–PIT1)

In stop mode (or in doze mode, if so programmed), the programmable interrupt timer (PIT) ceases operation, and freezes at the current value. When exiting these modes, the PIT resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the PIT may generate an interrupt to exit the low-power modes.

9.4.2.18 FlexCAN

When enabled, the FlexCAN module is capable of generating interrupts and bringing the device out of a low-power mode. The module has 35 interrupt sources (32 sources due to message buffers and 3 sources due to Bus-off, Error and Wake-up).

When in stop mode, a recessive to dominant transition on the CAN bus causes the WAKE-INT bit in the error & status register to be set. This event can cause a CPU interrupt if the WAKE-MASK bit in module configuration register (MCR) is set.

When setting stop mode in the FlexCAN (by setting the MCR[STOP] bit), the FlexCAN checks for the CAN bus to be idle or waits for the third bit of intermission and checks to see if it is recessive. When this condition exists, the FlexCAN waits for all internal activity other than in the CAN bus interface to complete and then the following occurs:

- The FlexCAN shuts down its clocks, stopping most of the internal circuits, to achieve maximum possible power saving.
- The internal bus interface logic continues operation, enabling CPU to access the MCR register.
- The FlexCAN ignores its Rx input pin, and drives its Tx pins as recessive.
- FlexCAN loses synchronization with the CAN bus, and STOP_ACK and NOT_RDY bits in MCR register are set.

Exiting stop mode is done in one of the following ways:

- Reset the FlexCAN (by hard reset or by asserting the SOFT_RST bit in MCR).
- Clearing the STOP bit in the MCR.
- Self-wake mechanism. If the SELF-WAKE bit in the MCR is set at the time the FlexCAN enters stop mode, then upon detection of recessive to dominant transition on the CAN bus, the FlexCAN resets the STOP bit in the MCR and resumes its clocks.

Recommendations for, and features of, FlexCAN's stop mode operation are as follows:

- Upon stop/self-wake mode entry, the FlexCAN tries to receive the frame that caused it to wake; that is, it assumes that the dominant bit detected is a start-of-frame bit. It does not arbitrate for the CAN bus then.
- Before asserting stop Mode, the CPU should disable all interrupts in the FlexCAN, otherwise it may be interrupted while in stop mode upon a non-wake-up condition. If desired, the WAKE-MASK bit should be set to enable the WAKE-INT.
- If stop mode is asserted while the FlexCAN is BUSOFF (see error and status register), then the FlexCAN enters stop mode and stops counting the synchronization sequence; it continues this count after stop mode is exited.
- The correct flow to enter stop mode with SELF-WAKE:
  - assert SELF-WAKE at the same time as STOP.
  - wait for STOP_ACK bit to be set.
- The correct flow to negate STOP with SELF-WAKE:

  - negate SELF-WAKE at the same time as STOP.
  - wait for STOP_ACK negation.
- SELF-WAKE should be set only when the MCR[STOP] bit is negated and the FlexCAN is ready; that is, the NOT_RDY bit in the MCR is negated.
- If STOP and SELF_WAKE are set and if a recessive to dominant edge immediately follows on the CAN bus, the STOP_ACK bit in the MCR may never be set, and the STOP bit in the MCR is reset.
- If the user does not want to have old frames sent when the FlexCAN is awakened (STOP with Self-Wake), the user should disable all Tx sources, including remote-response, before stop mode entry.
- If halt mode is active at the time the STOP bit is set, then the FlexCAN assumes that halt mode should be exited; hence it tries to synchronize to the CAN bus (11 consecutive recessive bits), and only then does it search for the correct conditions to stop.
- Trying to stop the FlexCAN immediately after reset is allowed only after basic initialization has been performed.

If stop with self-wake is activated, and the FlexCAN operates with single system clock per time-quanta, then there are extreme cases in which FlexCAN's wake-up upon recessive to dominant edge may not conform to the standard CAN protocol, in the sense that the FlexCAN synchronization is shifted one time quanta from the required timing. This shift lasts until the next recessive to dominant edge, which re-synchronizes the FlexCAN back to conform to the protocol. The same holds for auto-power save mode upon wake-up by recessive to dominant edge.

The auto-power save mode in the FlexCAN is intended to enable NORMAL operation with optimized power saving. Upon setting the AUTO POWER SAVE bit in the MCR register, the FlexCAN looks for a set of conditions in which there is no need for clocks to run. If all these conditions are met, then the FlexCAN stops its clocks, thus saving power. While its clocks are stopped, if any of the conditions below is not met, the FlexCAN resumes its clocks. It then continues to monitor the conditions and stops/resumes its clocks appropriately.

The following are conditions for the automatic shut-off of FlexCAN clocks:

- No Rx/Tx frame in progress.
- No moving of Rx/Tx frames between SMB and MB and no Tx frame is pending for transmission in any MB.
- No host access to the FlexCAN module.
- The FlexCAN is neither in halt mode (MCR bit 8), in stop mode (MCT bit 15), nor in BUSOFF.

9.4.2.19 PWM Module

The PWM module is user programmable as to how it behaves when the device enters wait mode (PWMCTL[PSWAI]) and doze mode (PWMCTL[PFRZ]). If either of these bits are set, the PWM input clock to the prescalar is disabled during the respective low power mode.

In stop mode the input clock is disabled and PWM generation is halted.

9.4.2.20 BDM

Entering halt mode via the BDM port (by asserting the external $\overline{\text{BKPT}}$ pin) causes the CPU to exit any low-power mode.

9.4.2.21 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and is not affected by the system clock. The JTAG cannot generate an event to cause the CPU to exit any low-power mode. Toggling TCLK during any low-power mode increases the system current consumption.

9.4.3 Summary of Peripheral State During Low-Power Modes

The functionality of each of the peripherals and CPU during the various low-power modes is summarized in Table 9-11. The status of each peripheral during a given mode refers to the condition the peripheral automatically assumes when the STOP instruction is executed and the LPCR[LPMD] field is set for the particular low-power mode. Individual peripherals may be disabled by programming its dedicated control bits. The wakeup capability field refers to the ability of an interrupt or reset by that peripheral to force the CPU into run mode.

Table 9-11. CPU and Peripherals in Low-Power Modes

Module	Peripheral Status[1] / Wakeup Capability					
	Wait Mode		Doze Mode		Stop Mode	
CPU	Stopped	No	Stopped	No	Stopped	No
SRAM	Stopped	No	Stopped	No	Stopped	No
Flash	Stopped	No	Stopped	No	Stopped	No
System Control Module	Enabled	Reset	Enabled	Reset	Stopped	No
Random Number Generator Accelerator	Enabled	No	Stopped	No	Stopped	No
DMA Controller	Enabled	Yes	Enabled	Yes	Stopped	No
UART0, UART1 and UART2	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
I^2C Module	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
QSPI	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
Interrupt Controller	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
I/O Ports	Enabled	No	Enabled	No	Enabled	No
Reset Controller	Enabled	Reset	Enabled	Reset	Enabled	Reset
Chip Configuration Module	Enabled	No	Enabled	No	Stopped	No
Power Management	Enabled	No	Enabled	No	Stopped	No
Clock Module	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Real-Time Clock Module	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt

Table 9-11. CPU and Peripherals in Low-Power Modes (continued)

Module	Peripheral Status[1] / Wakeup Capability					
	Wait Mode		Doze Mode		Stop Mode	
Edge port	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	No
ADC	Enabled	Interrupt	Program	Interrupt	Stopped	No
General Purpose Timer	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
FlexCAN	Enabled	Interrupt	Enabled	Interrupt	Stopped	No
PWM	Program	No	Program	No	Stopped	No
BDM	Enabled	Yes[2]	Enabled	Yes[2]	Enabled	Yes[2]
JTAG	Enabled	No	Enabled	No	Enabled	No

1 Program Indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

2 The BDM logic is clocked by a separate TCLK clock. Entering halt mode via the BDM port exits any low-power mode. Upon exit from halt mode, the previous low-power mode is re-entered and changes made in halt mode remains in effect.

Chapter 10
Reset Controller Module

10.1 Introduction

The reset controller is provided to determine the cause of reset, assert the appropriate reset signals to the system, and keep a history of what caused the reset. The low voltage detection module, which generates low-voltage detect (LVD) interrupts and resets, is implemented within the reset controller module.

10.2 Features

Module features include the following:

- Seven sources of reset:
  - External reset input
  - Power-on reset (POR)
  - Phase locked-loop (PLL) loss of lock
  - PLL loss of clock
  - Software
  - Low-voltage detector (LVD)
  - JTAG CLAMP, HIGHZ and EXTEST instructions
- Software-assertable $\overline{\text{RSTO}}$ pin independent of chip reset state
- Software-readable status flags indicating the cause of the last reset
- LVD control and status bits for setup and use of LVD reset or interrupt

10.3 Block Diagram

Figure 10-1 illustrates the reset controller and is explained in the following sections.

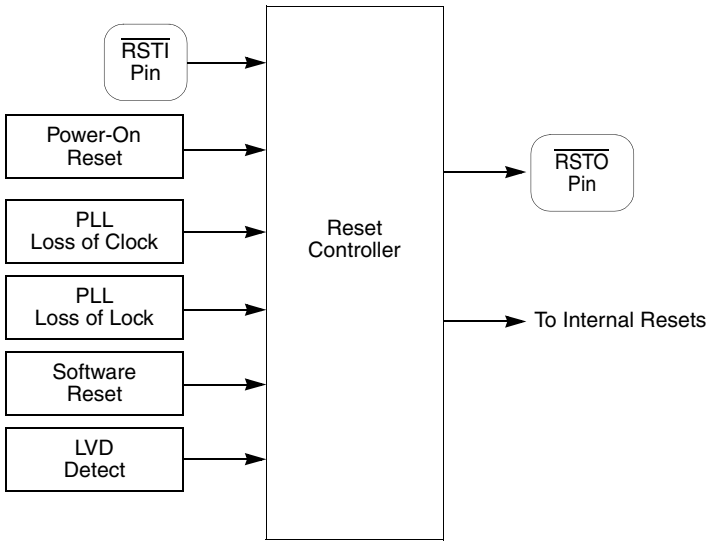


Figure 10-1. Reset Controller Block Diagram

10.4 Signals

Table 10-1 provides a summary of the reset controller signal properties. The signals are described in the following sections.

Table 10-1. Reset Controller Signal Properties

Name	Direction	Input Hysteresis	Input Synchronization
$\overline{\text{RSTI}}$	I	Yes	Yes[1]
$\overline{\text{RSTO}}$	O	—	—

1 $\overline{\text{RSTI}}$ is always synchronized except when in low-power stop mode.

10.4.1 $\overline{\text{RSTI}}$

Asserting the external $\overline{\text{RSTI}}$ for at least four rising CLKOUT edges causes the external reset request to be recognized and latched.

10.4.2 $\overline{\text{RSTO}}$

This active-low output signal is driven low when the internal reset controller module resets the chip. When $\overline{\text{RSTO}}$ is active, the user can drive override options on the data bus.

10.5 Memory Map and Registers

The reset controller programming model consists of these registers:

- Reset control register (RCR)—selects reset controller functions
- Reset status register (RSR)—reflects the state of the last reset source

See Table 10-2 for the memory map and the following paragraphs for a description of the registers.

Table 10-2. Reset Controller Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x11_0000	Reset Control Register (RCR)	8	R/W	0x05	10.5.1/10-3
0x11_0001	Reset Status Register (RSR)	8	R		10.5.2/10-4

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

10.5.1 Reset Control Register (RCR)

The RCR allows software control for requesting a reset, independently asserting the external $\overline{\text{RSTO}}$ pin, and controlling low-voltage detect (LVD) functions.

IPSBAR Offset: 0x11_0000 (RCR) Access: User read/write

	7	6	5	4	3	2	1	0
R	SOFTRST	FRCRSTOUT	0	LVDF	LVDIE	LVDRE	0	LVDE
W								
Reset:	0	0	0	0	0	1	0	1

Figure 10-2. Reset Control Register (RCR)

Table 10-3. RCR Field Descriptions

Field	Description
7 SOFTRST	Allows software to request a reset. The reset caused by setting this bit clears this bit. 1 Software reset request 0 No software reset request
6 FRCRSTOUT	Allows software to assert or negate the external $\overline{\text{RSTO}}$ pin. 1 Assert $\overline{\text{RSTO}}$ pin 0 Negate $\overline{\text{RSTO}}$ pin CAUTION: External logic driving reset configuration data during reset needs to be considered when asserting the $\overline{\text{RSTO}}$ pin when setting FRCRSTOUT.
5 —	Reserved, should be cleared.

Table 10-3. RCR Field Descriptions (continued)

Field	Description
4 LVDF	LVD flag. Indicates the low-voltage detect status if LVDE is set. Write a 1 to clear the LVDF bit. 1 Low voltage has been detected 0 Low voltage has not been detected **Note:** The setting of this flag causes an LVD interrupt if LVDE and LVDIE bits are set and LVDRE is cleared when the supply voltage V_{DD} drops below V_{DD} (minimum). The vector for this interrupt is shared with INT0 of the EPORT module. Interrupt arbitration in the interrupt service routine is necessary if both of these interrupts are enabled. Also, LVDF is not cleared at reset; however, it always initializes to a zero because the part does not come out of reset while in a low-power state (LVDE/LVDRE bits are enabled out of reset).
3 LVDIE	LVD interrupt enable. Controls the LVD interrupt if LVDE is set. This bit has no effect if the LVDE bit is a logic 0. 1 LVD interrupt enabled 0 LVD interrupt disabled
2 LVDRE	LVD reset enable. Controls the LVD reset if LVDE is set. This bit has no effect if the LVDE bit is a logic 0. LVD reset has priority over LVD interrupt, if both are enabled. 1 LVD reset enabled 0 LVD reset disabled
1 —	Reserved, should be cleared.
0 LVDE	Controls whether the LVD is enabled. 1 LVD is enabled 0 LVD is disabled

10.5.2 Reset Status Register (RSR)

The RSR contains a status bit for every reset source. When reset is entered, the cause of the reset condition is latched, along with a value of 0 for the other reset sources that were not pending at the time of the reset condition. These values are then reflected in RSR. One or more status bits may be set at the same time. The cause of any subsequent reset is also recorded in the register, overwriting status from the previous reset condition.

RSR can be read at any time. Writing to RSR has no effect.

IPSBAR Offset: 0x11_0001 (RSR) Access: User read-only

	7	6	5	4	3	2	1	0
R	0	LVD	SOFT	0	POR	EXT	LOC	LOL
W								

Reset: Reset Dependent

Figure 10-3. Reset Status Register (RSR)

Table 10-4. RSR Field Descriptions

Field	Description
7 —	Reserved, should be cleared.
6 LVD	Low voltage detect. Indicates that the last reset state was caused by an LVD reset. 1 Last reset state was caused by an LVD reset 0 Last reset state was not caused by an LVD reset
5 SOFT	Software reset flag. Indicates that the last reset was caused by software. 1 Last reset caused by software 0 Last reset not caused by software
4 —	Reserved, should be cleared.
3 POR	Power-on reset flag. Indicates that the last reset was caused by a power-on reset. 1 Last reset caused by power-on reset 0 Last reset not caused by power-on reset
2 EXT	External reset flag. Indicates that the last reset was caused by an external device asserting the external $\overline{\text{RSTI}}$ pin. 1 Last reset state caused by external reset 0 Last reset not caused by external reset
1 LOC	Loss-of-clock reset flag. Indicates that the last reset state was caused by a PLL loss of clock. 1 Last reset caused by loss of clock 0 Last reset not caused by loss of clock
0 LOL	Loss-of-lock reset flag. Indicates that the last reset state was caused by a PLL loss of lock. 1 Last reset caused by a loss of lock 0 Last reset not caused by loss of lock

10.6 Functional Description

10.6.1 Reset Sources

Table 10-5 defines the sources of reset and the signals driven by the reset controller.

Table 10-5. Reset Source Summary

Source	Type
Power on	Asynchronous
External $\overline{\text{RSTI}}$ pin (not stop mode)	Synchronous
External $\overline{\text{RSTI}}$ pin (during stop mode)	Asynchronous
Loss-of-clock	Asynchronous
Loss-of-lock	Asynchronous
Software	Synchronous
LVD reset	Asynchronous

To protect data integrity, a synchronous reset source is not acted upon by the reset control logic until the end of the current bus cycle. Reset is then asserted on the next rising edge of the system clock after the

cycle is terminated. When the reset control logic must synchronize reset to the end of the bus cycle, the internal bus monitor is automatically enabled regardless of the BME bit state in the chip configuration register (CCR). Then, if the current bus cycle is not terminated normally, the bus monitor terminates the cycle based on the length of time programmed in the BMT field of the CCR.

Internal byte, word, or longword writes are guaranteed to complete without data corruption when a synchronous reset occurs. External writes, including longword writes to 16-bit ports, are also guaranteed to complete.

Asynchronous reset sources usually indicate a catastrophic failure. Therefore, the reset control logic does not wait for the current bus cycle to complete. Reset is asserted immediately to the system.

10.6.1.1 Power-On Reset

At power up, the reset controller asserts $\overline{\text{RSTO}}$. $\overline{\text{RSTO}}$ continues to be asserted until V_{DD} has reached a minimum acceptable level and, if PLL clock mode is selected, until the PLL achieves phase lock. Then after approximately another 512 cycles, $\overline{\text{RSTO}}$ is negated and the part begins operation.

10.6.1.2 External Reset

Asserting the external $\overline{\text{RSTI}}$ for at least four rising CLKOUT edges causes the external reset request to be recognized and latched. The bus monitor is enabled and the current bus cycle is completed. The reset controller asserts $\overline{\text{RSTO}}$ for approximately 512 cycles after $\overline{\text{RSTI}}$ is negated and the PLL has acquired lock. The part then exits reset and begins operation.

In low-power stop mode, the system clocks are stopped. Asserting the external $\overline{\text{RSTI}}$ in stop mode causes an external reset to be recognized.

10.6.1.3 Loss-of-Clock Reset

This reset condition occurs in PLL clock mode when the LOCRE bit in the SYNCR is set and the PLL reference or the PLL itself fails. The reset controller asserts $\overline{\text{RSTO}}$ for approximately 512 cycles after the PLL has acquired lock. The device then exits reset and begins operation.

10.6.1.4 Loss-of-Lock Reset

This reset condition occurs in PLL clock mode when the LOLRE bit in the SYNCR is set and the PLL loses lock. The reset controller asserts $\overline{\text{RSTO}}$ for approximately 512 cycles after the PLL has acquired lock. The device then exits reset and resumes operation.

10.6.1.5 Software Reset

A software reset occurs when the SOFTRST bit is set. If the $\overline{\text{RSTI}}$ is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTO}}$ for approximately 512 cycles. Then the device exits reset and resumes operation.

10.6.1.6 LVD Reset

The LVD reset occurs when the supply input voltage, $V_{DD,}$ drops below V_{LVD} (minimum).

10.6.2 Reset Control Flow

The reset logic control flow is shown in Figure 10-4. In this figure, the control state boxes have been numbered, and these numbers are referred to (within parentheses) in the flow description that follows. All cycle counts given are approximate.

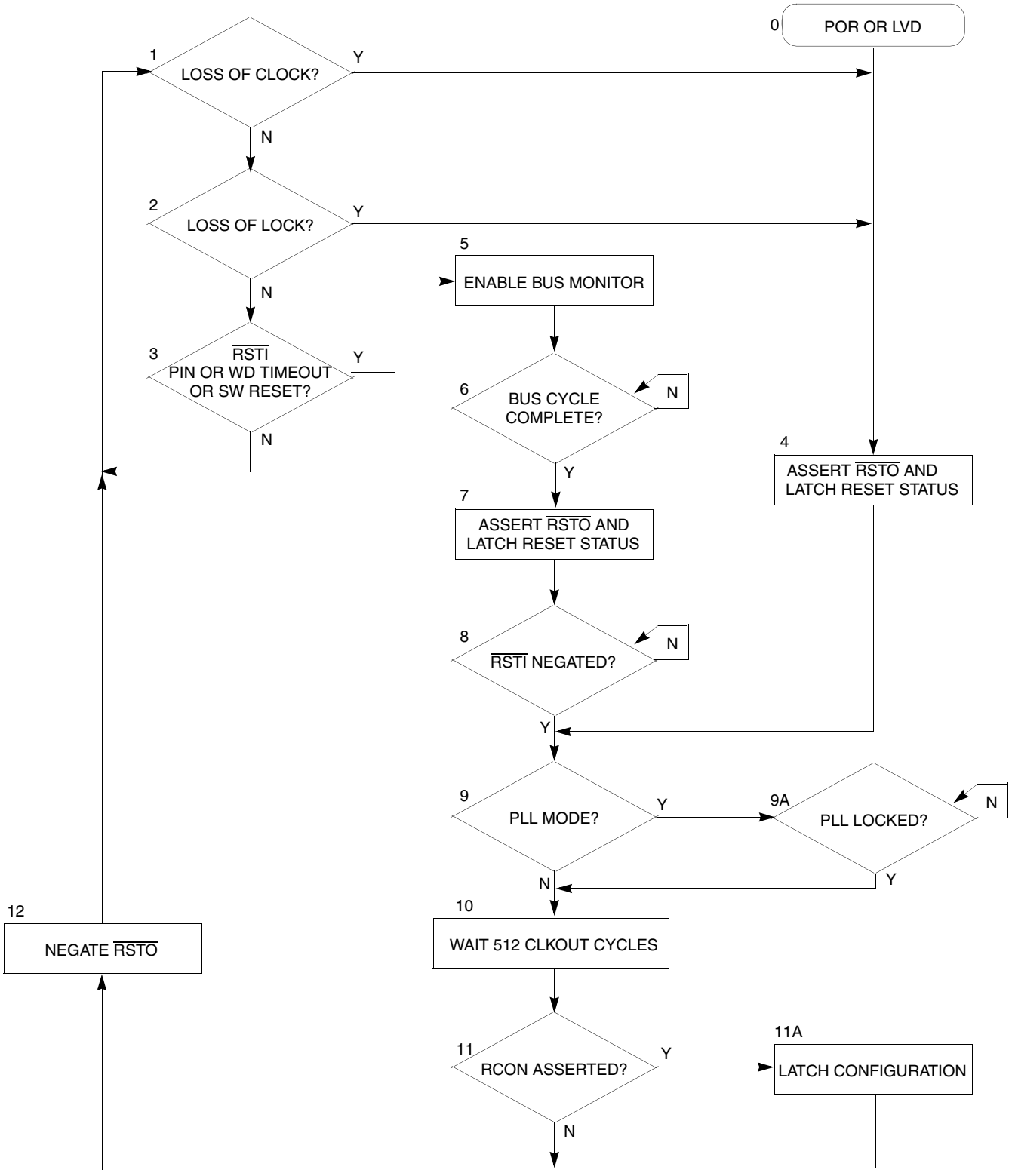


Figure 10-4. Reset Control Flow

10.6.2.1 Synchronous Reset Requests

In this discussion, the references in parentheses refer to the state numbers in Figure 10-4. All cycle counts given are approximate.

If the external RSTI signal is asserted by an external device for at least four rising CLKOUT edges (3) and if software requests a reset, the reset control logic latches the reset request internally and enables the bus monitor (5). When the current bus cycle is completed (6), RSTO is asserted (7). The reset control logic waits until the RSTI signal is negated (8) and for the PLL to attain lock (9, 9A) before waiting 512 CLKOUT cycles (1). The reset control logic may latch the configuration according to the RCON signal level (11, 11A) before negating RSTO (12).

If the external RSTI signal is asserted by an external device for at least four rising CLKOUT edges during the 512 count (10) or during the wait for PLL lock (9A), the reset flow switches to (8) and waits for the RSTI signal to be negated before continuing.

10.6.2.2 Internal Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of clock (1) or loss of lock (2), the reset control logic asserts RSTO (4). The reset control logic waits for the PLL to attain lock (9, 9A) before waiting 512 CLKOUT cycles (1). Then the reset control logic may latch the configuration according to the RCON pin level (11, 11A) before negating RSTO (12).

If loss of lock occurs during the 512 count (10), the reset flow switches to (9A) and waits for the PLL to lock before continuing.

10.6.2.3 Power-On Reset/Low-Voltage Detect Reset

When the reset sequence is initiated by power-on reset (0), the same reset sequence is followed as for the other asynchronous reset sources.

10.6.3 Concurrent Resets

This section describes the concurrent resets. As in the previous discussion, references in parentheses refer to the state numbers in Figure 10-4.

10.6.3.1 Reset Flow

If a power-on reset or low-voltage detect condition is detected during any reset sequence, the reset sequence starts immediately (0).

If the external RSTI pin is asserted for at least four rising CLKOUT edges while waiting for PLL lock or the 512 cycles, the external reset is recognized. Reset processing switches to wait for the external RSTI pin to negate (8).

If a loss-of-clock or loss-of-lock condition is detected while waiting for the current bus cycle to complete (5, 6) for an external reset request, the cycle is terminated. The reset status bits are latched (7) and reset processing waits for the external RSTI pin to negate (8).

If a loss-of-clock or loss-of-lock condition is detected during the 512 cycle wait, the reset sequence continues after a PLL lock (9, 9A).

10.6.3.2 Reset Status Flags

For a POR reset, the POR and LVD bits in the RSR are set, and the SOFT, WDR, EXT, LOC, and LOL bits are cleared even if another type of reset condition is detected during the reset sequence for the POR.

If a loss-of-clock or loss-of-lock condition is detected while waiting for the current bus cycle to complete (5, 6) for an external reset request, the EXT, SOFT, and/or WDR bits along with the LOC and/or LOL bits are set.

If the RSR bits are latched (7) during the EXT, SOFT, and/or WDR reset sequence with no other reset conditions detected, only the EXT, SOFT, and/or WDR bits are set.

If the RSR bits are latched (4) during the internal reset sequence with the $\overline{\text{RSTI}}$ pin not asserted and no SOFT or WDR event, then the LOC and/or LOL bits are the only bits set.

For a LVD reset, the LVD bit in the RSR is set, and the SOFT, WDR, EXT, LOC, and LOL bits are cleared to 0, even if another type of reset condition is detected during the reset sequence for LVD.

Chapter 11
Static RAM (SRAM)

11.1 Introduction

This chapter describes the on-chip static RAM (SRAM) implementation, including general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

11.1.1 Overview

The SRAM module provides a general-purpose memory block that the ColdFire processor can access in a single cycle. The location of the memory block can be specified to any 0-modulo-32K address. The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module is physically connected to the processor's high-speed local bus, it can service processor-initiated accesses or memory-referencing commands from the debug module.

The SRAM is dual-ported to DMAprovide access. The SRAM is partitioned into two physical memory arrays to allow simultaneous access to arrays by the processor core and another bus master. For more information see Chapter 12, “System Control Module (SCM).”

11.1.2 Features

The major features includes:

- One 32 Kbyte SRAM
- Single-cycle access
- Physically located on the processor's high-speed local bus
- Memory location programmable on any 0-modulo-32 Kbyte address
- Byte, word, and longword address capabilities

11.2 Memory Map/Register Description

The SRAM programming model shown in Table 11-1 includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

Table 11-1. SRAM Programming Model

Rc[11:0][1]	Register	Width (bits)	Access	Reset Value	Written w/ MOVEC	Section/Page
Supervisor Access Only Registers						
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	11.2.1/11-2

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 31, "Debug Module."

11.2.1 SRAM Base Address Register (RAMBAR)

The configuration information in the SRAM base-address register (RAMBAR) controls the operation of the SRAM module.

- The RAMBAR holds the SRAM base address. The MOVEC instruction provides write-only access to this register.
- The RAMBAR can be read or written from the debug module.
- All undefined bits in the register are reserved. These bits are ignored during writes to the RAMBAR and return zeroes when read from the debug module.
- A reset clears the RAMBAR's valid bit. This invalidates the processor port to the SRAM (The RAMBAR must be initialized before the core can access the SRAM.) All other bits are unaffected.

NOTE

Do not confuse this RAMBAR with the SCM RAMBAR in Section 12.5.2, "Memory Base Address Register (RAMBAR)." Although similar, this core RAMBAR enables core access to the SRAM memory, while the SCM RAMBAR enables peripheral (e.g., DMA and FEC) access to the SRAM.

The RAMBAR contains several control fields. These fields are shown in Figure 11-1.

Rc[11:0]: 0x0C05 (RAMBAR)

Access: User write-only
Debug read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BA																	0	0	0	PRIU	PRIL	SPV	WP	0	0	C/I	SC	SD	UC	UD	V
W																																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0	0	0	0	0	0	U	0	0	U	U	U	U	U	0

Figure 11-1. SRAM Base Address Register (RAMBAR)

Table 11-2. RAMBAR Field Descriptions

Field	Description
31–15 BA	Base Address. Defines the 0-modulo-32K base address of the SRAM module. By programming this field, the SRAM may be located on any 32-Kbyte boundary (16-Kbyte boundary for the MCF52230 and MCF52231).
14–12	Reserved, must be cleared.

Table 11-2. RAMBAR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>11–10
PRIU
PRIL</td><td>Priority Bit. PRIU determines if /FEC or CPU has priority in the upper 16K bank of memory. PRIL determines if DMA/FEC or CPU has priority in the lower 16K bank of memory. If a bit is set, the CPU has priority. If a bit is cleared, DMA/FEC has priority. Priority is determined according to the following table:
<table>
<tr><th>PRIU,PRIL</th><th>Upper Bank Priority</th><th>Lower Bank Priority</th></tr>
<tr><td>00</td><td>DMA/FEC</td><td>DMA/FEC</td></tr>
<tr><td>01</td><td>DMA/FEC</td><td>CPU</td></tr>
<tr><td>10</td><td>CPU</td><td>DMA/FEC</td></tr>
<tr><td>11</td><td>CPU</td><td>CPU</td></tr>
</table>
Note: The recommended setting (maximum performance) for the priority bits is 00.</td></tr>
<tr><td>9
SPV</td><td>Secondary port valid. Allows access by DMA and FEC.
0 DMA and FEC access to memory is disabled.
1 DMA and FEC access to memory is enabled.
Note: The SPV bit in the second RAMBAR register must also be set to allow dual port access to the SRAM. For more information, see Section 12.5.2, “Memory Base Address Register (RAMBAR).”</td></tr>
<tr><td>8
WP</td><td>Write Protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access from the core generates an access error exception to the ColdFire processor core.
0 Allows core read and write accesses to the SRAM module
1 Allows only core read accesses to the SRAM module
Note: This bit does not affect non-core write accesses.</td></tr>
<tr><td>7–6</td><td>Reserved, must be cleared.</td></tr>
<tr><td>5–1
C/I, SC, SD, UC, UD</td><td>Address Space Masks (ASn). These five bit fields allow types of accesses to be masked or inhibited from accessing the SRAM module. The address space mask bits are:
C/I = CPU space/interrupt acknowledge cycle mask
SC = Supervisor code address space mask
SD = Supervisor data address space mask
UC = User code address space mask
UD = User data address space mask

For each address space bit:
0 An access to the SRAM module can occur for this address space
1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module and is processed like any other non-SRAM reference.
These bits are useful for power management as detailed in Section 11.3.2, “Power Management.” In most applications, the C/I bit is set</td></tr>
<tr><td>0
V</td><td>Valid. When set, this bit enables the SRAM module; otherwise, the module is disabled. A hardware reset clears this bit.
0 Contents of RAMBAR are not valid
1 Contents of RAMBAR are valid</td></tr>
</table>

11.3 Initialization/Application Information

After a hardware reset, the SRAM module contents are undefined. The valid bit of the RAMBAR is cleared, disabling the processor port into the memory. If the SRAM requires initialization with instructions or data, perform the following steps:

1. Load the RAMBAR, mapping the SRAM module to the desired location within the address space.

2. Read the source data and write it to the SRAM. Various instructions support this function, including memory-to-memory move instructions, or the MOVEM opcode. The MOVEM instruction is optimized to generate line-sized burst fetches on 0-modulo-16 addresses, so this opcode generally provides maximum performance.
3. After the data loads into the SRAM, it may be appropriate to load a revised value into the RAMBAR with a new set of attributes. These attributes consist of the write-protect and address space mask fields.

The ColdFire processor or an external debugger using the debug module can perform these initialization functions.

11.3.1 SRAM Initialization Code

The following code segment describes how to initialize the SRAM. The code sets the base address of the SRAM at 0x2000_0000 and initializes the SRAM to zeros.

```
RAMBASE          EQU 0x20000000            ;set this variable to 0x20000000
RAMVALID         EQU 0x00000001
        move.l   #RAMBASE+RAMVALID,D0      ;load RAMBASE + valid bit into D0.
        movec.l  D0, RAMBAR                ;load RAMBAR and enable SRAM
```

The following loop initializes the entire SRAM to zero:

```
        lea.l    RAMBASE,A0                ;load pointer to SRAM
        move.l   #8192,D0                  ;load loop counter into D0 (SRAM size/4)

SRAM_INIT_LOOP:
        clr.l    (A0)+                     ;clear 4 bytes of SRAM
        clr.l    (A0)+                     ;clear 4 bytes of SRAM
        clr.l    (A0)+                     ;clear 4 bytes of SRAM
        clr.l    (A0)+                     ;clear 4 bytes of SRAM
        subq.l   #4,D0                     ;decrement loop counter
        bne.b    SRAM_INIT_LOOP            ;if done, then exit; else continue looping
```

11.3.2 Power Management

If the SRAM is used only for data operands, setting the AS*n* bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. Table 11-3 shows examples of typical RAMBAR settings.

Table 11-3. Typical RAMBAR Setting Examples

Data Contained in SRAM	RAMBAR[7:0]
Instruction Only	0x2B
Data Only	0x35
Instructions and Data	0x21

Chapter 12
Chip Configuration Module (CCM)

12.1 Introduction

This chapter describes the various operating configurations of the device. It provides a description of signals used by the CCM and a programming model.

12.1.1 Block Diagram

The chip configuration module (CCM) controls the chip configuration and mode of operation for the MCF52235.

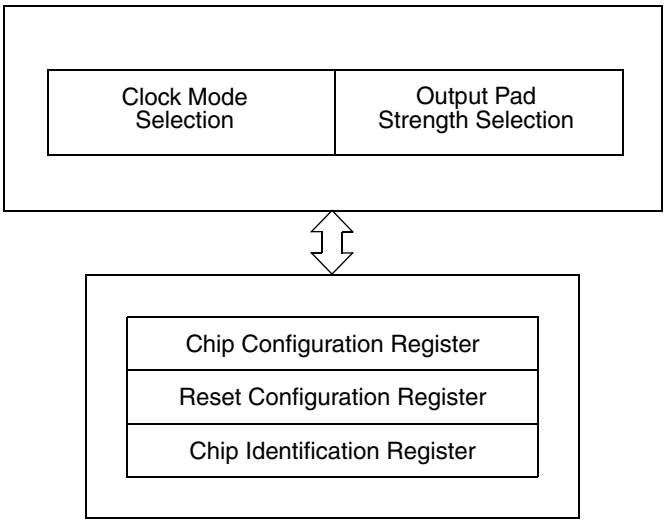


Figure 12-1. Chip Configuration Module Block Diagram

12.1.2 Features

The CCM selects the following:

- External clock or phase-lock loop (PLL) mode with internal or external reference
- Output pad drive strength
- Low-power configuration
- Processor status (PSTAT) and processor debug data (DDATA) functions
- BDM or JTAG mode

12.2 External Signal Descriptions

Table 12-1 provides an overview of the CCM signals.

Table 12-1. Signal Properties

Name	Function	Reset State
$\overline{\text{RCON}}$	Reset configuration select	Internal weak pull-up device

12.2.1 $\overline{\text{RCON}}$

If the external $\overline{\text{RCON}}$ pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins (see Section 12.4, “Functional Description”). The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

12.3 Memory Map/Register Definition

This subsection provides a description of the memory map and registers.

12.3.1 Programming Model

The CCM programming model consists of these registers:

- The chip configuration register (CCR) controls the main chip configuration.
- The reset configuration register (RCON) indicates the default chip configuration.
- The chip identification register (CIR) contains a unique part number.

Some control register bits are implemented as write-once bits. These bits are always readable, but after the bit has been written, additional writes have no effect, except during debug and test operations.

Some write-once bits can be read and written while in debug mode. When debug mode is exited, the chip configuration module resumes operation based on the current register values. If a write to a write-once register bit occurs while in debug mode, the register bit remains writable on exit from debug or test mode. Table 12-2 shows the accessibility of write-once bits.

Table 12-2. Write-Once Bits Read/Write Accessibility

Configuration	Read/Write Access
All configurations	Read-always
Debug operation	Write-always

12.3.2 Memory Map

Table 12-3. Chip Configuration Module Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Mode Access Only					
0x11_0004	Chip Configuration Register (CCR)	16	R	0x0001	12.3.3.1/12-3
0x11_0007	Low-Power Control Register (LPCR)[2]	8	R/W	0x00	9.2.4.1/9-8
0x11_0008	Reset Configuration Register (RCON)	16	R	0x0000	12.3.3.2/12-4
0x11_000A	Chip Identification Register (CIR)	16	R	0x2000	12.3.3.3/12-5
0x11_0010	Unimplemented[3]				—

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 SeeChapter 9, “Power Management,” for a description of the LPCR. It is shown here only to warn against accidental writes to this register.

3 Accessing an unimplemented address has no effect and causes a cycle termination transfer error.

NOTE

To safeguard against unintentionally activating test logic, write 0x0000 to the above reserved location during initialization (immediately after reset) to lock out test features. Setting any bits in the CCR may lead to unpredictable results.

12.3.3 Register Descriptions

The following section describes the CCM registers.

12.3.3.1 Chip Configuration Register (CCR)

IPSBAR Offset: 0x11_0004 (CCR) Access: Supervisor read-only

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

Figure 12-2. Chip Configuration Register (CCR)

Table 12-4. CCR Field Descriptions

Field	Description
1– —	Reserved, should be cleared.
6 SZEN	TSIZ[1:0] enable. This read/write bit enables the TSIZ[1:0] function of the external pins. 0 TSIZ[1:0] function disabled. 1 TSIZ[1:0] function enabled.
5 PSTEN	PST[3:0]/DDATA[3:0] enable. This read/write bit enables the Processor Status (PST) and Debug Data (DDATA) functions of the external pins. 0 PST/DDATA function disabled. 1 PST/DDATA function enabled.
4 —	Reserved, should be cleared.
3 BME	Bus monitor enable. This read/write bit enables the bus monitor to operate during external bus cycles. 0 Bus monitor disabled for external bus cycles. 1 Bus monitor enabled for external bus cycles. Table 12-2 shows the read/write accessibility of this write-once bit.
2–0 BMT	Bus monitor timing. This field selects the timeout period (in system clocks) for the bus monitor. 000 65536 001 32768 010 16384 011 8192 100 4096 101 2048 110 1024 111 512 Table 12-2 shows the read/write accessibility of this write-once bit.

12.3.3.2 Reset Configuration Register (RCON)

At reset, RCON determines the default operation of certain chip functions. All default functions defined by the RCON values can only be overridden during reset configuration (see Section 12.4.1, “Reset Configuration”) if the external $\overline{\text{RCON}}$ pin is asserted. RCON is a read-only register.

IPSBAR Offset: 0x11_0008 (RCON) Access: Supervisor read-only

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

Figure 12-3. Reset Configuration Register (RCON)

Table 12-5. RCON Field Descriptions

Field	Description
15–6	Reserved, should be cleared.
5 RLOAD	Pad driver load. Reflects the default pad driver strength configuration. 0 Partial drive strength (This is the default value.) 1 Full drive strength
4–1	Reserved, should be cleared.
0 MODE	Chip configuration mode. Reflects the default chip configuration mode. 0 Single chip – the chip takes instructions from the flash memory (This is the default value.) 1 Master chip – the chip takes instructions from $\overline{CS0}$

12.3.3.3 Chip Identification Register (CIR)

IPSBAR Offset: 0x11_000A (CIR) Access: read-only

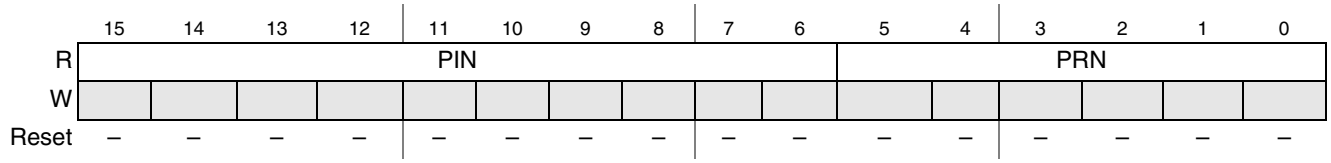


Figure 12-4. Chip Identification Register (CIR)

Table 12-6. CIR Field Description

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. MCF52230 = 0x48 MCF52231 = 0x49 MCF52233 = 0x4A MCF52234 = 0x4B MCF52235 = 0x4C
5–0 PRN	Part revision number. This number is increased by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order, beginning with zero.

12.4 Functional Description

Three functions are defined within the chip configuration module:

- Reset configuration
- Output pad strength configuration
- Clock mode selections

These functions are described in the following sections.

12.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. Table 12-7 shows the states of the external pins while in reset.

Table 12-7. Reset Configuration Pin States During Reset

Pin	Pin Function[1]	I/O	Output State	Input State
RCON	$\overline{\text{RCON}}$ function for all modes[2]	Input	—	Internal weak pull-up device

1 If the external $\overline{\text{RCON}}$ pin is not asserted during reset, pin functions are determined by the default operation mode defined in the RCON register. If the external $\overline{\text{RCON}}$ pin is asserted, pin functions are determined by the override values driven on the external data bus pins.

2 During reset, the external $\overline{\text{RCON}}$ pin assumes its $\overline{\text{RCON}}$ pin function, but this pin changes to the function defined by the chip operation mode immediately after reset. See Table 12-8.

If the $\overline{\text{RCON}}$ pin is not asserted during reset, the chip configuration and the reset configuration pin functions after reset are determined by RCON or fixed defaults, regardless of the states of the external data pins. The internal configuration signals are driven to levels specified by the RCON register's reset state for default module configuration.

If the $\overline{\text{RCON}}$ pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins (see Table 12-8). The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

Table 12-8. Configuration During Reset[1]

Pin(s) Affected	Default Configuration	Override Pins in Reset[2],	Function
All output pins	RCON[5] = 1	0	Partial strength
		1	Full strength[4]

1 Modifying the default configurations is possible only if the external $\overline{\text{RCON}}$ pin is asserted.

2 The external reset override circuitry drives the data bus pins with the override values while $\overline{\text{RSTOUT}}$ is asserted. It must stop driving the data bus pins within one CLKOUT cycle after $\overline{\text{RSTOUT}}$ is negated. To prevent contention with the external reset override circuitry, the reset override pins are forced to inputs during reset and do not become outputs until at least one CLKOUT cycle after $\overline{\text{RSTOUT}}$ is negated. $\overline{\text{RCON}}$ must also be negated within one cycle after $\overline{\text{RSTOUT}}$ is negated.

12.4.2 Output Pad Strength Configuration

Output pad strength is determined during reset configuration.

12.5 Reset

Reset initializes CCM registers to a known startup state as described in Section 12.3, "Memory Map/Register Definition." The CCM controls chip configuration at reset as described in Section 12.4, "Functional Description."

Chapter 13
System Control Module (SCM)

13.1 Introduction

This section details the functionality of the system control module (SCM) that provides the programming model for the system access control unit (SACU), system bus arbiter, 32-bit core watchdog timer (CWT), and system control registers and logic. Specifically, the system control includes the internal peripheral system (IPS) base address register (IPSBAR), the processor's dual-port RAM base address register (RAMBAR), and system control registers that include the core watchdog timer control.

13.2 Overview

The SCM provides the control and status for a variety of functions including base addressing and address space masking for the IPS peripherals and resources (IPSBAR) and the ColdFire core memory spaces (RAMBAR). The CPU core supports two memory banks, one for the internal SRAM and the other for the internal flash.

The SACU provides the mechanism needed to implement secure bus transactions to the system address space.

The programming model for the system bus arbitration resides in the SCM. The SCM sources the necessary control signals to the arbiter for bus master management.

The CWT provides a means of preventing system lockup due to uncontrolled software loops via a special software service sequence. If periodic software servicing action does not occur, the CWT times out with a programmed response (system reset or interrupt) to allow recovery or corrective action to be taken.

13.3 Features

The SCM includes these distinctive features:

- IPS base address register (IPSBAR)
  - — Base address location for 1-Gbyte peripheral space
  - — User control bits
- Processor-local memory base address register (RAMBAR)
- System control registers
  - — Core reset status register (CRSR) indicates type of last reset
  - — Core watchdog service register (CWSR) services watchdog timer
  - — Core watchdog control register (CWCR) for watchdog timer control

- System bus master arbitration programming model (MPARK)
- System access control unit (SACU) programming model
  - Master privilege register (MPR)
  - Peripheral access control registers (PACRs)
  - Grouped peripheral access control registers (GPACR0, GPACR1)

13.4 Memory Map and Register Definition

The memory map for the SCM registers is shown in Table 13-1. All the registers in the SCM are memory-mapped as offsets within the 1-Gbyte IPS address space and accesses are controlled to these registers by the control definitions programmed into the SACU.

Table 13-1. SCM Register Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x00_0000	IPS Base Address Register (IPSBAR)	32	R/W	0x4000_0001	13.5.1/13-3
0x00_0008	Memory Base Address Register (RAMBAR)	32	R/W	0x0000_0000	13.5.2/13-4
0x00_000C	Peripheral Power Management Register High (PPMRH)[2]	32	R/W	0x0000_0000	9.2.1/9-2
0x00_0010	Core Reset Status Register (CRSR)	8	R/W	See Section	13.5.3/13-6
0x00_0011	Core Watchdog Control Register (CWCR)	8	R/W	0x00	13.5.4/13-7
0x00_0012	Low-Power Interrupt Control Register (LPICR)[2]	8	R/W	0x00	9.2.2/9-5
0x00_0013	Core Watchdog Service Register (CWSR)	8	R/W	Uninitialized	13.5.5/13-9
0x00_0014	DMA Request Control Register (DMAREQC)	32	R/W	0x0000_0000	20.3.1/20-4
0x00_0018	Peripheral Power Management Register Low (PPMRL)[2]	32	R/W	0x0000_0001	9.2.1/9-2
0x00_001C	Default Bus Master Park Register (MPARK)	32	R/W	0x30E1_0000	13.6.3/13-11
0x00_0020	Master Privilege Register (MPR)	8	R/W	0x03	13.7.3.1/13-14
0x00_0021	Peripheral Power Management Set Register (PPMRS)[2]	8	W	0x00	9.2.3/9-7
0x00_0022	Peripheral Power Management Clear Register (PPMRC)[2]	32	R/W	0x00	9.2.4/9-8
0x00_0023	IPS Bus Timeout Monitor Register (IPSBMT)[2,3]	32	R/W	0x0000_0008	9.3/9-9
0x00_0024	Peripheral Access Control Register (PACR0)	8	R/W	0x00	13.7.3.2/13-15
0x00_0025	Peripheral Access Control Register (PACR1)	8	R/W	0x00	13.7.3.2/13-15
0x00_0026	Peripheral Access Control Register (PACR2)	8	R/W	0x00	13.7.3.2/13-15
0x00_0027	Peripheral Access Control Register (PACR3)	8	R/W	0x00	13.7.3.2/13-15
0x00_0028	Peripheral Access Control Register (PACR4)	8	R/W	0x00	13.7.3.2/13-15
0x00_0029	Peripheral Access Control Register (PACR5)	8	R/W	0x00	13.7.3.2/13-15
0x00_002A	Peripheral Access Control Register (PACR6)	8	R/W	0x00	13.7.3.2/13-15

Table 13-1. SCM Register Map (continued)

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x00_002B	Peripheral Access Control Register (PACR7)	8	R/W	0x00	13.7.3.2/13-15
0x00_002C	Peripheral Access Control Register (PACR8)	8	R/W	0x00	13.7.3.2/13-15
0x00_002E	Peripheral Access Control Register (PACR9)	8	R/W	0x00	13.7.3.2/13-15
0x00_0030	Grouped Peripheral Access Control Register 0 (GPACR0)	8	R/W	0x00	13.7.3.3/13-16
0x00_0031	Grouped Peripheral Access Control Register 1 (GPACR1)	8	R/W	0x00	13.7.3.3/13-16

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 The PPMRH, LPICR, PPMRL, PPMRS, PPMRC, and IPSBMT are described in Chapter 9, “Power Management.”

3 Register must be addressed as a byte.

13.5 Register Descriptions

13.5.1 Internal Peripheral System Base Address Register (IPSBAR)

The IPSBAR specifies the base address for the 1-Gbyte memory space associated with the on-chip peripherals. At reset, the base address is loaded with a default location of 0x4000_0000 and marked as valid (IPSBAR[V]=1). If desired, the address space associated with the internal modules can be moved by loading a different value into the IPSBAR at a later time.

NOTE

Accessing reserved IPSBAR memory space could result in an unterminated bus cycle that causes the core to hang. Only a hard reset allows the core to recover from this state. Therefore, all bus accesses to IPSBAR space should fall within a module’s memory map space.

If an address hits in overlapping memory regions, the following priority is used to determine what memory is accessed:

1. IPSBAR
2. RAMBAR

NOTE

This is the list of memory access priorities when viewed from the processor core.

See Figure 13-1 and Table 13-2 for descriptions of the bits in IPSBAR.

IPSBAR Offset: 0x000 (IPSBAR) Access: read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA31	BA30	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 13-1. IPS Base Address Register (IPSBAR)

Table 13-2. IPSBAR Field Description

Field	Description
31–30 BA	Base address. Defines the base address of the 1-Gbyte internal peripheral space. This is the starting address for the IPS registers when the valid bit is set.
29–1	Reserved, should be cleared.
0 V	Valid. Enables/disables the IPS Base address region. V is set at reset. 0 IPS Base address is not valid. 1 IPS Base address is valid.

13.5.2 Memory Base Address Register (RAMBAR)

The device supports dual-ported local SRAM memory. This processor-local memory can be accessed directly by the core and/or other system bus masters. Because this memory provides single-cycle accesses at processor speed, it is ideal for applications where double-buffer schemes can be used to maximize system-level performance. For example, a DMA channel in a typical double-buffer application (also known as a ping-pong scheme) may load data into one portion of the dual-ported SRAM while the processor is manipulating data in another portion of the SRAM. After the processor completes the data calculations, it begins processing the recently loaded buffer while the DMA moves out the recently calculated data from the other buffer, and reloads the next data block into the recently freed memory region. The process repeats with the processor and the DMA ping-ponging between alternate regions of the dual-ported SRAM.

The device design implements the dual-ported SRAM in the memory space defined by the RAMBAR register. There are two physical copies of the RAMBAR register: one located in the processor core and accessible only via the privileged MOVEC instruction at CPU space address 0xC05 and another located in the SCM at IPSBAR + 0x008. ColdFire core accesses to this memory are controlled by the processor-local copy of the RAMBAR, while module accesses are enabled by the SCM's RAMBAR.

The physical base address programmed in both copies of the RAMBAR is typically the same value; however, they can be programmed to different values. By definition, the base address must be a 0-modulo-size value.

IPSBAR Offset: 0x008 (RAMBAR) Access: read/write

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

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

Figure 13-2. Memory Base Address Register (RAMBAR)

Table 13-3. RAMBAR Field Description

Field	Description
31–16 BA	Base address. Defines the memory module's base address on a 64-Kbyte boundary corresponding to the physical array location within the 4 Gbyte address space supported by ColdFire.
15–10	Reserved, should be cleared.
9 BDE	Back door enable. Qualifies non-core master module accesses to the memory. 0 Disables non-core master module accesses to the internal SRAM 1 Enables non-core master module accesses to the internal SRAM **Note:** The SPV bit in the CPU's RAMBAR must also be set to allow dual port access to the SRAM. For more information, see Section 11.2.1, "SRAM Base Address Register (RAMBAR)."
8–0	Reserved, should be cleared.

The SRAM modules are configured through the RAMBAR shown in Figure 13-2.

- RAMBAR specifies the base address of the SRAM.
- All undefined bits are reserved. These bits are ignored during writes to the RAMBAR and return zeros when read.
- The back door enable bit, RAMBAR[BDE], is cleared at reset, disabling the module access to the SRAM.

NOTE

The RAMBAR default value of 0x0000_0000 is invalid. The RAMBAR located in the processor's CPU space must be initialized with the valid bit set before the CPU (or modules) can access the on-chip SRAM (see Chapter 11, "Static RAM (SRAM)," for more information.

For details on the processor's view of the local SRAM memories, see Section 11.2.1, "SRAM Base Address Register (RAMBAR)."

13.5.3 Core Reset Status Register (CRSR)

The CRSR contains a bit that indicates the reset source to the CPU. When the EXT bit (bit 7) reads as 1, an external device driving $\overline{\text{RSTI}}$ has caused the most recent reset. The CRSR is updated by the control logic when the reset is complete. Only one bit is set at a time in the CRSR. The register reflects the cause of the most recent reset. To clear a bit, a logic 1 must be written to the bit location; writing a zero has no effect. Unused bits are reserved and should not be written.

NOTE

The reset status register (RSR) in the reset controller module provides indication of all reset sources except the core watchdog timer (see Chapter 10, "Reset Controller Module").

IPSBAR Access: read/write
Offset: 0x0010 (CRSR)

	7	6	5	4	3	2	1	0
R	EXT	0	0	0	0	0	0	0
W								
Reset:	See Note	0	0	0	0	0	0	0

Note: The reset value of EXT depend on the last reset source. All other bits are initialized to zero.

Figure 13-3. Core Reset Status Register (CRSR)

Table 13-4. CRSR Field Descriptions

Field	Description
7 EXT	External reset. 1 An external device driving $\overline{\text{RSTI}}$ caused the last reset. Assertion of reset by an external device causes the processor core to initiate reset exception processing. All registers are forced to their initial state.
6–0	Reserved, should read as 0. Do not write to these locations.

13.5.4 Core Watchdog Control Register (CWCR)

The core watchdog timer prevents system lockup if the software becomes trapped in a loop with no controlled exit. The core watchdog timer can be enabled or disabled through CWCR[CWE]. It is disabled by default. If enabled, the watchdog timer requires the periodic execution of a core watchdog servicing sequence. If this periodic servicing action does not occur, the timer times out, resulting in a watchdog timer interrupt, as programmed by CWCR[CWRI]. If the timer times out and the core watchdog transfer acknowledge enable bit (CWCR[CWTA]) is set, a watchdog timer interrupt is asserted. If a core watchdog timer interrupt acknowledge cycle has not occurred after another timeout, CWT TA is asserted in an attempt to allow the interrupt acknowledge cycle to proceed by terminating the bus cycle. The setting of CWCR[CWTAVAL] indicates that the watchdog timer TA was asserted.

To prevent the core watchdog timer from interrupting, the CWSR must be serviced by performing the following sequence:

1. Write 0x55 to CWSR.
2. Write 0xAA to the CWSR.

Both writes must occur in order before the time-out, but any number of instructions can be executed between the two writes. This order allows interrupts and exceptions to occur, if necessary, between the two writes. Caution should be exercised when changing CWCR values after the software watchdog timer has been enabled with the setting of CWCR[CWE], because it is difficult to determine the state of the core watchdog timer while it is running. The countdown value is constantly compared with the time-out period specified by CWCR[CWT]. The following steps must be taken to change CWT:

1. Disable the core watchdog timer by clearing CWCR[CWE].
2. Reset the counter by writing 0x55 and then 0xAA to CWSR.
3. Update CWCR[CWT].
4. Re-enable the core watchdog timer by setting CWCR[CWE]. This step can be performed in step 3.

The CWCR controls the software watchdog timer, time-out periods, and software watchdog timer transfer acknowledge. The register can be read at any time, but can be written only if the CWT is not pending. At system reset, the software watchdog timer is disabled.

IPSBAR Access: read/write

Offset: 0x0011 (CWCR)

	7	6	5	4	3	2	1	0
R W	CWE	CWRI	CWT[2:0]			CWTA	CWTAVAL	CWTIF
Reset:	0	0	0	0	0	0	0	0

Figure 13-4. Core Watchdog Control Register (CWCR)

Table 13-5. CWCR Field Description

Field	Description
7 CWE	Core watchdog enable. 0 SWT disabled. 1 SWT enabled.
6 CWRI	Core watchdog interrupt select. 0 If a time-out occurs, the CWT generates an interrupt to the processor core. The interrupt level for the CWT is programmed in the interrupt control register 8 (ICR8) of INTC0. 1 Reserved. If a one is written, undetermined behavior results. **Note:** If a core reset is required, the watchdog interrupt should set the soft reset bit in the interrupt controller.
5–3 CWT[2:0]	Core watchdog timing delay. These bits select the timeout period for the CWT as shown in the following table. At system reset, the CWT field is cleared signaling the minimum time-out period but the watchdog is disabled (CWCR[CWE] = 0). the following table shows the core watchdog timer delay.
2 CWTA	Core watchdog transfer acknowledge enable. 0 CWTA Transfer acknowledge disabled. 1 CWTA Transfer Acknowledge enabled. After one CWT time-out period of the unacknowledged assertion of the CWT interrupt, the transfer acknowledge asserts, which allows CWT to terminate a bus cycle and allow the interrupt acknowledge to occur.
1 CWTAVAL	Core watchdog transfer acknowledge valid. 0 CWTA Transfer Acknowledge has not occurred. 1 CWTA Transfer Acknowledge has occurred. Write a 1 to clear this flag bit.
0 CWTIF	Core watchdog timer interrupt flag. 0 CWT interrupt has not occurred 1 CWT interrupt has occurred. Write a 1 to clear the interrupt request.

CWT [2:0]	CWT Time-Out Period
000	2^9 Bus clock frequency
001	2^{11} Bus clock frequency
010	2^{13} Bus clock frequency
011	2^{15} Bus clock frequency
100	2^{19} Bus clock frequency
101	2^{23} Bus clock frequency
110	2^{27} Bus clock frequency
111	2^{31} Bus clock frequency

13.5.5 Core Watchdog Service Register (CWSR)

The software watchdog service sequence must be performed using the CWSR as a data register to prevent a CWT time-out. The service sequence requires two writes to this data register: first a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the CWT time-out, but any number of instructions or accesses to the CWSR can be executed between the two writes. If the CWT has already timed out, writing to this register has no effect in negating the CWT interrupt. Figure 13-5 illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.

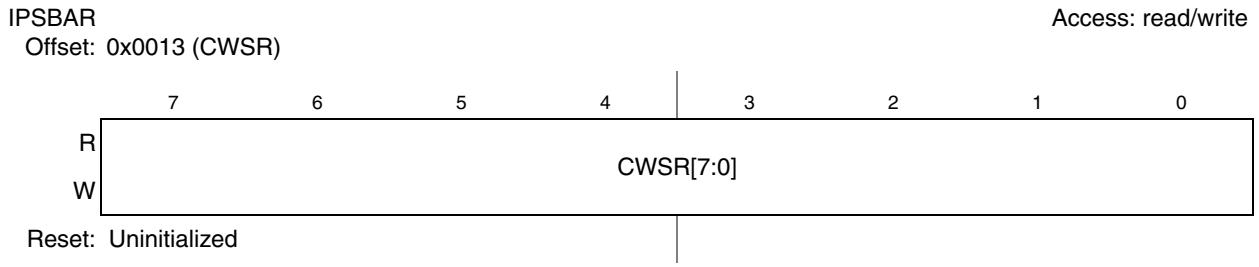


Figure 13-5. Core Watchdog Service Register (CWSR)

13.6 Internal Bus Arbitration

The internal bus arbitration is performed by the on-chip bus arbiter, which containing the arbitration logic that controls which of up to four MBus masters (M0–M3 in Figure 13-6) has access to the external buses. The function of the arbitration logic is described in this section.

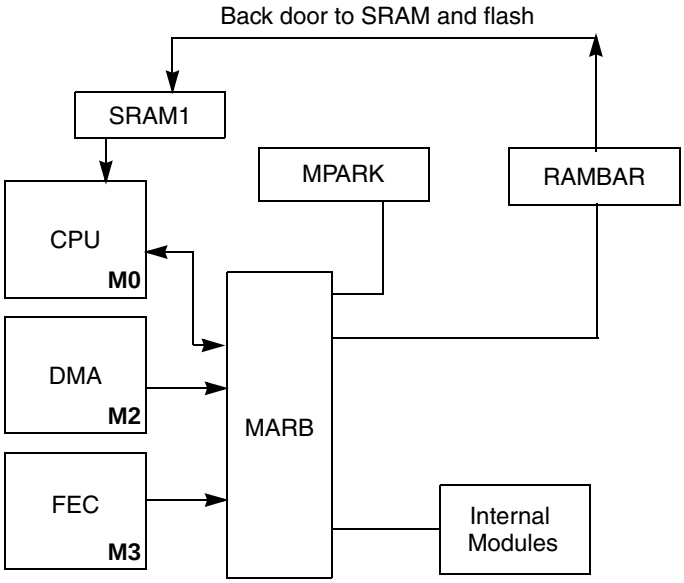


Figure 13-6. Arbiter Module Functions

13.6.1 Overview

The basic functionality is that of a 2-port, pipelined internal bus arbitration module with the following attributes:

- The master pointed to by the current arbitration pointer may get on the bus with zero latency if the address phase is available. All other requesters face at least a one cycle arbitration pipeline delay to meet bus timing constraints on address phase hold.
- If a requester receives an immediate address phase (it is pointed to by the current arbitration pointer and the bus address phase is available), it is the current bus master and is ignored by arbitration. All remaining requesting ports are evaluated by the arbitration algorithm to determine the next-state arbitration pointer.
- There are two arbitration algorithms: fixed and round-robin. Fixed arbitration sets the next-state arbitration pointer to the highest priority requester. Round-robin arbitration sets the next-state arbitration pointer to the highest priority requester (calculated by adding a requester's fixed priority to the current bus master's fixed priority and then taking this sum modulo the number of possible bus masters).
- The default priority is DMA (M2) > CPU (M0) > FEC (M3), where M2 is the highest and M3 the lowest priority.
- There are two actions for an idle arbitration cycle, leave the current arbitration pointer as is or set it to the lowest priority requester.
- The anti-lock-out logic for the fixed priority scheme forces the arbitration algorithm to round-robin if any requester has been held for longer than a specified cycle count.

13.6.2 Arbitration Algorithms

There are two modes of arbitration: fixed and round-robin. This section discusses the differences between them.

13.6.2.1 Round-Robin Mode

Round-robin arbitration is the default mode after reset. This scheme cycles through the sequence of masters as specified by MPARK[M*n*_PRTY] bits. Upon completion of a transfer, the master is given the lowest priority and the priority for all other masters is increased by one.

	M2 =01 M0 = 10 M3 = 00
next +1	M2 =10 M0 = 11 M3 = 01
next +2	M2 =11 M0 = 00 M3 = 10
next +3	M2 =00 M0 = 01 M3 = 11

If no masters are requesting, the arbitration unit must park, pointing at one of the masters. There are two possibilities: park the arbitration unit on the last active master, or park pointing to the highest priority master. Setting MPARK[PRK_LAST] causes the arbitration pointer to be parked on the highest priority master. In round-robin mode, programming the timeout enable and lockout bits MPARK[13,11:8] has no effect on the arbitration.

13.6.2.2 Fixed Mode

In fixed arbitration, the master with highest priority (as specified by the MPARK[M*n*_PRTY] bits) wins the bus. That master relinquishes the bus when all transfers to that master are complete.

If MPARK[TIMEOUT] is set, a counter increments for each master for every cycle it is denied access. When a counter reaches the limit set by MPARK[LCKOUT_TIME], the arbitration algorithm is changed to round-robin arbitration mode until all locks are cleared. The arbitration then returns to fixed mode and the highest priority master is granted the bus.

As in round-robin mode, if no masters are requesting, the arbitration pointer parks on the highest priority master if MPARK[PRK_LAST] is set or parks on the master whose last requested the bus if cleared.

13.6.3 Bus Master Park Register (MPARK)

The MPARK controls the operation of the system bus arbitration module. The platform bus master connections are defined as the following:

- Master 2 (M2): 4-channel DMA
- Master 3 (M3): Fast Ethernet controller (FEC)
- Master 0 (M0): V2 ColdFire Core

IPSBAR Access: read/write

Offset: 0x001C (MPARK)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	M2_P_EN	BCR24BIT	M3_PRTY		M2_PRTY		M0_PRTY		0	0
W																
Reset	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	1

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

Figure 13-7. Default Bus Master Park Register (MPARK)

Table 13-6. MPARK Field Descriptions

Field	Description
31–26	Reserved, should be cleared.
25 M2_P_EN	DMA bandwidth control enable 0 Disable the use of the DMA's bandwidth control to elevate the priority of its bus requests. 1 Enable the use of the DMA's bandwidth control to elevate the priority of its bus requests.
24 BCR24BIT	Enables the use of 24 bit byte count registers in the DMA module 0 DMA BCRs function as 16 bit counters. 1 DMA BCRs function as 24 bit counters.
23–22 M3_PRTY	Master priority level for master 3 (FEC) 00 Fourth (lowest) priority 01 Third priority 10 Second priority 11 First (highest) priority

Table 13-6. MPARK Field Descriptions (continued)

Field	Description
21–20 M2_PRTY	Master priority level for master 2 (DMA Controller) 00 Fourth (lowest) priority 01 Third priority 10 Second priority 11 First (highest) priority
19–18 M0_PRTY	Master priority level for master 0 (ColdFire Core) 00 Fourth (lowest) priority 01 Third priority 10 Second priority 11 First (highest) priority
17–15	Reserved, should be cleared.
14 FIXED	Fixed or round robin arbitration 0 Round robin arbitration 1 Fixed arbitration
13 TIMEOUT	Timeout Enable 0 Disable count for when a master is locked out by other masters. 1 Enable count for when a master is locked out by other masters and allow access when LCKOUT_TIME is reached.
12 PRKLAST	Park on the last active master or highest priority master if no masters are active 0 Park on last active master 1 Park on highest priority master
11–8 LCKOUT_TIME	Lock-out Time. Lock-out time for a master being denied the bus. The lock out time is defined as 2^ LCKOUT_TIME[3:0].
7–0	Reserved, should be cleared.

The initial state of the master priorities is M3 > M2 > M1 > M0. System software should guarantee that the programmed M*n*_PRTY fields are unique, otherwise the hardware defaults to the initial-state priorities.

NOTE

The M1_PRTY field should not be set for a priority higher than third (default).

13.7 System Access Control Unit (SACU)

This section details the functionality of the system access control unit (SACU), which provides the mechanism needed to implement secure bus transactions to the address space mapped to the internal modules.

13.7.1 Overview

The SACU supports the traditional model of two privilege levels: supervisor and user. Typically, memory references with the supervisor attribute have total accessibility to all the resources in the system, while user mode references cannot access system control and configuration registers. In many systems, the operating system executes in supervisor mode, while application software executes in user mode.

The SACU further partitions the access control functions into two parts: one control register defines the privilege level associated with each bus master, and another set of control registers define the access levels associated with the peripheral modules and memory space.

The SACU's programming model is physically implemented as part of the system control module (SCM) with the actual access control logic included as part of the arbitration controller. Each bus transaction targeted for the IPS space is first checked to see if its privilege rights allow access to the given memory space. If the privilege rights are correct, the access proceeds on the bus. If the privilege rights are insufficient for the targeted memory space, the transfer is immediately aborted and terminated with an exception, and the targeted module is not accessed.

13.7.2 Features

Each bus transfer can be classified by its privilege level and the reference type. The complete set of access types includes the following:

- Supervisor instruction fetch
- Supervisor operand read
- Supervisor operand write
- User instruction fetch
- User operand read
- User operand write

Instruction fetch accesses are associated with the execute attribute.

It should be noted that while the bus does not implement the concept of reference type (code versus data) and only supports the user/supervisor privilege level, the reference type attribute is supported by the system bus. Accordingly, the access checking associated with privilege level and reference type is performed in the IPS controller using the attributes associated with the reference from the system bus.

The SACU partitions the access control mechanisms into three distinct functions:

- Master privilege register (MPR)
  - — Allows each bus master to be assigned a privilege level:
    - – Disable the master's user/supervisor attribute and force to user mode access
    - – Enable the master's user/supervisor attribute
  - — The reset state provides supervisor privilege to the processor core (bus master 0).
  - — Input signals allow the non-core bus masters to have their user/supervisor attribute enabled at reset. This is intended to support the concept of a trusted bus master, and also controls the ability of a bus master to modify the register state of any of the SACU control registers; that is, only trusted masters can modify the control registers.
- Peripheral access control registers (PACRs)
  - — Provide read/write access rights, supervisor/user privilege levels.
  - — Reset state provides supervisor-only read/write access to these modules.
  - — Nine 8-bit registers control access to 17 of the on-chip peripheral modules
- Grouped peripheral access control registers (GPACR0, GPACR1)

— Provide read/write/execute access rights, supervisor/user privilege levels.
— One single register (GPACR0) controls access to 14 of the on-chip peripheral modules.
— One register (GPACR1) controls access for IPS reads and writes to the flash module.
— Reset state provides supervisor-only read/write access to each of these peripheral spaces.

13.7.3 Memory Map/Register Definition

The memory map for the SACU program-visible registers within the system control module (SCM) is shown in Figure 13-7. The MPR, PACR, and GPACRs are 8 bits wide.

Table 13-7. SACU Register Memory Map

IPSBAR Offset	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
0x020	MPR		PPMRS		PPMRC		IPSBMT	
0x024	PACR0		PACR1		PACR2		PACR3	
0x028	PACR4		PACR5		PACR6		PACR7	
0x02C	PACR8		PACR9		—		—	
0x030	GPACR0		GPACR1		—		—	
0x034	—		—		—		—	
0x038	—		—		—		—	
0x03C	—		—		—		—	

13.7.3.1 Master Privilege Register (MPR)

The MPR specifies the access privilege level associated with each bus master in the platform. The register provides one bit per bus master, where bit 3 corresponds to master 3 (Fast Ethernet controller), bit 2 to master 2 (DMA Controller), and bit 0 to master 0 (ColdFire core).

IPSBAR Access: read/write
Offset: 0x0020 (MPR)

	7	6	5	4	3	2	1	0
R	0	0	0	0	MPR			
W								
Reset:	0	0	0	0	0	0	1	1

Figure 13-8. Master Privilege Register (MPR)

Table 13-8. MPR[n] Field Descriptions

Field	Description
7–4	Reserved. Should be cleared.
3–0 MPR	Each 1-bit field defines the access privilege level of the given bus master *n*. 0 All bus master accesses are in user mode. 1 All bus master accesses use the sourced user/supervisor attribute.

Only trusted bus masters can modify the access control registers. If a non-trusted bus master attempts to write any of the SACU control registers, the access is aborted with an error termination and the registers remain unaffected.

The processor core is connected to bus master 0 and is always treated as a trusted bus master. Accordingly, MPR[0] is forced to 1 at reset.

13.7.3.2 Peripheral Access Control Registers (PACR0–PACR9)

Access to several on-chip peripherals is controlled by shared peripheral access control registers. A single PACR defines the access level for each of the two modules. These modules only support operand reads and writes. Each PACR follows the format illustrated in Figure 13-9. For a list of PACRs and the modules that they control, refer to Table 13-11.

IPSBAR Access: read/write

Offset: 0x0024 + Offset[1] (PACR*n*)

	7	6	5	4	3	2	1	0
R W	LOCK1	ACCESS_CTRL1			LOCK0	ACCESS_CTRL0		
Reset:	0	0	0	0	0	0	0	0

Figure 13-9. Peripheral Access Control Register (PACR*n*)

1 See Table 13-1 for the full list of addresses.

Table 13-9. PACR Field Descriptions

Field	Description
7 LOCK1	This bit, when set, prevents subsequent writes to ACCESSCTRL1. Any attempted write to the PACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
6–4 ACCESS_CTRL1	This 3-bit field defines the access control for the given platform peripheral. The encodings for this field are shown in Table 13-10.
3 LOCK0	This bit, when set, prevents subsequent writes to ACCESSCTRL0. Any attempted write to the PACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
2–0 ACCESS_CTRL0	This 3-bit field defines the access control for the given platform peripheral. The encodings for this field are shown in Table 13-10.

Table 13-10. PACR ACCESSCTRL Bit Encodings

Bits	Supervisor Mode	User Mode
000	Read/Write	No Access
001	Read	No Access
010	Read	Read
011	Read	No Access
100	Read/Write	Read/Write
101	Read/Write	Read
110	Read/Write	Read/Write
111	No Access	No Access

Table 13-11. Peripheral Access Control Registers (PACRs)

IPSBAR Offset	Name	Modules Controlled[1]	
		ACCESS_CTRL1	ACCESS_CTRL0
0x024	PACR0	SCM	—
0x025	PACR1	—	DMA
0x026	PACR2	UART0	UART1
0x027	PACR3	UART2	—
0x028	PACR4	I^2C	QSPI
0x029	PACR5	—	—
0x02A	PACR6	DTIM0	DTIM1
0x02B	PACR7	DTIM2	DTIM3
0x02C	PACR8	INTC0	INTC1
0x02E	PACR9	FEC	—

[1] A value of "—" in these columns indicates that the bits are not associated with any module and are reserved.

At reset, these on-chip modules are configured to have only supervisor read/write access capabilities. If an instruction fetch access to any of these peripheral modules is attempted, the IPS bus cycle is immediately terminated with an error.

13.7.3.3 Grouped Peripheral Access Control Registers (GPACR0 & GPACR1)

The on-chip peripheral space starting at IPSBAR is subdivided into sixteen 64-Mbyte regions. Each of the first two regions has a unique access control register associated with it. The other 14 regions are in reserved space; the access control registers for these regions are not implemented. Bits [29:26] of the address select the specific GPACRn to be used for a given reference within the IPS address space. These access control registers are 8 bits wide so that read, write, and execute attributes may be assigned to the given IPS region.

NOTE

The access control for modules with memory space protected by PACR0–PACR9 are determined by the PACR0–PACR9 settings. The access control is not affected by GPACR0, even though the modules are mapped in its 64-Mbyte address space.

IPSBAR 0x0030 (GPACR0) Access: read/write
Offsets: 0x0031 (GPACR1)

	7	6	5	4	3	2	1	0
R	LOCK	0	0	0	ACCESS_CTRL			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 13-10. GPACR Register

Table 13-12. Grouped Peripheral Access Control Register (GPACR) Field Descriptions

Field	Description
7 LOCK	This bit, after set, prevents subsequent writes to the GPACR. Any attempted write to the GPACR generates an error termination and the contents of the register are not affected. Only a system reset clears this flag.
6–4	Reserved, should be cleared.
3–0 ACCESS_CTRL	This 4-bit field defines the access control for the given memory region. The encodings for this field are shown in Table 13-13.

At reset, these on-chip modules are configured to have only supervisor read/write access capabilities. Bit encodings for the ACCESS_CTRL field in the GPACR are shown in Table 13-13. Table 13-14 shows the memory space protected by the GPACRs and the modules mapped to these spaces.

Table 13-13. GPACR ACCESS_CTRL Bit Encodings

Bits	Supervisor Mode	User Mode
0000	Read / Write	No Access
0001	Read	No Access
0010	Read	Read
0011	Read	No Access
0100	Read / Write	Read / Write
0101	Read / Write	Read
0110	Read / Write	Read / Write
0111	No Access	No Access
1000	Read / Write / Execute	No Access
1001	Read / Execute	No Access
1010	Read / Execute	Read / Execute
1011	Execute	No Access

Table 13-13. GPACR ACCESS_CTRL Bit Encodings (continued)

Bits	Supervisor Mode	User Mode
1100	Read / Write / Execute	Read / Write / Execute
1101	Read / Write / Execute	Read / Execute
1110	Read / Write	Read
1111	Read / Write / Execute	Execute

Table 13-14. GPACR Address Space

Register	Space Protected (IPSBAR Offset)	Modules Protected
GPACR0	0x0000_0000–0x03FF_FFFF	Ports, CCM, PMM, Reset controller, Clock, EPORT, WDOG, PIT0–PIT3, QADC, GPTA, GPTB, FlexCAN, CFM (Control)
GPACR1	0x0400_0000–0x07FF_FFFF	CFM (Flash module's backdoor access for programming or access by a bus master other than the core)

Chapter 14
General Purpose I/O Module

14.1 Introduction

Many of the pins associated with the external interface may be used for several different functions. When not used for their primary function, many of the pins may be used as general-purpose digital I/O pins. In some cases, the pin function is set by the operating mode, and the alternate pin functions are not supported.

The digital I/O pins are grouped into 8-bit ports. Some ports do not use all 8 bits. Each port has registers that configure, monitor, and control the port pins. Figure 14-1 is a block diagram of the MCF52235 ports.

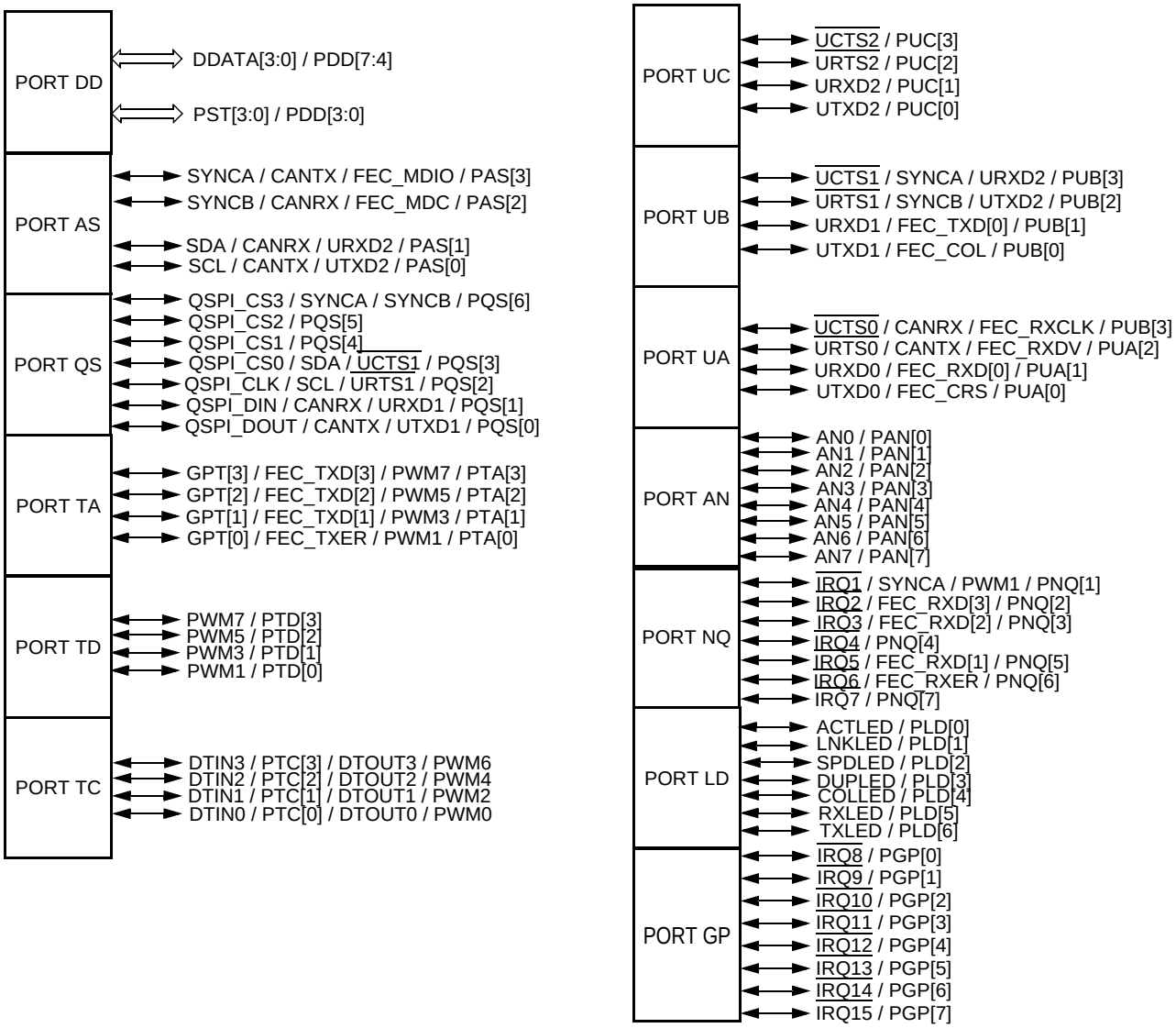


Figure 14-1. General Purpose I/O Module Block Diagram

14.2 Overview

The MCF52235 ports module controls the configuration for the following external pins:

- External bus accesses
- Chip selects
- Debug data
- Processor status
- FlexCAN transmit/receive data
- I^2C serial control
- QSPI
- UART transmit/receive

- 32-bit DMA timers

14.3 Features

The MCF52235 ports includes these distinctive features:

- Control of primary function use on all ports
- Digital I/O support for all ports; registers for:
  - Storing output pin data
  - Controlling pin data direction
  - Reading current pin state
  - Setting and clearing output pin data registers

14.4 Signal Descriptions

Refer to Chapter 2, “Signal Descriptions,” for more detailed information on the different signals and pins.

14.5 Memory Map/Register Definition

14.5.1 Ports Memory Map

Table 14-1 summarizes all the registers in the MCF52235 ports address space.

Table 14-1. Ports Module Memory Map

Address[1]	31–24	23–16	15–8	7–0	Access[2]
Port Output Data Registers					
0x10_0000	Reserved				S/U
0x10_0004	Reserved				S/U
0x10_0008	PORTNQ	Reserved	PORTAN	PORTAS	S/U
0x10_000C	PORTQS	Reserved	PORTTA	PORTTC	S/U
0x10_0010	PORTTD	PORTUA	PORTUB	PORTUC	S/U
0x10_0014	PORTDD	PORTLD	PORTGP	Reserved	S/U
Port Data Direction Registers					
0x10_0018	Reserved				S/U
0x10_001C	Reserved				S/U
0x10_0020	DDRNQ	Reserved	DDRAN	DDRAS	S/U
0x10_0024	DDRQS	Reserved	DDRTA	DDRTC	S/U
0x10_0028	DDRTD	DDRUA	DDRUB	DDRUC	S/U
0x10_002C	DDRDD	DDRLD	DDRGP	Reserved	S/U
Port Pin Data/Set Data Registers					
0x10_0030	Reserved				S/U
0x10_0034	Reserved				S/U
0x10_0038	SETNQ	Reserved	SETAN	SETAS	S/U
0x10_003C	SETQS	Reserved	SETTA	SETTC	S/U
0x10_0040	SETTD	SETUA	SETUB	SETUC	S/U
0x10_0044	SETDD	SETLD	SETGP	Reserved	S/U
Port Clear Output Data Registers					
0x10_0048	Reserved				S/U
0x10_004C	Reserved				S/U
0x10_0050	CLRNQ	Reserved	CLRAN	CLRAS	S/U
0x10_0054	CLRQS	Reserved	CLRTA	CLRTC	S/U
0x10_0058	CLRTD	CLRUA	CLRUB	CLRUC	S/U
0x10_005C	CLRDD	CLRLD	CLRGP	Reserved	S/U
Port Pin Assignment Registers					
0x10_0060	Reserved				S/U
0x10_0064	Reserved				S/U
0x10_0068	PNQPAR		PANPAR	PASPAR	S/U
0x10_006C	PQSPAR		PTAPAR	PTCPAR	S/U
0x10_0070	PTDPAR	PUAPAR	PUBPAR	PUCPAR	S/U
0x10_0074	PDDPAR	PLDPAR	PGPPAR	Reserved	S/U
Port Pad Control Registers					
0x10_0078	PWOR[15:0]		PDSR1		S/U
0x10_007C	PDSR0				S/U

[1]The register address is the sum of the IPSBAR address and the value in this column.
[2]S/U equals supervisor or user mode access. User mode accesses to supervisor-only addresses have no effect and cause a cycle termination transfer error.

14.6 Register Descriptions

14.6.1 Port Output Data Registers (PORT*n*)

The PORT*n* registers store the data to be driven on the corresponding port *n* pins when the pins are configured for digital output.

The PORT*n* registers with a full 8-bit implementation are shown in Figure 14-2. The remaining PORT*n* registers use fewer than 8 bits. Their bit definitions are shown in Figure 14-3, Figure 14-4, and Figure 14-5. The fields are described in Table 14-2, which applies to all PORT*n* registers.

The PORT*n* registers are read/write. At reset, all bits in the PORT*n* registers are set.

Reading a PORT*n* register returns the current values in the register, not the port *n* pin values.

PORT*n* bits can be set by setting the PORT*n* register, or by setting the corresponding bits in the PORT*n*P/SET*n* register. They can be cleared by clearing the PORT*n* register, or by clearing the corresponding bits in the CLR*n* register.

IPSBAR 0x10_000A (PORTAN) Access: User read/write
Offsets: 0x10_0014 (PORTDD)
0x10_0016 (PORTGP)

	7	6	5	4	3	2	1	0
R W	PORT*n*7	PORT*n*6	PORT*n*5	PORT*n*4	PORT*n*3	PORT*n*2	PORT*n*1	PORT*n*0
Reset:	1	1	1	1	1	1	1	1

Figure 14-2. Port Output Data Registers with Bits 7:0 Implemented (PORTAN, PORTDD, PORTGP)

IPSBAR 0x10_000B (PORTAS) Access: User read/write
Offsets: 0x10_000E (PORTTA)
0x10_000F (PORTTC)
0x10_0010 (PORTTD)
0x10_0011 (PORTUA)
0x10_0012 (PORTUB)
0x10_0013 (PORTUC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	PORT*n*3	PORT*n*2	PORT*n*1	PORT*n*0
W								
Reset:	0	0	0	0	1	1	1	1

Figure 14-3. Port Output Data Registers with Bits 3:0 Implemented (PORTAS, PORTTA, PORTTC, PORTTD, PORTUA, PORTUB, PORTUC)

IPSBAR 0x10_000C (PORTQS) Access: User read/write
Offsets: 0x10_0015 (PORTLD)

	7	6	5	4	3	2	1	0
R	0	PORT*n*6	PORT*n*5	PORT*n*4	PORT*n*3	PORT*n*2	PORT*n*1	PORT*n*0
W								
Reset:	0	1	1	1	1	1	1	1

Figure 14-4. Port Output Data Registers with Bits 6:0 Implemented (PORTQS, PORTLD)

IPSBAR Access: User read/write
Offset: 0x10_0008 (PORTNQ)

	7	6	5	4	3	2	1	0
R	PORT*n*7	PORT*n*6	PORT*n*5	PORT*n*4	PORT*n*3	PORT*n*2	PORT*n*1	0
W								
Reset:	1	1	1	1	1	1	1	0

Figure 14-5. Port NQ Output Data Register (PORTNQ)

Table 14-2. PORT*n* Field Descriptions

Field	Description
Port*n*x	Data to be driven when the port pin is configured as a digital output. 1 Output is a logic 1 0 Output is a logic 0

14.6.2 Port Data Direction Registers (DDR*n*)

The DDR*n* registers control the direction of the port *n* pin drivers when the pins are configured for digital I/O.

The DDRn registers with a full 8-bit implementation are shown in Figure 14-6. The remaining DDR*n* registers use fewer than eight bits. Their bit definitions are shown in Figure 14-7, Figure 14-8, and Figure 14-9. The fields are described in Table 14-3, which applies to all DDR*n* registers.

The DDR*n* registers are read/write. At reset, all bits in the DDR*n* registers are cleared to 0s.

Setting any bit in a DDR*n* register configures the corresponding port *n* pin as an output. Clearing any bit in a DDR*n* register configures the corresponding pin as an input.

IPSBAR 0x10_0022 (DDRAN) Access: User read/write
Offsets: 0x10_002C (DDRDD)
0x10_002E (DDRGP)

	7	6	5	4	3	2	1	0
R W	DDR*n*7	DDR*n*6	DDR*n*5	DDR*n*4	DDR*n*3	DDR*n*2	DDR*n*1	DDR*n*0
Reset:	0	0	0	0	0	0	0	0

Figure 14-6. Port Data Direction Registers with Bits 7:0 Implemented (DDRAN, DDRDD, DDRGP)

IPSBAR 0x10_0023 (DDRAS) Access: User read/write
Offsets: 0x10_0026 (DDRTA)
0x10_0027 (DDRTC)
0x10_0028 (DDRTD)
0x10_0029 (DDRUA)
0x10_002A (DDRUB)
0x10_002B (DDRUC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	DDR*n*3	DDR*n*2	DDR*n*1	DDR*n*0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-7. Port Data Direction Registers with Bits 3:0 Implemented (DDRAS, DDRTA, DDRTC, DDRTD, DDRUA, DDRUB, DDRUC)

IPSBAR 0x10_0024 (DDRQS) Access: User read/write
Offsets: 0x10_002D (DDRLD)

	7	6	5	4	3	2	1	0
R	0	DDR*n*6	DDR*n*5	DDR*n*4	DDR*n*3	DDR*n*2	DDR*n*1	DDR*n*0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-8. Port Data Direction Registers with Bits 6:0 Implemented (DDRQS, DDRLD)

IPSBAR Access: User read/write
Offset: 0x10_0020 (DDRNQ)

	7	6	5	4	3	2	1	0
R	DDR*n*7	DDR*n*6	DDR*n*5	DDR*n*4	DDR*n*3	DDR*n*2	DDR*n*1	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-9. Port NQ Data Direction Register (DDRNQ)

Table 14-3. DDR*n* Field Descriptions

Field	Description
DDR*nx*	Sets data direction for port *nx* pin when the port is configured as a digital output. 1 DDR*nx* is configured as an output 0 DDR*nx* is configured as an input

14.6.3 Port Pin Data/Set Data Registers (SET*n*)

The SET*n* registers reflect the current pin states and control the setting of output pins when the pin is configured for digital I/O.

The SET*n* registers with a full 8-bit implementation are shown in Figure 14-10. The remaining SET*n* registers use fewer than eight bits. Their bit definitions are shown in Figure 14-11, Figure 14-12, and Figure 14-13. The fields are described in Table 14-4, which applies to all SET*n* registers.

The SET*n* registers are read/write. At reset, the bits in the SET*n* registers are set to the current pin states.

Reading a SET*n* register returns the current state of the port *n* pins.

Writing 1s to a SET*n* register sets the corresponding bits in the PORT*n* register. Writing 0s has no effect.

IPSBAR 0x10_003A (SETAN) — Access: User read/write
Offsets: 0x10_0044 (SETDD)
0x10_0046 (SETGP)

	7	6	5	4	3	2	1	0
R W	SET*n*7	SET*n*6	SET*n*5	SET*n*4	SET*n*3	SET*n*2	SET*n*1	SET*n*0
Reset:	1	1	1	1	1	1	1	1

Figure 14-10. Port Pin Data/Set Data Registers with Bits 7:0 Implemented (SETAN, SETDD, SETGP)

IPSBAR 0x10_003B (SETAS) — Access: User read/write
Offsets: 0x10_003E (SETTA)
0x10_003F (SETTC)
0x10_0040 (SETTD)
0x10_0041 (SETUA)
0x10_0042 (SETUB)
0x10_0043 (SETUC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	SET*n*3	SET*n*2	SET*n*1	SET*n*0
W								
Reset:	0	0	0	0	1	1	1	1

Figure 14-11. Port Pin Data/Set Data Registers with Bits 3:0 Implemented (SETAS, SETTA, SETTC, SETTD, SETUA, SETUB, SETUC)

IPSBAR 0x10_003C (SETQS) Access: User read/write
Offsets: 0x10_0045 (SETLD)

	7	6	5	4	3	2	1	0
R	0	SET*n*6	SET*n*5	SET*n*4	SET*n*3	SET*n*2	SET*n*1	SET*n*0
W								
Reset:	0	1	1	1	1	1	1	1

Figure 14-12. Port Pin Data/Set Data Registers with Bits 6:0 Implemented (SETQS, SETLD)

IPSBAR Access: User read/write
Offset: 0x10_0038 (SETNQ)

	7	6	5	4	3	2	1	0
R	SET*n*7	SET*n*6	SET*n*5	SET*n*4	SET*n*3	SET*n*2	SET*n*1	0
W								
Reset:	1	1	1	1	1	1	1	0

Figure 14-13. Port NQ Pin Data/Set Data Register (SETNQ)

Table 14-4. SET*n* Field Descriptions

Field	Description
SET*n*x	Port *n* pin data/set data bits. 1 Port *n* pin *x* state is 1 (read); writing a 1 sets the corresponding bit to 1 0 Port *n* pin x state is 0 (read)

14.6.4 Port Clear Output Data Registers (CLR*n*)

Writing 0s to a CLR*n* register clears the corresponding bits in the PORT*n* register. Writing 1s has no effect. Reading the CLR*n* register returns 0s.

The CLR*n* registers with a full 8-bit implementation are shown in Figure 14-14. The remaining DDR*n* registers use fewer than eight bits. Their bit definitions are shown in Figure 14-15, Figure 14-16, and Figure 14-17. The fields are described in Table 14-5, which applies to all CLR*n* registers.

The CLR*n* registers are read/write.

IPSBAR 0x10_005C (CLRDD) Access: User read/write
Offsets: 0x10_0052 (CLRAN)
0x10_005E (CLRGP)

	7	6	5	4	3	2	1	0
R W	CLR*n*7	CLR*n*6	CLR*n*5	CLR*n*4	CLR*n*3	CLR*n*2	CLR*n*1	CLR*n*0
Reset:	0	0	0	0	0	0	0	0

Figure 14-14. Port Clear Output Data Registers with Bits 7:0 Implemented (CLRDD, CLRAN, CLRGP)

IPSBAR 0x10_0053 (CLRAS) Access: User read/write
Offsets: 0x10_0056 (CLRTA)
0x10_0057 (CLRTC)
0x10_0058 (CLRTD)
0x10_0059 (CLRUA)
0x10_005A (CLRUB)
0x10_005B (CLRUC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	CLR*n*3	CLR*n*2	CLR*n*1	CLR*n*0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-15. Port Clear Output Data Registers with Bits 3:0 Implemented (CLRAS, CLRTA, CLRTC, CLRTD, CLRUA, CLRUB, CLRUC)

IPSBAR 0x10_0054 (CLRQS) Access: User read/write
Offsets: 0x10_005D (CLRLD)

	7	6	5	4	3	2	1	0
R	0	CLR*n*6	CLR*n*5	CLR*n*4	CLR*n*3	CLR*n*2	CLR*n*1	CLR*n*0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-16. Port Clear Output Data Registers with Bits 6:0 Implemented (CLRQS, CLRLD)

IPSBAR Access: User read/write
Offset: 0x10_0050 (CLRNQ)

	7	6	5	4	3	2	1	0
R	CLR*n*7	CLR*n*6	CLR*n*5	CLR*n*4	CLR*n*3	CLR*n*2	CLR*n*1	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-17. Port NQ Clear Output Data Register (CLRNQ)

Table 14-5. CLR*n* Field Descriptions

Field	Description
CLR*nx*	Port *n*x pin data/set data bits. 1 Never returned for reads; no effect for writes 0 Always returned for reads; clears corresponding port *n*x bit for writes

14.6.5 Pin Assignment Registers

All pin assignment registers are read/write. Refer to Table 2-1 for the different functions assignable to each pin.

Some signals can be assigned to different pins (see Table 2-1). However, a signal should not be assigned to more than one pin at the same time. If a signal is assigned to two or more pins simultaneously, the result is undefined.

14.6.5.1 Dual Function Pin Assignment Registers

The dual function pin assignment registers allow each pin controlled by each register bit to be configured for the primary function or the GPIO function. The fields are described in Table 14-6, which applies to all dual-function registers.

IPSBAR 0x10_006A (PANPAR) Access: User read/write
Offsets: 0x10_0074 (PDDPAR)
0x10_0076 (PGPPAR)

	7	6	5	4	3	2	1	0
R W	P*n*PAR7	P*n*PAR6	P*n*PAR5	P*n*PAR4	P*n*PAR3	P*n*PAR2	P*n*PAR1	P*n*PAR0
Reset:	0	0	0	0	0	0	0	0

Figure 14-18. Dual-Function Pin Assignment Registers with Bits 7:0 Implemented (PANPAR, PDDPAR, PGPPAR)

IPSBAR Access: User read/write
Offset: 0x10_0075 (PLDPAR)

	7	6	5	4	3	2	1	0
R	0	P*n*PAR6	P*n*PAR5	P*n*PAR4	P*n*PAR3	P*n*PAR2	P*n*PAR1	P*n*PAR0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-19. Port LD Pin Assignment Register (PLDPAR)

IPSBAR 0x10_0070 (PTDPAR) Access: User read/write
Offsets: 0x10_0073 (PUCPAR)

	7	6	5	4	3	2	1	0
R	0	0	0	0	P*n*PAR3	P*n*PAR2	P*n*PAR1	P*n*PAR0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-20. Dual-Function Pin Assignment Registers with Bits 3:0 Implemented (PTDPAR, PUCPAR)

Table 14-6. Dual-Function P*n*PAR Field Descriptions

Field	Description
P*n*PARx	P*n*PARx pin assignment register bits. 1 Pin assumes its primary function 0 Pin assumes its GPIO function

14.6.5.2 Quad-Function Pin Assignment Registers

The quad function pin assignment registers allow each pin controlled by each register bit to be configured for the primary, alternate 1 (secondary), alternate 2 (tertiary), and GPIO (quaternary) functions. The fields are described in Table 14-7, which applies to all quad-function registers.

IPSBAR Access: User read/write
Offset: 0x10_0068 (PNQPAR)

	15	14	13	12	11	10	9	8
R	P*n*PAR7		P*n*PAR6		P**n**PAR5		P*n*PAR4	
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	P*n*PAR3		P*n*PAR2		P*n*PAR1		0	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 14-21. Port NQ Pin Assignment Register (PNQPAR)

IPSBAR
Offset: 0x10_006C (PQSPAR)

Access: User read/write

	15	14	13	12	11	10	9	8
R	0	0	P*n*PAR6		P**n**PAR5		P*n*PAR4	
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	P*n*PAR3		P*n*PAR2		P*n*PAR1		P*n*PAR0	
W								
Reset	0	0	0	0	0	0	0	0

Figure 14-22. Port QS Pin Assignment Register (PQSPAR)

IPSBAR 0x10_006B (PASPAR)
Offsets: 0x10_006E (PTAPAR)
0x10_006F (PTCPAR)
0x10_0071 (PUAPAR)
0x10_0072 (PUBPAR)

Access: User read/write

	7	6	5	4	3	2	1	0
R	P*n*PAR3		P*n*PAR2		P*n*PAR1		P*n*PAR0	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-23. Quad-Function Pin Assignment Registers with Bits 7:0 Implemented (PASPAR, PTAPAR, PTCPAR, PUAPAR, PUBPAR)

Table 14-7. Quad-Function P*n*PAR Field Descriptions

Field	Description
P*n*PARx	P*n*PARx pin assignment register bits. 00 Pin assumes its GPIO function 01 Pin assumes its primary function 10 Pin assumes its alternate 1 function 11 Pin assumes its alternate 2 function

14.6.5.3 Pin Wired OR Register (PWOR)

The Pin Wired OR register (PWOR) is read/write and each bit resets to logic 0. Refer to Table 2-1 for details of which PWOR bit controls which pin.

IPSBAR Access: User read/write

Offset: 0x10_0078 (PWOR)

	15	14	13	12	11	10	9	8
R W	PWOR15	PWOR14	PWOR13	PWOR12	PWOR11	PWOR10	PWOR9	PWOR8
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R W	PWOR7	PWOR6	PWOR5	PWOR4	PWOR3	PWOR2	PWOR1	PWOR0
Reset	0	0	0	0	0	0	0	0

Figure 14-24. Pin Wired OR Register (PWOR)

Table 14-8. PWOR Field Descriptions

Field	Description
PWOR*n*	Wired OR configuration bits. 0 Configures the selected bit for normal operation 1 Configures the selected bit for wired OR operation

14.6.5.4 Pin Drive Strength Registers

The pin drive strength registers (PDSR0 and PDSR1) are read/write, and each bit resets to logic 0 in single chip mode (MCF52235 default) and logic 1 in EzPort and FAST mode.

IPSBAR Access: User read/write

Offset: 0x10_007A (PDSR1)

	15	14	13	12	11	10	9	8
R W	PDSR47	PDSR46	PDSR45	PDSR44	PDSR43	PDSR42	PDSR41	PDSR40
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

	7	6	5	4	3	2	1	0
R W	PDSR39	PDSR38	PDSR37	PDSR36	PDSR35	PDSR34	PDSR33	PDSR32
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

1) Each bit resets to logic 0 in Single Chip mode and logic 1 in EzPort/FAST mode.

Figure 14-25. Pin Drive Strength Register 1 (PDSR1)

IPSBAR Access: User read/write

Offset: 0x10_007C (PDSR0)

	31	30	29	28	27	26	25	24
R W	PDSR31	PDSR30	PDSR29	PDSR28	PDSR27	PDSR26	PDSR25	PDSR24
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

	23	22	21	20	19	18	17	16
R W	PDSR23	PDSR22	PDSR21	PDSR20	PDSR19	PDSR18	PDSR17	PDSR16
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

	15	14	13	12	11	10	9	8
R W	PDSR15	PDSR14	PDSR13	PDSR12	PDSR11	PDSR10	PDSR9	PDSR8
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

	7	6	5	4	3	2	1	0
R W	PDSR7	PDSR6	PDSR5	PDSR4	PDSR3	PDSR2	PDSR1	PDSR0
Reset	See footnote 1		See footnote 1		See footnote 1		See footnote 1	

1) Each bit resets to logic 0 in Single Chip mode and logic 1 in EzPort/FAST mode.

Figure 14-26. Pin Drive Strength Register 0 (PDSR0)

Table 14-9. PDSR0 and PDSR1 Field Descriptions

Field	Description
PDSR*n*	Pin drive strength register control bits. 0 Pin is configured for low drive strength (2 mA) 1 Pin is configured for high drive strength (10 mA)

14.7 Ports Interrupts

The ports module does not generate interrupt requests.

Chapter 15
Interrupt Controller Module

This section details the functionality for the interrupt controller. The general features of the interrupt controller include:

- Interrupt sources
  - fully-programmable interrupt sources (of which some are reserved)
  - 7 fixed-level interrupt sources
- Each of the sources has a unique interrupt control register (ICR*nx*) to define the software-assigned levels and priorities within the level
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports hardware and software interrupt acknowledge cycles
- Wake-up signal from low-power stop modes

The 50 fully-programmable and seven fixed-level interrupt sources for the interrupt controller manage the complete set of interrupt sources from all of the modules on the device. This section describes how the interrupt sources are mapped to the interrupt controller logic and how interrupts are serviced.

15.1 68K/ColdFire Interrupt Architecture Overview

Before continuing with the specifics of the interrupt controller, a brief review of the interrupt architecture of the 68K/ColdFire family is appropriate.

The interrupt architecture of ColdFire is exactly the same as the M68000 family, where there is a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing 7 levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. The processor samples for active interrupt requests once per instruction by comparing the encoded priority level against a 3-bit interrupt mask value (I) contained in bits 10:8 of the core's status register (SR). If the priority level is greater than the SR[I] field at the sample point, the processor suspends normal instruction execution and initiates interrupt exception processing. Level 7 interrupts are treated as non-maskable and edge-sensitive within the processor, while levels 1–6 are treated as level-sensitive and may be masked depending on the value of the SR[I] field. For correct operation, ColdFire requires that the interrupt source, after asserted, remains asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU enters supervisor mode, disables trace mode, and then fetches an 8-bit vector from the interrupt controller. This byte-sized operand fetch is known as the interrupt acknowledge (IACK) cycle, with the ColdFire implementation using a special encoding of the transfer type and transfer modifier attributes to distinguish this data fetch from a normal memory access. The fetched data provides an index into the exception vector table, which contains 256 addresses, each pointing

to the beginning of a specific exception service routine. In particular, vectors 64–255 of the exception vector table are reserved for user interrupt service routines. The first 64 exception vectors are reserved for the processor to manage reset, error conditions (access, address), arithmetic faults, system calls, etc. After the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are 2 longwords in length and contain 32 bits of vector and status register data, along with the 32-bit program counter value of the instruction that was interrupted (see Section 3.3.3.1, “Exception Stack Frame Definition,” for more information on the stack frame format).

After the exception stack frame is stored in memory, the processor accesses the 32-bit pointer from the exception vector table using the vector number as the offset, and then jumps to that address to begin execution of the service routine. After the status register is stored in the exception stack frame, the SR[I] mask field is set to the level of the interrupt being acknowledged, effectively masking that level and all lower values while in the service routine.

For this device, the processing of the interrupt acknowledge cycle is fundamentally different than previous 68K/ColdFire cores. In the new approach, all IACK cycles are directly managed by the interrupt controller, so the requesting peripheral device is not accessed during IACK. As a result, the interrupt request must be explicitly cleared in the peripheral during the interrupt service routine. For more information, see Section 15.1.1.3, “Interrupt Vector Determination.”

Unlike the M68000 family, all ColdFire processors guarantee that the first instruction of the service routine is executed before sampling for interrupts is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled if required.

During the execution of the service routine, the appropriate actions must be performed on the peripheral to negate the interrupt request.

For more information on exception processing, perform a keyword search at http://www.freescale.com/coldfire for “CFPRM,” then click the *CFPRM: ColdFire Family Programmer’s Reference Manual* link in the list of search results.

15.1.1 Interrupt Controller Theory of Operation

To support the interrupt architecture of the 68K/ColdFire programming model, the combined 63 interrupt sources are organized as 7 levels, with each level supporting up to 9 prioritized requests. Consider the priority structure within a single interrupt level (from highest to lowest priority) as shown in Table 15-1.

Table 15-1. Interrupt Priority Within a Level

ICR[2:0]	Priority	Interrupt Sources
111	7 (Highest)	8–63
110	6	8–63
101	5	8–63
100	4	8–63
—	Fixed Midpoint Priority	1–7

Table 15-1. Interrupt Priority Within a Level (continued)

ICR[2:0]	Priority	Interrupt Sources
011	3	8–63
010	2	8–63
001	1	8–63
000	0 (Lowest)	8–63

The level and priority is fully programmable for all sources except interrupt sources 1–7. Interrupt source 1–7 (from the edge port module) are fixed at the corresponding level's midpoint priority. Thus, a maximum of 8 fully-programmable interrupt sources are mapped into a single interrupt level. The fixed interrupt source is hardwired to the given level and represents the mid-point of the priority within the level. For the fully-programmable interrupt sources, the 3-bit level and the 3-bit priority within the level are defined in the 8-bit interrupt control register (ICR*nx*).

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector determination during IACK

15.1.1.1 Interrupt Recognition

The interrupt controller continuously examines the request sources and the interrupt mask register to determine if there are active requests. This is the recognition phase.

15.1.1.2 Interrupt Prioritization

As an active request is detected, it is translated into the programmed interrupt level, and the resulting 7-bit decoded priority level (IRQ[7:1]) is driven out of the interrupt controller.

15.1.1.3 Interrupt Vector Determination

After the core has sampled for pending interrupts and begun interrupt exception processing, it generates an interrupt acknowledge (IACK) cycle. The IACK transfer is treated as a memory-mapped byte read by the processor and routed to the appropriate interrupt controller. Next, the interrupt controller extracts the level being acknowledged from address bits 4:2, determines the highest priority interrupt request active for that level, and returns the 8-bit interrupt vector for that request to complete the cycle. The 8-bit interrupt vector is formed using the following algorithm:

```
vector_number = 64 + interrupt source number
```

Recall that vector numbers 0–63 are reserved for the ColdFire processor and its internal exceptions. Thus, the mapping of bit positions to vector numbers is as follows:

```
if interrupt source 1 is active and acknowledged,       then Vector number =  65
if interrupt source 2 is active and acknowledged,       then Vector number =  66
```

```
...
if interrupt source 8 is active and acknowledged,        then Vector number =  72
if interrupt source 9 is active and acknowledged,        then Vector number =  73
...
if interrupt source 62 is active and acknowledged,       then Vector number = 126
```

The net effect is a fixed mapping between the bit position within the source to the actual interrupt vector number.

If there is no active interrupt source for the given level, a special spurious interrupt vector (vector number = 24) is returned. It is the responsibility of the service routine to manage this error situation.

This protocol implies the interrupting peripheral is not accessed during the acknowledge cycle because the interrupt controller completely services the acknowledge. This means the interrupt source must be explicitly disabled in the interrupt service routine. This design provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

15.2 Memory Map

The register programming model for the interrupt controllers is memory-mapped to a 256-byte space. In the following discussion, there are a number of program-visible registers greater than 32 bits. For these control fields, the physical register is partitioned into two 32-bit values: a register high (the upper longword, represented by an appended H) and a register low (the lower longword, represented by an appended L).

The registers and their locations are defined in Table 15-3. The register names include the (zero-based) interrupt controller number *n*, which is useful in devices with multiple controllers. This device has only one interrupt controller; hence, *n* = 0.

Table 15-2. Interrupt Controller Base Addresses

Interrupt Controller Number	Base Address
INTC0	IPSBAR + 0xC00
Global IACK Registers Space[1]	IPSBAR + 0xF00

[1] This address space only contains the L1ACK-L7IACK registers. See Section 15.3.7, “Software and Level m IACK Registers (SWIACKn, LmIACKn)" for more information

Table 15-3. Interrupt Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
Interrupt Controller 0					
0x00_0C00	Interrupt Pending Register High (IPRH0)	32	R	0x0000_0000	15.3.1/15-5
0x00_0C04	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	15.3.1/15-5
0x00_0C08	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	15.3.2/15-6
0x00_0C0C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	15.3.2/15-6

Table 15-3. Interrupt Controller Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0x00_0C10	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	15.3.3/15-8
0x00_0C14	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	15.3.3/15-8
0x00_0C18	Interrupt Request Level Register (IRLR0)	8	R/W	0x00	15.3.4/15-8
0x00_0C19	Interrupt Acknowledge Level and Priority Register (IACKLPR0)	8	W	0x00	15.3.5/15-9
0x00_0C40 + *n* (*n*=0:63)	Interrupt Control Registers (ICR0*n*)	8	W	0x00	15.3.6/15-10
0x00_0CE0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	15.3.7/15-11
0x00_0CE0 + 4*n* (*n*=1:7)	Level *m* Interrupt Acknowledge Registers (L*m*IACK0)	8	R	0x00	15.3.7/15-11
Global IACK Registers					
0x00_0F0E0 + 4*n* (*n*=1:7)	Global Level *m* Interrupt Acknowledge Registers (GL*m*IACK)	8	R	0x00	15.3.8/15-12

15.3 Register Descriptions

The interrupt controller registers are described in the following sections.

15.3.1 Interrupt Pending Registers (IPRH*n*, IPRL*n*)

The IPRH*n* and IPRL*n* registers, Figure 15-1 and Figure 15-2, each 32 bits, provide a bit map for each interrupt request to indicate if there is an active request (1 = active request, 0 = no request) for the given source. The state of the interrupt mask register does not affect the IPR*n*. The IPR*n* is cleared by reset. The IPR*n* is a read-only register, so any attempted write to this register is ignored. Bit 0 is not implemented and reads as a zero.

IPSBAR Offset: 0x00_0C00 (IPRH0) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT[63:32]																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-1. Interrupt Pending Register High (IPRH*n*)

Table 15-4. IPRH*n* Field Descriptions

Field	Description
31–0 INT	Interrupt pending. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRH*n* samples the signal generated by the interrupting source. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

IPSBAR Offset: 0x00_0C04 (IPRL0) Access: User read-only

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

Figure 15-2. Interrupt Pending Register Low (IPRL*n*)

Table 15-5. IPRL*n* Field Descriptions

Field	Description
31–1 INT	Interrupt Pending. Each bit corresponds to an interrupt source. The corresponding IMRL*n* bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRL*n* samples the signal generated by the interrupting source. The corresponding IPRL*n* bit reflects the state of the interrupt signal even if the corresponding IMRL*n* bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending
0	Reserved, must be cleared.

15.3.2 Interrupt Mask Registers (IMRH*n*, IMRL*n*)

The IMRH*n* and IMRL*n* registers are each 32 bits and provide a bit map for each interrupt to allow the request to be disabled (1 = disable the request, 0 = enable the request). The IMR*n* is set to all ones by reset, disabling all interrupt requests. The IMR*n* can be read and written. A write that sets bit 0 of the IMRL*n* forces the other 63 bits to be set, disabling all interrupt sources, and providing a global mask-all capability.

IPSBAR Offset: 0x00_0C08 (IMRH0) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT_MASK[63:32]																															
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 15-3. Interrupt Mask Register High (IMRH*n*)

Table 15-6. IMRH*n* Field Descriptions

Field	Description
31–0 INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

IPSBAR Offset: 0x00_0C0C (IMRL0) Access: User read/write

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

Figure 15-4. Interrupt Mask Register Low (IMRL*n*)

Table 15-7. IMRL*n* Field Descriptions

Field	Description
31–1 INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRL*n* bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRL*n* bit reflects the state of the interrupt signal even if the corresponding IMRL*n* bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked
0 MASKALL	Mask all interrupts. Setting this bit forces the other 63 bits of the IMRH*n* and IMRL*n* to ones, disabling all interrupt sources, and providing a global mask-all capability.

NOTE

A spurious interrupt may occur if an interrupt source is being masked in the interrupt controller mask register (IMR) or a module's interrupt mask register while the interrupt mask in the status register (SR[I]) is set to a value lower than the interrupt's level. This is because by the time the status register acknowledges this interrupt, the interrupt has been masked. A spurious interrupt is generated because the CPU cannot determine the interrupt source.

To avoid this situation for interrupts sources with levels 1–6, first write a higher level interrupt mask to the status register, before setting the mask in the IMR or the module's interrupt mask register. After the mask is set, return the interrupt mask in the status register to its previous value. Because level 7 interrupts cannot be disabled in the status register prior to masking, use of the IMR or module interrupt mask registers to disable level 7 interrupts is not recommended.

15.3.3 Interrupt Force Registers (INTFRCH*n*, INTFRCL*n*)

The INTFRCH*n* and INTFRCL*n* registers, each 32 bits, provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (1 = force request, 0 = negate request) in the appropriate INTFRC*n* register. The assertion of an interrupt request via the INTFRC*n* register is not affected by the interrupt mask register. The INTFRC*n* register is cleared by reset.

IPSBAR Offset: 0x00_0C10 (INTFRCH0) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INTFRCH[63:32]																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-5. Interrupt Force Register High (INTFRCH*n*)

Table 15-8. INTFRCH*n* Field Descriptions

Field	Description
31–0 INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source

IPSBAR Offset: 0x00_0C14 (INTFRCL0) Access: User read/write

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

Figure 15-6. Interrupt Force Register Low (INTFRCL*n*)

Table 15-9. INTFRCL*n* Field Descriptions

Field	Description
31–1 INTFRCL	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source
0	Reserved, must be cleared.

15.3.4 Interrupt Request Level Register (IRLR*n*)

This 7-bit register is updated each machine cycle and represents the current interrupt requests for each interrupt level, where bit 7 corresponds to level 7, bit 6 to level 6, etc.

IPSBAR Offset: 0x00_0C18 (IRLR0) Access: Read-only

	7	6	5	4	3	2	1	0
R	IRQ[7:1]							0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 15-7. Interrupt Request Level Register (IRLR*n*)

Table 15-10. IRLR*n* Field Descriptions

Field	Description
7–1 IRQ	Interrupt requests. Represents the prioritized active interrupts for each level. 0 There are no active interrupts at this level 1 There is an active interrupt at this level
0	Reserved

15.3.5 Interrupt Acknowledge Level and Priority Register (IACKLPR*n*)

Each time an IACK is performed, the interrupt controller responds with the vector number of the highest priority source within the level being acknowledged. In addition to providing the vector number directly for the byte-sized IACK read, this 8-bit register is also loaded with information about the interrupt level and priority being acknowledged. This register provides the association between the acknowledged physical interrupt request number and the programmed interrupt level/priority.

IPSBAR Offset: 0x00_0C19 (IACKLPR0) Access: Read-only

	7	6	5	4	3	2	1	0
R	0	LEVEL			PRI			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 15-8. IACK Level and Priority Register (IACKLPR*n*)

Table 15-11. IACKLPR*n* Field Descriptions

Field	Description
7	Reserved

Table 15-11. IACKLPR*n* Field Descriptions (continued)

Field	Description
6–4 LEVEL	Interrupt level. Represents the interrupt level of the interrupt currently being acknowledged.
3–0 PRI	Interrupt Priority. Represents the priority within the interrupt level of the interrupt currently being acknowledged. 0 Priority 0 1 Priority 1 2 Priority 2 3 Priority 3 4 Priority 4 5 Priority 5 6 Priority 6 7 Priority 7 8 Mid-point priority associated with the fixed level interrupts only

15.3.6 Interrupt Control Registers (ICR*nx*)

Each ICR*nx*, where *x* = 1, 2,..., 63, specifies the interrupt level (1–7) and the priority within the level (0–7). As shown in Table 15-12, all ICR*nx* registers can be read, but only ICR*n*8 through ICR*n*63 can be written. Registers ICR*n*1 through ICR*n*7 are read-only because the interrupt levels for IRQ1 through IRQ7 are hard-coded to their respective source numbers (see Section 15.1.1, “Interrupt Controller Theory of Operation”). The registers are described in Figure 15-9 and Table 15-13.

It is the responsibility of the software to program the ICR*nx* registers with unique and non-overlapping level and priority definitions. Failure to program the ICR*nx* registers in this manner can result in undefined behavior. If a specific interrupt request is completely unused, the ICR*nx* value can remain in its reset (and disabled) state.

Table 15-12. ICR*nx* Register Accessibility

Registers	Access
ICR*n*1 – ICR*n*7	Read-only
ICR*n*8 – ICR*n*63	Read / write

IPSBAR Access: R/W (Read only for ICR*n*1-ICR*n*7)

Offsets: See Table 15-2 for register offsets (ICR*nx*)

	7	6	5	4	3	2	1	0
R	0	0	IL			IP		
W								
Reset:	0	0	0	0	0	0	0	0

Note: It is the responsibility of the software to program the ICR*nx* registers with unique and non-overlapping level and priority definitions. Failure to program the ICR*nx* registers in this manner can result in undefined behavior. If a specific interrupt request is completely unused, the ICR*nx* value can remain in its reset (and disabled) state.

Figure 15-9. Interrupt Control Register (ICR*nx*)

Table 15-13. ICR*nx* Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–3 IL	Interrupt level. Indicates the interrupt level assigned to each interrupt input.
2–0 IP	Interrupt priority. Indicates the interrupt priority for internal modules within the interrupt-level assignment. 0x0 represents the lowest priority and 0x7 represents the highest. For the fixed level interrupt sources, the priority is fixed at the midpoint for the level, and the IP field always reads as 0x0.

15.3.7 Software and Level *m* IACK Registers (SWIACK*n*, L*m*IACK*n*)

The eight IACK registers can be explicitly addressed via the CPU, or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing. In either case, the interrupt controller's actions are very similar.

When a level-*m* IACK arrives in the interrupt controller, the controller examines all the currently-active level m interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle. In addition to providing the vector number, the interrupt controller also loads the level and priority number for the level into the IACKLPR register, where it may be retrieved later.

This interrupt controller design also supports the concept of a software IACK. A software IACK allows an interrupt service routine to determine if there are other pending interrupts so that the overhead associated with interrupt exception processing (including machine state save/restore functions) can be minimized. In general, the software IACK is performed near the end of an interrupt service routine, and if there are additional active interrupt sources, the current interrupt service routine (ISR) passes control to the appropriate service routine, but without taking another interrupt exception.

When the interrupt controller receives a software IACK read, it returns the vector number associated with the highest level, highest priority unmasked interrupt source for that interrupt controller. The IACKLPR register is also loaded as the software IACK is performed. If there are no active sources, the interrupt controller returns an all-zero vector as the operand. For this situation, the IACKLPR register is also cleared.

IPSBAR Offsets: See Table 15-2 for register offsets (SWIACK*n*, L*m*IACK*n*) Access: read-only

	7	6	5	4	3	2	1	0
R	VECTOR							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 15-10. Software and Level *m* IACK Registers (SWIACK*n*, L*m*IACK*n*)

Table 15-14. SWIACK*n* and L*m*IACK*n* Field Descriptions

Field	Description
7–0 VECTOR	Vector number. A read from the SWIACK register returns the vector number associated with the highest level, highest priority unmasked interrupt source. A read from one of the L*m*IACK registers returns the highest priority unmasked interrupt source within the level.

15.3.8 Global Level *m* IACK Registers (GL*m*IACK)

In addition to the software IACK registers (Section 15.3.7, “Software and Level m IACK Registers (SWIACKn, LmIACKn)”), there are global IACK registers, GL*m*IACK. (There is no global SWIACK register.) On devices with multiple interrupt controllers, a read from one of the GL*m*IACK registers returns the vector for the highest priority unmasked interrupt within a level for all interrupt controllers.

IPSBAR Offsets: See Table 15-2 for register offsets (GL*m*IACK) Access: read-only

	7	6	5	4	3	2	1	0
R	VECTOR							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 15-11. Global Level *m* IACK Registers (GL*m*IACK)

Table 15-15. GSWIACK and GL*m*IACK Field Descriptions

Field	Description
7–0 VECTOR	Vector number. A read from one of the L*m*IACK registers returns the vector for the highest priority unmasked interrupt within a level for all interrupt controllers. As implemented on the MCF52, these registers contain the same information as L*m*IACK.

15.3.8.1 Interrupt Sources

Table 15-6 lists the interrupt sources for each interrupt request line.

15.4 Low-Power Wakeup Operation

The system control module (SCM) contains an 8-bit low-power interrupt control register (LPICR) used explicitly for controlling the low-power stop mode. This register must explicitly be programmed by software to enter low-power mode.

The interrupt controller provides a special combinatorial logic path to provide a special wake-up signal to exit from the low-power stop mode. This special mode of operation works as follows:

1. LPICR[6:4] is loaded with the specified mask level while the core is in stop mode. LPICR[7] must be set to enable this mode of operation.

NOTE

The wakeup mask level taken from LPICR[6:4] is adjusted by hardware to allow a level 7 IRQ to generate a wakeup. That is, the wakeup mask value used by the interrupt controller must be in the range of 0–6.

2. The processor executes a STOP instruction which places it in stop mode. After the processor is stopped, each interrupt controller enables a special logic path that evaluates the incoming interrupt sources in a purely combinatorial path; that is, there are no clocked storage elements. If an active interrupt request is asserted and the resulting interrupt level is greater than the mask value contained in LPICR[6:4], then the interrupt controller asserts the wake-up output signal, which is routed to the SCM and PLL module to re-enable the device’s clock trees and resume processing.

Chapter 16
Edge Port Modules (EPORT*n*)

16.1 Introduction

Although this device has two edge port modules, the description included herein treats each module as a single entity. Pay particular attention to the note below, as the two modules are not completely identical. Specifically, edge port module 0 has seven interrupt inputs while module 1 contains eight.

The edge port module (EPORT) has up to eight interrupt pins, $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Each pin can be configured individually as a level-sensitive interrupt pin, an edge-detecting interrupt pin (rising edge, falling edge, or both), or a general-purpose input/output (I/O) pin.

NOTE

Not all EPORT signals may be output from the device. See Chapter 2, "Signal Descriptions," to determine which signals are available.

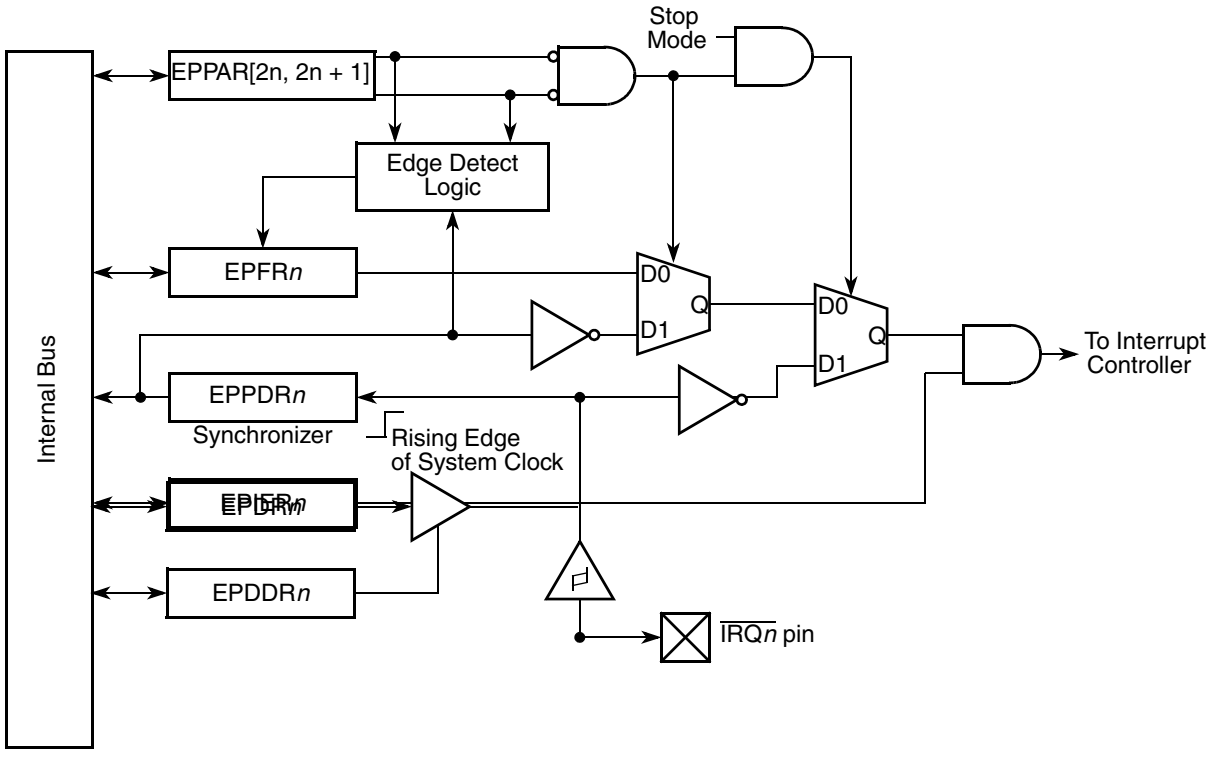


Figure 16-1. EPORT Block Diagram

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “General Purpose I/O Module”) prior to configuring the edge-port module.

16.2 Low-Power Mode Operation

This section describes the operation of the EPORT module in low-power modes. For more information on low-power modes, see Chapter 9, “Power Management”. Table 16-1 shows EPORT-module operation in low-power modes and describes how this module may exit each mode.

NOTE

The low-power control register (LPCR) in the system control module specifies the interrupt level at or above what is needed to bring the device out of a low-power mode.

Table 16-1. Edge Port Module Operation in Low-Power Modes

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any $\overline{\text{IRQ}}n$ interrupt at or above level in LPCR
Doze	Normal	Any $\overline{\text{IRQ}}n$ interrupt at or above level in LPCR
Stop	Level-sensing only	Any $\overline{\text{IRQ}}n$ interrupt set for level-sensing at or above level in LPCR. See note below.

In wait and doze modes, the EPORT module continues to operate as it does in run mode. It may be configured to exit the low-power modes by generating an interrupt request on a selected edge or a low level on an external pin. In stop mode, no clocks are available to perform the edge-detect function. Only the level-detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

NOTE

In stop mode, the input pin synchronizer is bypassed for the level-detect logic because no clocks are available.

16.3 Signal Descriptions

All EPORT pins default to general-purpose input pins at reset. The pin value is synchronized to the rising edge of CLKOUT when read from the EPORT pin data register (EPPDR). The values used in the edge/level detect logic are also synchronized to the rising edge of CLKOUT. These pins use Schmitt-triggered input buffers with built-in hysteresis designed to decrease the probability of generating false, edge-triggered interrupts for slow rising and falling input signals.

When a pin is configured as an output, it is driven to a state whose level is determined by the corresponding bit in the EPORT data register (EPDR). All bits in the EPDR are set at reset.

16.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to Table 16-2 for a description of the EPORT memory map.

Table 16-2. Edge Port Module Memory Map

IPSBAR Offset EPORT0 EPORT1	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers[1]					
0x13_0000 0x14_0000	EPORT Pin Assignment Register (EPPAR*n*)	16	R/W	0x0000	16.4.1/16-3
0x13_0002 0x14_0002	EPORT Data Direction Register (EPDDR*n*)	8	R/W	0x00	16.4.2/16-4
0x13_0003 0x14_0003	EPORT Interrupt Enable Register (EPIER*n*)	8	R/W	0x00	16.4.3/16-5
Supervisor/User Access Registers					
0x13_0004 0x14_0004	EPORT Data Register (EPDR*n*)	8	R/W	0xFF	16.4.4/16-5
0x13_0005 0x14_0005	EPORT Pin Data Register (EPPDR*n*)	8	R	See Section	16.4.5/16-5
0x13_0006 0x14_0006	EPORT Flag Register (EPFR*n*)	8	R/W	0x00	16.4.6/16-6

[1] User access to supervisor-only address locations have no effect and result in a bus error.

16.4.1 EPORT Pin Assignment Register (EPPAR)

The EPORT pin assignment register (EPPAR) controls the function of each pin individually.

IPSBAR 0x13_0000 (EPPAR0)
Offset: 0x14_0000 (EPPAR1)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EPPA7		EPPA6		EPPA5		EPPA4		EPPA3		EPPA2		EPPA1		EPPA0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-2. EPORT Pin Assignment Register (EPPAR)

Table 16-3. EPPAR Field Descriptions

Field	Description
15–0 EPPA*n*	EPORT pin assignment select fields. The read/write EPPA*n* fields configure EPORT pins for level detection and rising and/or falling edge detection. Pins configured as level-sensitive are active-low (logic 0 on the external pin represents a valid interrupt request). Level-sensitive interrupt inputs are not latched. To guarantee that a level-sensitive interrupt request is acknowledged, the interrupt source must keep the signal asserted until acknowledged by software. Level sensitivity must be selected to bring the device out of stop mode with an $\overline{\text{IRQ}}n$ interrupt. Pins configured as edge-triggered are latched and need not remain asserted for interrupt generation. A pin configured for edge detection can trigger an interrupt regardless of its configuration as input or output. Interrupt requests generated in the EPORT module can be masked by the interrupt controller module. EPPAR functionality is independent of the selected pin direction. Reset clears the EPPA*n* fields. 00 Pin $\overline{\text{IRQ}}n$ level-sensitive 01 Pin $\overline{\text{IRQ}}n$ rising edge triggered 10 Pin $\overline{\text{IRQ}}n$ falling edge triggered 11 Pin $\overline{\text{IRQ}}n$ falling edge and rising edge triggered

16.4.2 EPORT Data Direction Register (EPDDR)

The EPORT data direction register (EPDDR) controls the direction of each one of the pins individually.

IPSBAR 0x13_0002 (EPDDR0) Access: Supervisor read/write
Offset: 0x14_0002 (EPDDR1)

	7	6	5	4	3	2	1	0
R W	EPDD7	EPDD6	EPDD5	EPDD4	EPDD3	EPDD2	EPDD1	EPDD0
Reset	0	0	0	0	0	0	0	0

Figure 16-3. EPORT Data Direction Register (EPDDR)

Table 16-4. EPDDR Field Descriptions

Field	Description
7–0 EPDD*n*	Setting any bit in the EPDDR configures the corresponding pin as an output. Clearing any bit in EPDDR configures the corresponding pin as an input. Pin direction is independent of the level/edge detection configuration. Reset clears EPDD7–EPDD0. To use an EPORT pin as an external interrupt request source, its corresponding bit in EPDDR must be clear. Software can generate interrupt requests by programming the EPORT data register when the EPDDR selects output. 0 Corresponding EPORT pin configured as input 1 Corresponding EPORT pin configured as output

16.4.3 Edge Port Interrupt Enable Register (EPIER)

The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.

IPSBAR 0x13_0003 (EPIER0) Access: User read/write
Offset: 0x14_0003 (EPIER1)

	7	6	5	4	3	2	1	0
R W	EPIE7	EPIE6	EPIE5	EPIE4	EPIE3	EPIE2	EPIE1	EPIE0
Reset:	0	0	0	0	0	0	0	0

Figure 16-4. EPORT Port Interrupt Enable Register (EPIER)

Table 16-5. EPIER Field Descriptions

Field	Description
7–0 EPIE*n*	Edge port interrupt enable bits enable EPORT interrupt requests. If a bit in EPIER is set, EPORT generates an interrupt request when: • The corresponding bit in the EPORT flag register (EPFR) is set or later becomes set • The corresponding pin level is low and the pin is configured for level-sensitive operation Clearing a bit in EPIER negates any interrupt request from the corresponding EPORT pin. Reset clears EPIE7–EPIE0. 0 Interrupt requests from corresponding EPORT pin disabled 1 Interrupt requests from corresponding EPORT pin enabled

16.4.4 Edge Port Data Register (EPDR)

The EPORT data register (EPDR) holds the data to be driven to the pins.

IPSBAR 0x13_0004 (EPDR0) Access: User read/write
Offset: 0x14_0004 (EPDR1)

	7	6	5	4	3	2	1	0
R W	EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	EPD0
Reset:	1	1	1	1	1	1	1	1

Figure 16-5. EPORT Port Data Register (EPDR)

Table 16-6. EPDR Field Descriptions

Field	Description
7–0 EPD*n*	Edge port data bits. An internal register stores data written to EPDR; if any pin of the port is configured as an output, the bit stored for that pin is driven onto the pin. Reading EDPR returns the data stored in the register. Reset sets EPD7 – EPD0.

16.4.5 Edge Port Pin Data Register (EPPDR)

The EPORT pin data register (EPPDR) reflects the current state of the pins.

IPSBAR 0x13_0005 (EPPDR0) Access: User read-only
Offset: 0x14_0005 (EPPDR1)

	7	6	5	4	3	2	1	0
R	EPPD7	EPPD6	EPPD5	EPPD4	EPPD3	EPPD2	EPPD1	EPPD0
W								
Reset:	[$\overline{\text{IRQ7}}$]	[$\overline{\text{IRQ6}}$]	[$\overline{\text{IRQ5}}$]	[$\overline{\text{IRQ4}}$]	[$\overline{\text{IRQ3}}$]	[$\overline{\text{IRQ2}}$]	[$\overline{\text{IRQ1}}$]	[$\overline{\text{IRQ0}}$]

Figure 16-6. EPORT Port Pin Data Register (EPPDR)

Table 16-7. EPPDR Field Descriptions

Field	Description
7–0 EPPD*n*	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.

16.4.6 Edge Port Flag Register (EPFR)

The EPORT flag register (EPFR) individually latches EPORT edge events.

IPSBAR 0x13_0006 (EPFR0) Access: User read/write
Offset: 0x14_0006 (EPFR1)

	7	6	5	4	3	2	1	0
R	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 16-7. EPORT Port Flag Register (EPFR)

Table 16-8. EPFR Field Descriptions

Field	Description
7–0 EPF*n*	Edge port flag bits. When an EPORT pin is configured for edge triggering, its corresponding read/write bit in EPFR indicates that the selected edge has been detected. Reset clears EPF7 – EPF0. Bits in this register are set when the selected edge is detected on the corresponding pin. A bit remains set until cleared by writing a 1 to it. Writing 0 has no effect. If a pin is configured as level-sensitive (EPPAR*n* = 00), pin transitions do not affect this register. 0 Selected edge for $\overline{\text{IRQ}}n$ pin not detected 1 Selected edge for $\overline{\text{IRQ}}n$ pin detected

Chapter 17
ColdFire Flash Module (CFM)

17.1 Introduction

17.1.1 Overview

The ColdFire Flash Module (CFM) is a non-volatile memory (NVM) module for integration with a CPU. The CFM provides 256 Kbytes of 32-bit flash memory serving as electrically erasable and programmable, non-volatile memory. The flash memory is ideal for program and data storage for single-chip applications, allowing for field reprogramming without requiring external programming voltage sources. The MCF52235 incorporates SuperFlash® technology licensed from SST.

The common flash bus interface executes read operations to the flash memory using one or two system bus cycles to access each flash physical block, with access latency depending on the factory setting of the CLKSEL bits in the CFMCLKSEL register. Flash physical blocks are interleaved between odd and even addresses to form a flash logical block. Interleaving allows back-to-back read operations to the flash memory at an effective access rate of one system bus cycle per word after the initial two-cycle access if the CLKSEL bits are not set for single cycle access.

It is not possible to read from any flash logical block while the same logical block is being erased, programmed, or verified. Flash logical blocks are divided into multiple logical pages that can be erased separately. An erased bit reads 1 and a programmed bit reads 0.

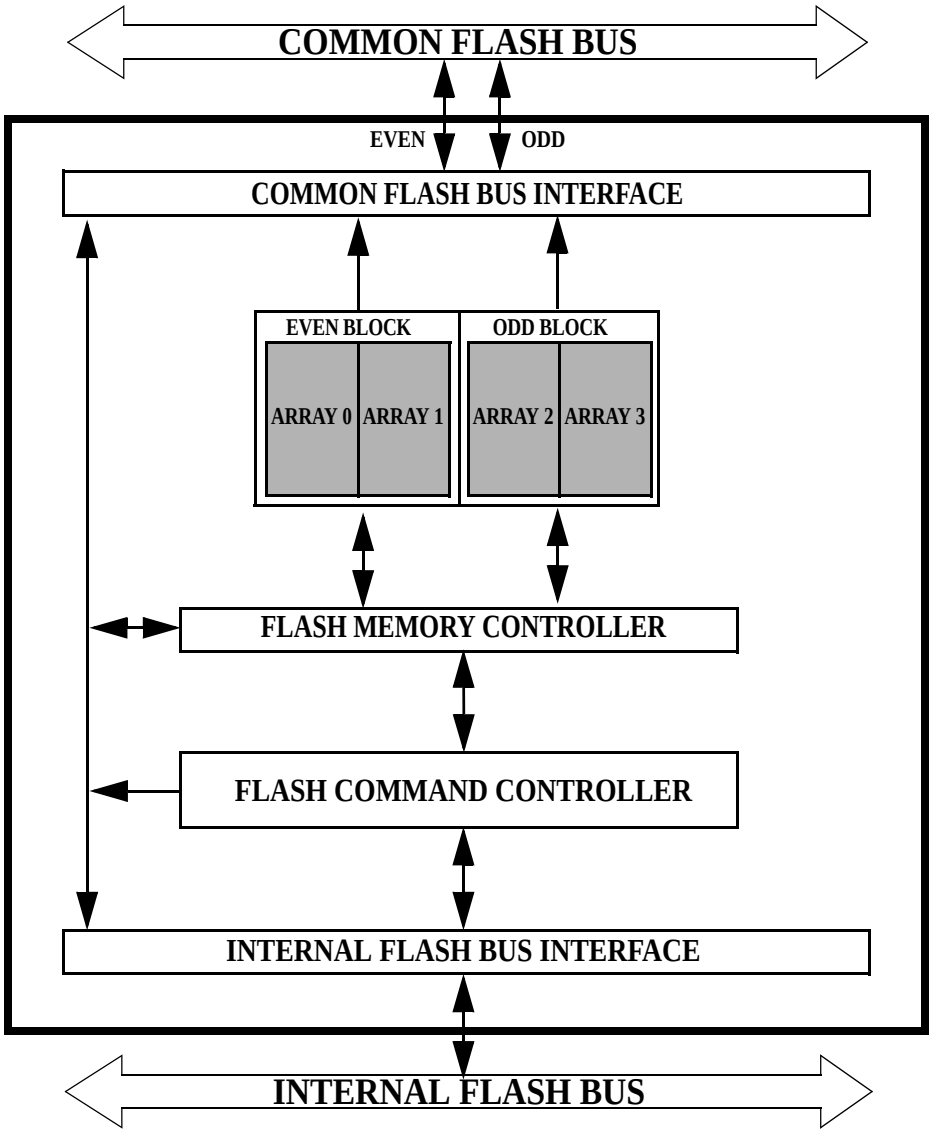


Figure 17-1. CFM Block Diagram

17.1.2 Features

- 256 Kbytes of 32-bit flash memory
- Automated program, erase, and verify operations
- Single power supply for program and erase operations
- Software programmable interrupts on command completion, access violations, or protection violations
- Fast page erase operation
- Fast word program operation

- Protection scheme to prevent accidental program or erase of flash memory
- Access restriction control for supervisor/user and data/instruction operations
- Security feature to prevent unauthorized access to the flash memory

17.2 External Signal Description

The CFM contains no signals that connect off-chip for the end customer.

17.3 Memory Map and Register Definition

This section describes the CFM memory map and registers.

17.3.1 Memory Map

The memory map for the CFM flash memory is shown in Figure 17-2. The starting address of the flash memory is determined by the flash array base address as defined by the system level configuration. The flash memory map shows how a pair of 32-bit flash physical blocks (even and odd) interleave every 4 bytes to form a contiguous memory space as follows:

Flash Block 0 includes byte addresses (PROGRAM_ARRAY_BASE+0x0000_0000) to (PROGRAM_ARRAY_BASE+0x0003_FFFF).

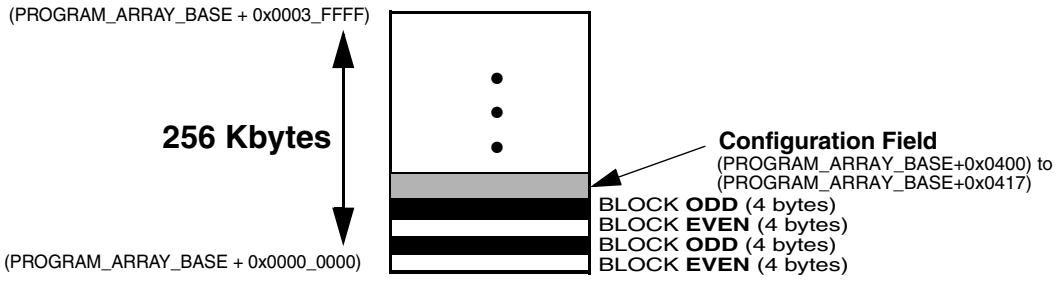


Figure 17-2. CFM Flash Memory Map

The CFM has hardware interlocks that protect data from accidental corruption using program or erase operations. A flexible scheme allows the protection of any combination of flash logical sectors as described in Section 17.3.3.4, “CFMPROT — CFM Protection Register.” A similar scheme is available to control supervisor/user and data/instruction access to these flash logical sectors.

Security information that allows the MCU to prevent intrusive access to the flash memory is stored in the flash configuration field. The flash configuration field is composed of 24 bytes of reserved memory space within the flash memory, which contains information that determines the CFM protection and access restriction scheme out of reset. A description of each byte found in the flash configuration field is given in Table 17-1.

Table 17-1. CFM Configuration Field

Address Offset (from PROGRAM_ARRAY_BASE)	Size (bytes)	Description	Factory Default
0x0400 - 0x0407	8	Backdoor Comparison Key	0xFFFF_FFFF_FFFF_FFFF
0x0408 - 0x040B	4	Flash Protection Bytes (see Section 17.3.3.4, “CFMPROT — CFM Protection Register”)	0xFFFF_FFFF
0x040C - 0x040F	4	Flash SUPV Access Bytes (see Section 17.3.3.5, “CFMSACC — CFM Supervisor Access Register”)	0xFFFF_FFFF
0x0410 - 0x0413	4	Flash DATA Access Bytes (see Section 17.3.3.6, “CFMDACC — CFM Data Access Register”)	0xFFFF_FFFF
0x0414 - 0x0417	4	Flash Security Word (see Section 17.3.3.3, “CFMSEC — CFM Security Register”)	0xFFFF_FFFF

17.3.2 Flash Base Address Register (FLASHBAR)

The configuration information in the flash base address register (FLASHBAR) controls the operation of the flash module.

- The FLASHBAR holds the base address of the flash. The MOVEC instruction provides write-only access to this register.
- The FLASHBAR can be read or written from the debug module in a similar manner.
- All undefined bits in the register are reserved. These bits are ignored during writes to the FLASHBAR, and return zeroes when read from the debug module.
- The FLASHBAR valid bit is programmed according to the chip mode selected at reset (see Chapter 12, “Chip Configuration Module (CCM)” for more details). All other bits are unaffected.

The FLASHBAR register contains several control fields. These fields are shown in Figure 17-3.

NOTE

The default value of the FLASHBAR is determined by the chip configuration selected at reset (see Chapter 12, “Chip Configuration Module (CCM)” for more information).

NOTE

Flash accesses (reads/writes) by a bus master other than the core, DMA controller, or writes to flash by the core during programming must use the backdoor flash address of IPSBAR plus an offset of 0x0400_0000. For example, for a DMA transfer from the first location of flash when IPSBAR remains at its default location of 0x4000_0000, the source register would be loaded with 0x4400_0000. Backdoor access to flash for reads can be made by the bus master, but it takes 2 cycles longer than a direct read of the flash if using its FLASHBAR address.

NOTE

The flash is marked as valid on reset based on the RCON (reset configuration) pin state. Flash space is valid on reset when booting in single chip mode (RCON pin asserted and D[26]/D[17]/D[16] set to 110), or when booting internally in master mode (RCON asserted and D[26]/D[17]/D[16] are set to 111 and D[18] and D[19] are set to 00). See Chapter 12, "Chip Configuration Module (CCM)" for more details. When the default reset configuration is not overridden, the MCU (by default) boots up in single chip mode and the flash space is marked as valid at address 0x0. The flash configuration field is checked during the reset sequence to see if the flash is secured. If it is, the device always boots from internal flash because it is marked as valid, regardless of what is done for chip configuration.

Address: CPU + 0x0C04 (FLASHBAR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA31	BA30	BA29	BA28	BA27	BA26	BA25	BA24	BA23	BA22	BA21	BA20	BA19	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	WP	0	AFS	C/I	SC	SD	UC	UD	V[1]
W																
Reset	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0

Figure 17-3. Flash Base Address Register (FLASHBAR)

[1] The reset value for the valid bit is determined by the chip mode selected at reset (see Chapter 12, "Chip Configuration Module (CCM)").

Table 17-2. FLASHBAR Field Descriptions

Bits	Description
31–19 BA[31:18]	Base address field. Defines the 0-modulo-512K base address of the flash module. By programming this field, the flash may be located on any 512Kbyte boundary within the processor's four gigabyte address space.
18–9	Reserved, should be cleared.
8 WP	Write protect. Read only. Allows only read accesses to the flash. This bit is always set and any attempted write access generates an access error exception to the ColdFire processor core. 0 Allows read and write accesses to the flash module 1 Allows only read accesses to the flash module
7	Reserved, should be cleared.
6 AFS	Address fetch speculation. Performance enhancement to generate speculative flash accesses. to reduce the effective flash access time from the actual two-cycle array time to a smaller number approaching one cycle. 0 Speculation enabled 1 Disable speculation

Table 17-2. FLASHBAR Field Descriptions (continued)

Bits	Description
5–1 C/I, SC, SD, UC, UD	Address space masks (AS*n*). These five bit fields allow certain types of accesses to be masked, or inhibited from accessing the flash module. The address space mask bits are: C/I CPU space/interrupt acknowledge cycle mask SC Supervisor code address space mask SD Supervisor data address space mask UC User code address space mask UD User data address space mask For each address space bit: 0 An access to the flash module can occur for this address space 1 Disable this address space from the flash module. If a reference using this address space is made, it is inhibited from accessing the flash module, and is processed like any other non-flash reference. These bits are useful for power management as detailed in Chapter 9, “Power Management.”
0 V	Valid. When set, this bit enables the flash module; otherwise, the module is disabled. 0 Contents of FLASHBAR are not valid 1 Contents of FLASHBAR are valid

The CFM contains a set of control and status registers located at the register base address as defined by the system level configuration. A summary of the CFM registers is given in Table 17-3.

Table 17-3. CFM Register Address Map

IPSBAR Offset	Register Bits			
	31 - 24	23 - 16	15 - 8	7 - 0
0x1D_0000	CFMMCR		CFMCLKD	RESERVED[1]
0x1D_0004	RESERVED[1]			
0x1D_0008	CFMSEC			
0x1D_000C	RESERVED[1]			
0x1D_0010	CFMPROT			
0x1D_0014	CFMSACC			
0x1D_0018	CFMDACC			
0x1D_001C	RESERVED[1]			
0x1D_0020	CFMUSTAT	RESERVED[1]		
0x1D_0024	CFMCMD	RESERVED[1]		
0x1D_0028	RESERVED[1]			
0x1D_002C	RESERVED[1]			
0x1D_0030	RESERVED[1]			
0x1D_0034	RESERVED[1]			
0x1D_0038	RESERVED[1]			
0x1D_003C	RESERVED[1]			
0x1D_0040	RESERVED[1]			
0x1D_0044	RESERVED[1]			
0x1D_0048	RESERVED[1]		CFMCLKSEL	

[1] Access to reserved address locations generate a cycle termination transfer error.

17.3.3 Register Descriptions

17.3.3.1 CFMMCR — CFM Module Configuration Register

The CFMMCR register is used to configure and control the operation of the internal bus interface.

IPSBAR Access: User read/write

Offset: 0x1D_0000 (CFMMCR)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	LOCK	PVIE	AEIE	CBEIE	CCIE	KEYACC	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-4. CFM Module Configuration Register (CFMMCR)

CFMMCR register bits [10:5] are readable and writable with restrictions, while the remaining bits read 0 and are not writable.

Table 17-4. CFMMCR Field Descriptions

Field	Description
15-11	Reserved, reads as 0
10 LOCK	Write lock control. The LOCK bit is always readable and is set once. 1 = CFMPROT, CMFSACC, and CFMDACC registers are write-locked. 0 = CFMPROT, CMFSACC, and CFMDACC registers are writable.
9 PVIE	Protection violation interrupt enable The PVIE bit is always readable and writable. The PVIE bit enables an interrupt in case the protection violation flag, PVIOL in the CFMUSTAT register, is set. 1 = An interrupt is requested when the PVIOL flag is set. 0 = PVIOL interrupt disabled.
8 AEIE	Access error interrupt enable The AEIE bit is always readable and writable. The AEIE bit enables an interrupt in case the access error flag, ACCERR in the CFMUSTAT register, is set. 1 = An interrupt is requested when the ACCERR flag is set. 0 = ACCERR interrupt disabled.
7 CBEIE	Command buffer empty interrupt enable The CBEIE bit is always readable and writable. The CBEIE bit enables an interrupt in case the command buffer empty flag, CBEIF in the CFMUSTAT register, is set. 1 = An interrupt is requested when the CBEIF flag is set. 0 = CBEIF interrupt disabled.
6 CCIE	Command complete interrupt enable The CCIE bit is always readable and writable. The CCIE bit enables an interrupt in case the command completion flag, CCIF in the CFMUSTAT register, is set. 1 = An interrupt is requested when the CCIF flag is set. 0 = CCIF interrupt disabled.

Table 17-4. CFMMCR Field Descriptions (continued)

Field	Description
5 KEYACC	Enable security key writing The KEYACC bit is readable and only writable if the KEYEN bits in the CFMSEC register are set to enable backdoor key access. 1 = Writes to CFM flash memory are interpreted as keys to release security. 0 = Writes to CFM flash memory are interpreted as the start of a command write sequence.
4-0-	Reserved, reads as 0

17.3.3.2 CFMCLKD — CFM Clock Divider Register

The CFMCLKD register is used to control the period of the clock used for timed events in program and erase algorithms.

IPSBAR Access: User read/write
Offset: 0x1D_0002 (CFMCLKD)

	7	6	5	4	3	2	1	0
R	DIVLD	PRDIV8	DIV					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 17-5. CFM Clock Divider Register (CFMCLKD)

All CFMCLKD register bits are readable, while bits [6:0] write once and bit 7 is not writable.

Table 17-5. CFMCLKD Field Descriptions

Field	Description
7 DIVLD	Clock divider loaded 1 = CFMCLKD register has been written to since the last reset. 0 = CFMCLKD register has not been written.
6 PRDIV8	Enable prescalar by 8 1 = Enables a prescalar to divide the internal flash bus clock by 8 before feeding into the clock divider. 0 = The internal flash bus clock is directly fed into the clock divider.
5-0 DIV	Clock divider bits The combination of PRDIV8 and DIV effectively divides the internal flash bus clock down to a frequency of 150 KHz - 200 KHz. The internal flash bus clock frequency range is 150 KHz < internal flash bus clock < 102.4 MHz. The CFMCLKD register bits PRDIV8 and DIV must be set with appropriate values before programming or erasing the CFM flash memory Section 17.4.2.3.1, “Writing the CFMCLKD Register”.

17.3.3.3 CFMSEC — CFM Security Register

The CFMSEC register is used to store the flash security word and CFM security state.

IPSBAR Access: User read/write

Offset: 0x1D_0008 (CFMSEC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	KEYEN	SECSTAT	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	F[1]	-[2]	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 17-6. CFM Security Register (CFMSEC)

1 The reset state for bit 31 is loaded from the flash configuration field at offset 0x0414 during reset (see Table 17-1).

2 Reset state determined by security state of CFM.

CFMSEC register bits [31:30,15:0] are readable, while the remaining bits read 0 and all bits are not writable.

Table 17-6. CFMSEC Field Descriptions

Field	Description
31 KEYEN	Enable backdoor key access to unlock security 1 = Backdoor key access to flash module is enabled. 0 = Backdoor key access to flash module is disabled.
30 SECSTAT	Flash memory security status 1 = Flash security is enabled. 0 = Flash security is disabled.
29-16	Reserved, should read 0
15 - 0 SEC	Flash memory security bits The SEC bits define the security state of the MCU as shown in Table 17-7, which defines the single code that enables the security feature in the CFM

The CFMSEC register is loaded from the flash configuration field in the flash block at offset 0x0414 during the reset sequence, indicated by F in Figure 17-6.

Table 17-7. CFM Security States

SEC[15:0]	Description
0x4AC8[1]	Flash Memory Secured
All other combinations	Flash Memory Unsecured

1 This value was chosen because it represents the ColdFire HALT instruction, making it unlikely that a user compiled code accidentally programmed at the security configuration field location would unintentionally secure the flash memory.

The CFM flash security operation is described in Section 17.4.3, “Flash Security Operation.

17.3.3.4 CFMPROT — CFM Protection Register

The CFMPROT register defines which flash logical sectors are protected against program and erase operations.

IPSBAR Access: User read/write

Offset: 0x1D_0010 (CFMPROT)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PROTECT															
Reset	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]

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

[1] The reset state for all bits in CFMPROT is loaded from the flash configuration field at offset 0x0408 during reset (see Table 17-1).

Figure 17-7. CFM Protection Register (CFMPROT)

All CFMPROT register bits are readable and only writable when LOCK equals 0.

The flash memory is divided into logical sectors for the purpose of data protection using the CFMPROT register. The flash memory consists of thirty-two 8-Kbyte sectors, as shown in Figure 17-8.

To change the flash memory protection on a temporary basis, the CFMPROT register should be written after the LOCK bit in the CFMMCR register has been cleared. To change the flash memory protection that is loaded during the reset sequence, the flash logical sector containing the flash configuration field must first be unprotected, then the flash protection bytes must be programmed with the desired value.

Table 17-8. CFMPROT Field Descriptions

Field	Description
31 - 0 PROTECT	Each flash logical sector can be protected from program and erase operations by setting the PROTECT[M] bit. PROTECT[M] = 1: Flash logical sector M is protected. PROTECT[M] = 0: Flash logical sector M is not protected.

PROTECT[31:0]

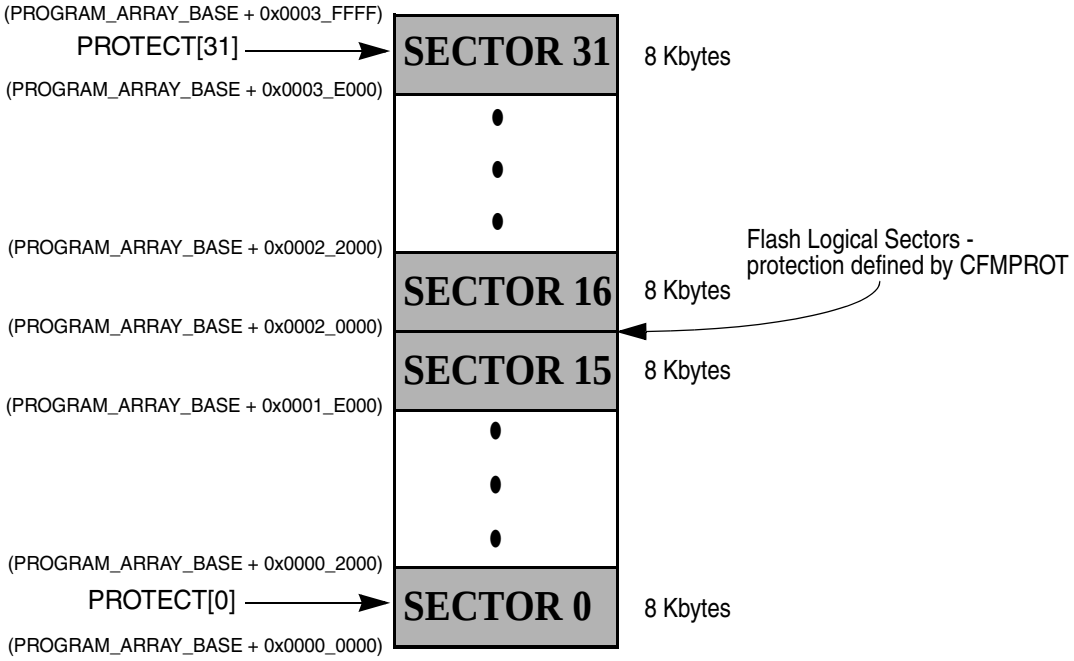


Figure 17-8. CFMPROT Protection Diagram

17.3.3.5 CFMSACC — CFM Supervisor Access Register

The CFMSACC register is used to control supervisor/user access to the flash memory.

IPSBAR Access: User read/write

Offset: 0x1D_0014 (CFMSACC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SUPV															
Reset	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]

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

[1] The reset state for all bits in CFMSACC is loaded from the flash configuration field at offset 0x040C during reset (see Table 17-1).

Figure 17-9. CFM Supervisor Access Register (CFMSACC)

All CFMSACC register bits are readable and only writable when LOCK equals 0.

To change the flash supervisor access on a temporary basis, the CFMSACC register should be written after the LOCK bit in the CFMMCR register has been cleared. To change the flash supervisor access that is loaded during the reset sequence, the flash logical sector containing the flash configuration field must first be unprotected, then the flash supervisor access bytes must be programmed with the desired value. Each

flash logical sector may be mapped into supervisor or unrestricted address space (see Figure 17-8 for details on flash sector mapping).

Table 17-9. CFMSACC Field Descriptions

Field	Description
31 - 0 SUPV	Flash address space assignment for supervisor/user access SUPV[M] = 1: Flash logical sector M is placed in supervisor address space. SUPV[M] = 0: Flash logical sector M is placed in unrestricted address space.

17.3.3.6 CFMDACC — CFM Data Access Register

The CFMDACC register is used to control data/instruction access to the flash memory.

IPSBAR Access: User read/write
Offset: 0x1D_0018 (CFMDACC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DACC															
Reset	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]	F[1]

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

[1] The reset state for all bits in CFMDACC is loaded from the flash configuration field at offset 0x0410 during reset (see Table 17-1).

Figure 17-10. CFM Data Access Register (CFMDACC)

All CFMDACC register bits are readable and only writable when LOCK equals 0.

To change the flash data access on a temporary basis, the CFMDACC register should be written after the LOCK bit in the CFMMCR register has been cleared. To change the flash data access that is loaded during the reset sequence, the flash logical sector containing the flash configuration field must first be unprotected, then the flash data access bytes must be programmed with the desired value. Each flash logical sector may be mapped into data or both data and instruction address space (see Figure 17-8 for details on flash sector mapping).

Table 17-10. CFMDACC Field Descriptions

Field	Description
31 - 0 DACC	Flash memory address space assignment for data/instruction access DACC[M] = 1: Flash logical sector M is placed in data address space. DACC[M] = 0: Flash logical sector M is placed in data and instruction address space.

17.3.3.7 CFMUSTAT — CFM User Status Register

The CFMUSTAT register defines the flash command controller status and flash memory access, protection and verify status.

IPSBAR
Offset: 0x1D_0020 (CFMUSTAT)

Access: User read/write

	7	6	5	4	3	2	1	0
R	CBEIF	CCIF	PVIOL	ACCERR	0	BLANK	0	0
W								
Reset:	1	1	0	0	0	0	0	0

Figure 17-11. CFM User Status Register (CFMUSTAT)

CFMUSTAT register bits CBEIF, PVIOL, ACCERR, and BLANK are readable and writable while CCIF is readable but not writable, and remaining bits read 0 and are not writable.

The CFMUSTAT register bits CBEIF, CCIF, PVIOL, ACCERR, and BLANK are available as external signals cfm_status_bits[7:4,2] on the module boundary.

NOTE

Only one CFMUSTAT register bit can be cleared at a time.

Table 17-11. CFMUSTAT Field Descriptions

Field	Description
7 CBEIF	Command buffer empty interrupt flag The CBEIF flag, set by the flash command controller, indicates that the address, data and command buffers are empty so that a new command write sequence can be started. The CBEIF flag is cleared by writing a 1 to CBEIF as part of a command write sequence. Writing a 0 to the CBEIF flag has no effect on CBEIF but can be used to abort a command write sequence. The CBEIF flag can generate an interrupt if the CBEIE bit in the CFMMCR register is set. 1 = Buffers are ready to accept a new command write sequence. 0 = Buffers are full.
6 CCIF	Command complete interrupt flag The CCIF flag, set by the flash command controller, indicates that there are no more commands pending. The CCIF flag is cleared by the flash command controller when CBEIF is cleared and sets upon completion of all active and pending commands. Writing to the CCIF flag has no effect on CCIF. The CCIF flag can generate an interrupt if the CCIE bit in the CFMMCR register is set. 1 = All commands are completed. 0 = Command in progress.
5 PVIOL	Protection violation The PVIOL flag, set by the flash command controller, indicates an attempt was made to program or erase an address in a protected flash logical sector. The PVIOL flag is cleared by writing a 1 to PVIOL. Writing a 0 to the PVIOL flag has no effect on PVIOL. While the PVIOL flag is set, it is not possible to launch a command or start a command write sequence. 1 = Protection violation has occurred. 0 = No protection violation has been detected.

Table 17-11. CFMUSTAT Field Descriptions (continued)

Field	Description
4 ACCERR	Access error The ACCERR flag, set by the flash command controller, indicates an illegal access was made to the flash memory or registers caused by an illegal command write sequence. The ACCERR flag is cleared by writing a 1 to the ACCERR flag. Writing a 0 to the ACCERR flag has no effect on ACCERR. While the ACCERR flag is set, it is not possible to launch a command or start a command write sequence. See Section 17.4.2.3.5, “Flash Normal Mode Illegal Operations” for details on what action sets the ACCERR flag. 1 = Access error has occurred. 0 = No access error has been detected.
3	Reserved, should read 0
2 BLANK	All flash memory locations or the selected flash logical page have been verified as erased The BLANK flag, set by the flash command controller, indicates that a blank check or page erase verify operation has checked all flash memory locations or the selected flash logical page and found them to be erased. The BLANK flag is cleared by writing a 1 to BLANK. Writing a 0 to the BLANK flag has no effect on BLANK. 1 = All flash memory locations or selected logical page verify as erased. 0 = If a blank check or page erase verify command has been executed, and the CCIF flag is set, then a 0 in the BLANK flag indicates that all flash memory locations are not erased or the selected flash logical page is not erased.
1 -0	Reserved, should read 0

17.3.3.8 CFMCMD — CFM Command Register

The CFMCMD register is the flash command register.

IPSBAR Access: User read/write
Offset: 0x1D_0024 (CFMCMD)

	7	6	5	4	3	2	1	0
R	0				CMD			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 17-12. CFM Command Buffer and Register (CFMCMD)

All CFMCMD register bits are readable and writable except bit 7, which reads zero and is not writable.

Table 17-12. CFMCMD Field Descriptions

Field	Description
7	Reserved, should read 0
6 - 0 CMD	Valid flash memory commands are shown in Table 17-13. Writing a command other than those listed in Table 17-13 during a command write sequence causes the ACCERR flag in the CFMUSTAT register to set.

Table 17-13. CFM Flash Memory Commands

CMD[6:0]	Description
0x05	Blank Check
0x06	Page Erase Verify
0x20	Word Program
0x40	Page Erase
0x41	Mass Erase

17.3.3.9 CFMCLKSEL — CFM Clock Select Register

The CFMCLKSEL register reflects the factory setting for read access latency from the system bus to the flash block.

IPSBAR Access: User read/write

Offset: 0x1D_004A(CFMCLKSEL)

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

[1]Reset state set by factory.

Figure 17-13. CFM Clock Select Register (CFMCLKSEL)

CFMCLKSEL register bits [1:0] are read-only, while the remaining bits read 0 and are not writable.

Table 17-14. CFMCLKSEL Field Descriptions

Field	Description
15 - 2	Reserved, should read 0
1 - 0 CLKSEL	Flash read access latency select The CLKSEL bits set the read access latency to the flash block. Table 17-15 describes the setting that selects between single-cycle and two-cycle flash block read access.

Table 17-15. Clock Select States

CLKSEL[1:0]	Description	Burst Read Access
2'b10	Single-Cycle Flash Block Read Access	1-1-1-1
All other combinations	Two-cycle Flash Block Read Access	2-1-1-1

17.4 Functional Description

17.4.1 General

The following modes and operations are described in the following sections:

1. Flash normal mode (Section 17.4.2, “Flash Normal Mode”)
   a) Read operation (Section 17.4.2.1, “Read Operation”)
   b) Write operation (Section 17.4.2.2, “Write Operation”)
   c) Program, erase, and verify operations (Section 17.4.2.3, “Program, Erase, and Verify Operations”)
   d) Stop mode (Section 17.4.2.4, “Stop Mode”)
2. Flash security operation (Section 17.4.3, “Flash Security Operation”)

17.4.2 Flash Normal Mode

In flash normal mode, the user can access the CFM registers and the CFM flash memory (see Section 17.3.1, “Memory Map”).

17.4.2.1 Read Operation

A valid read operation occurs when a transfer request is initiated, the address is equal to an address within the valid range of the CFM flash memory space and the read/write control indicates a read cycle.

17.4.2.2 Write Operation

A valid write operation occurs when a transfer request is initiated, the address is equal to an address within the valid range of the CFM flash memory space and the read/write control indicates a write cycle. The action taken on a valid flash array write depends on the subsequent user command issued as part of a valid command write sequence. Only 32-bit write operations are allowed to the flash memory space. Byte and half-word write operations to the flash memory space results in a cycle termination transfer error.

17.4.2.3 Program, Erase, and Verify Operations

Write and read operations are used for the program, erase, and verify algorithms described in this section. These algorithms are controlled by the flash memory controller whose timebase, for program and erase operations, is derived from the internal flash bus clock via a programmable counter. The command register as well as the associated address and data registers operate as a buffer and a register (2-stage FIFO), so that a new command along with the necessary data and address can be stored to the buffer while the previous command remains in progress. This buffering operation provides time optimization when programming more than one word on a physical row in the flash memory as the high voltage generation can be kept active in between two programming operations, thereby saving the time overhead needed for setup of the high voltage charge pumps. Buffer empty as well as command completion are signaled by flags in the CFMUSTAT register with interrupts generated, if enabled.

The next four sections describe the following:

- How to write the CFMCLKD register
- Command write sequences used to program, erase, and verify the flash memory
- Valid flash commands
- Errors resulting from illegal command write sequences to the flash memory

17.4.2.3.1 Writing the CFMCLKD Register

Prior to issuing any command, it is first necessary to write the CFMCLKD register to divide the input clock to within the 150 KHz to 200 KHz range. The CFMCLKD register bits PRDIV8 and DIV are set as follows:

For frequencies of the input clock greater than 12.8 MHz, the CFMCLKD bit PRDIV8 must be set.

CFMCLKD DIV bit field must be chosen such that the following equation is valid:

```
If PRDIV8 == 1 then FCLK = input clock / 8, else FCLK = input clock
If (FCLK[KHz] / 200KHz) is integer then DIV = (FCLK[KHz] / 200KHz) - 1,
else DIV = INT (FCLK[KHz] / 200kHz)
```

FCLK, the clock to the flash block timing control, is therefore:

```
FCLK = (input clock) / (DIV + 1)
150KHz < FCLK <= 200KHz
```

For example, if the input clock frequency is 33 MHz, the CFMCLKD DIV field should be set to 0x14 and bit PRDIV8 set to 1. The resulting FCLK is 196.4 KHz. As a result, the flash memory program and erase algorithm timings are increased over the optimum target by:

```
(200 - 196.4) / 200 x 100% = 1.78%
```

Remark: INT(X) means taking the integer part of X
Example: INT(33MHz/8/200KHz) = 20

CAUTION

Programming the flash with input clock < 150 KHz should be avoided. Setting CFMCLKD to a value such that FCLK < 150 KHz can destroy the flash memory due to overstress. Setting CFMCLKD to a value such that FCLK > 200 KHz can result in incomplete programming or erasure of the flash memory array cells.

NOTE

Program and Erase command execution time increases proportionally with the period of FCLK.

If the CFMCLKD register is written, the DIVLD bit is set automatically. If the DIVLD bit is 0, the CFMCLKD register has not been written since the last reset. No command can be executed if the CFMCLKD register has not been written to Section 17.4.2.3.5, “Flash Normal Mode Illegal Operations.”

17.4.2.3.2 Command Write Sequence

The flash command controller is used to supervise the command write sequence to execute blank check, page erase verify, program, page erase, and mass erase algorithms.

Before starting a command write sequence, the ACCERR and PVIOL flags in the CFMUSTAT register must be clear and the CBEIF flag should be tested to determine the state of the address, data, and command buffers. If the CBEIF flag is set, indicating the buffers are empty, a new command write sequence can be executed.

A command write sequence consists of three steps which must be strictly adhered to, because writes to the CFM are not permitted between steps. However, flash register and array reads are allowed during a command write sequence. The basic command write sequence is as follows:

1. Write to one or more addresses in the flash memory.
2. Write a valid command to the CFMCMD register.
3. Clear CBEIF flag by writing a 1 to CBEIF to launch the command.

When the CBEIF flag is cleared, the CCIF flag is cleared on the same bus cycle by the flash command controller indicating that the command was successfully launched. The CBEIF flag is set again indicating that the address, data, and command buffers are ready for a new command write sequence to begin. A buffered command waits for the active command to be completed before being launched. The CCIF flag in the CFMUSTAT register is set upon completion of all active and buffered commands.

A command write sequence can be aborted at anytime prior to clearing the CBEIF flag in the CFMUSTAT register by writing a 0 to the CBEIF flag. The ACCERR flag in the CFMUSTAT register is set after successfully aborting a command write sequence and the ACCERR flag must be cleared prior to starting a new command write sequence.

17.4.2.3.3 Bus Arbitration During Write Operations

After a command has been successfully launched, the CFM signals the core platform to hold off read accesses to any active flash physical block until all active and buffered commands have completed (CCIF=1). A flash write operation from the internal flash bus holds off the Core platform until it is completed.

17.4.2.3.4 Flash Normal Mode Commands

Table 17-16 summarizes the valid flash normal mode commands.

Table 17-16. CFM Flash Memory Command Description

CFMCMD	Meaning	Description
$05	Blank Check	Verify that the entire flash memory is erased. If all bits are erased, the BLANK bit is set in the CFMUSTAT register, Figure 17-11, upon command completion.
$06	Page Erase Verify	Verifies that a flash logical page is erased. If the flash logical page is erased, the BLANK bit is set in the CFMUSTAT register, Figure 17-11, upon command completion.
$20	Program	Program a 32-bit word.
$40	Page Erase	Erase a flash logical page.
$41	Mass Erase	Erase the entire flash memory. All flash memory protection must be disabled.

Blank Check

The blank check operation verifies that all flash memory addresses in the CFM are erased.

An example flow to execute the blank check command is shown in Figure 17-14. The blank check command write sequence is as follows:

1. Write to any flash memory address to start the command write sequence for the blank check command. The specific address and data written during the blank check command write sequence is ignored.
2. Write the blank check command, $05, to the CFMCMD register.
3. Clear the CBEIF flag by writing a 1 to CBEIF to launch the blank check command.

Because all flash physical blocks are verified simultaneously, the number of internal flash bus cycles required to execute the blank check operation on a fully erased flash memory is equal to the number of word addresses in a flash logical block plus 15 internal flash bus cycles as measured from the time the CBEIF flag is cleared until the CCIF flag is set in the CFMUSTAT register. Upon completion of the blank check operation (CCIF=1), the BLANK flag sets in the CFMUSTAT register if the entire flash memory is erased. If any flash memory location is not erased, the blank check operation terminates and the BLANK flag remains clear.

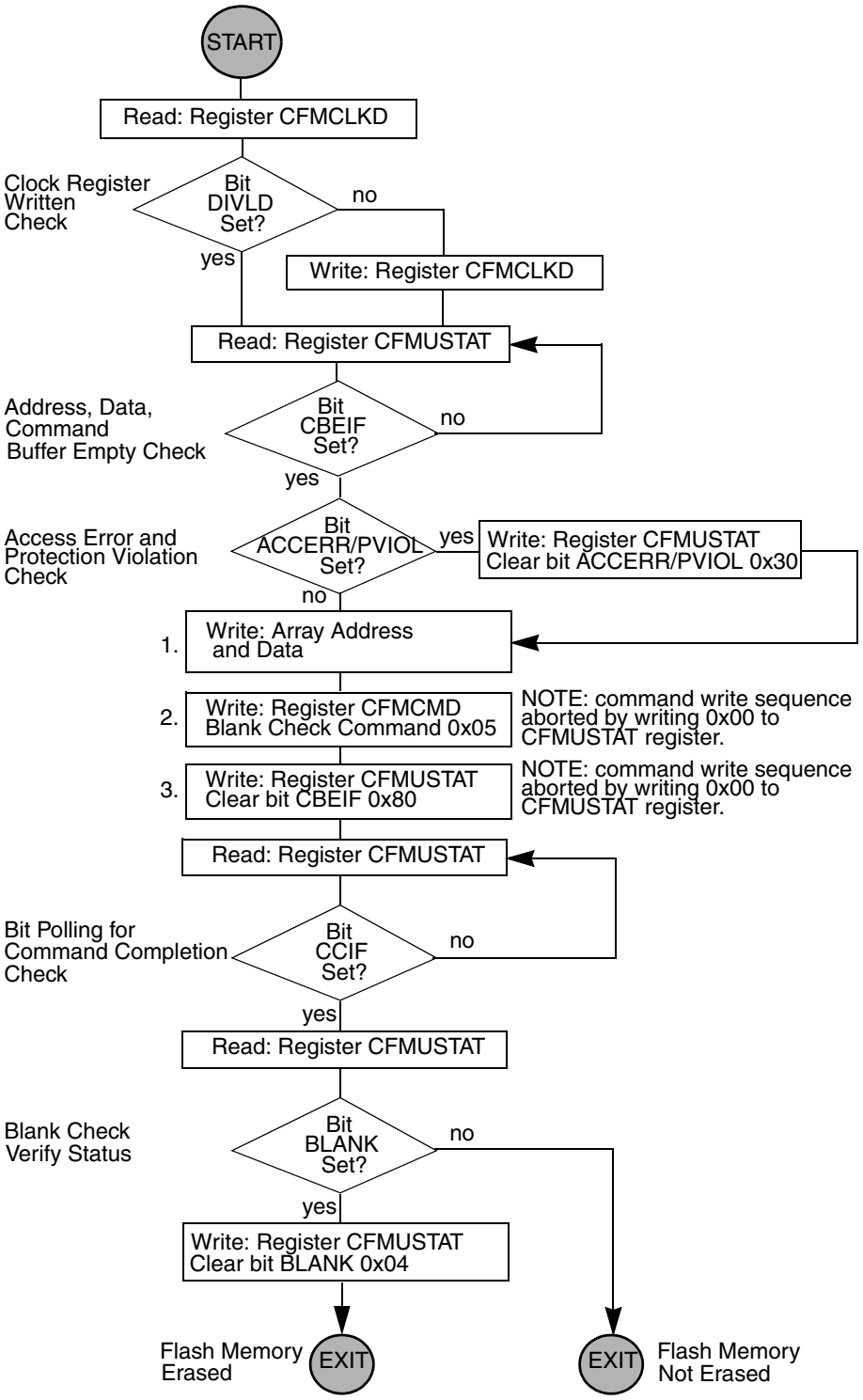


Figure 17-14. Example Blank Check Command Flow

Page Erase Verify

The page erase verify operation verifies all memory addresses in a flash logical page are erased.

An example flow to execute the page erase verify operation is shown in Figure 17-15. The page erase verify command write sequence is as follows:

1. Write to any word address in a flash logical page to start the command write sequence for the page erase verify command. The address written determines the flash logical page to be verified, while the data written during the page erase verify command write sequence is ignored.
2. Write the page erase verify command, $06, to the CFMCMD register.
3. Clear the CBEIF flag by writing a 1 to CBEIF to launch the page erase verify command.

Because the word addresses in even and odd flash blocks are interleaved, pages from adjacent interleaving flash physical blocks are automatically erase verified at the same time. The number of internal flash bus cycles required to execute the page erase verify operation on a fully erased flash logical page is equal to the number of word addresses in a flash logical page plus 15 internal flash bus cycles as measured from the time the CBEIF flag is cleared until the CCIF flag is set in the CFMUSTAT register.

Upon completion of any page erase verify operation (CCIF=1), the BLANK flag in the CFMUSTAT register is set if all addresses in the selected flash logical page are verified to be erased. If any address in the selected flash logical page is not erased, the page erase verify operation terminates and the BLANK flag remains clear.

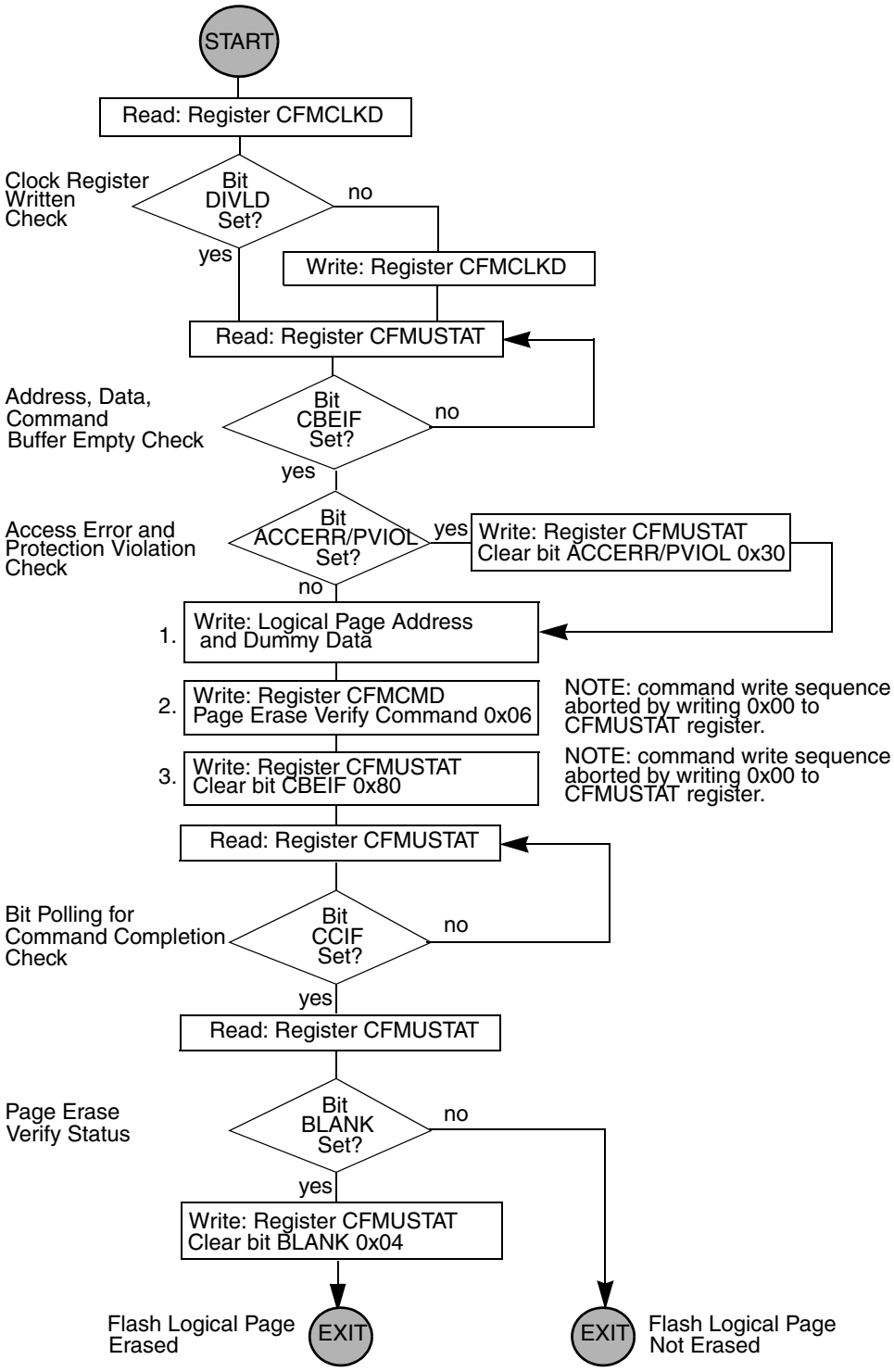


Figure 17-15. Example Page Erase Verify Command Flow

Program

The program operation programs a previously erased address in the flash memory using an embedded algorithm.

An example flow to execute the program operation is shown in Figure 17-16. The program command write sequence is as follows:

1. Write to a word address in a flash physical block to start the command write sequence for the program command. The word address written determines the flash physical block address to program while the data written during the program command write sequence determines the data stored at that address.
   To write to two physical blocks simultaneously, perform the following steps:
   a) Write data to even address (where address is a multiple of eight)
   b) Write data to odd address (previous address + 4)
   c) Write PROGRAM command to CFMCMD
   d) Clear CBEIF, by writing a 1 to it

   The flash physical block written to in the first array write limits the ability to simultaneously program in block order only those flash physical blocks that remain.
2. Write the program command, $20, to the CFMCMD register.
3. Clear the CBEIF flag by writing a 1 to CBEIF to launch the program command.

If the address to be programmed is in a protected sector of the flash memory, the PVIOL flag in the CFMUSTAT register sets and the program command does not launch. After the program command has successfully launched, the CCIF flag in the CFMUSTAT register sets after the program operation has completed unless a new command write sequence has been buffered.

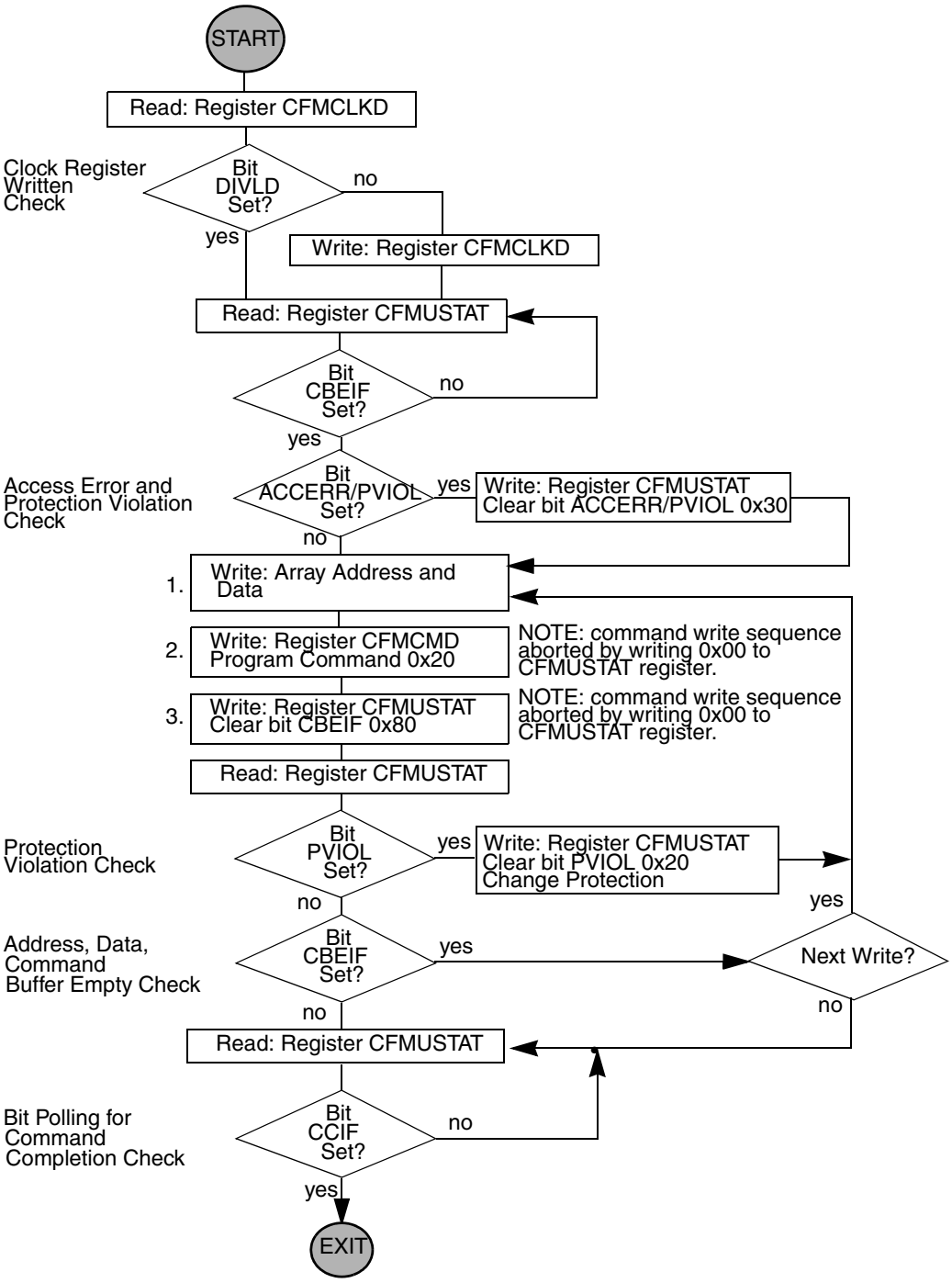


Figure 17-16. Example Program Command Flow

Page Erase

The page erase operation erases all memory addresses in a flash logical page using an embedded algorithm.

An example flow to execute the page erase operation is shown in Figure 17-17. The page erase command write sequence is as follows:

1. Write to any word address in a flash logical page to start the command write sequence for the page erase command. The word address written determines the flash logical page to erase while the data written during the page erase command write sequence is ignored.
2. Write the page erase command, $40, to the CFMCMD register.
3. Clear the CBEIF flag by writing a 1 to CBEIF to launch the page erase command.

If the flash logical page to be erased is in a protected sector of the flash memory, the PVIOL flag in the CFMUSTAT register sets and the page erase command does not launch. After the page erase command has successfully launched, the CCIF flag in the CFMUSTAT register sets after the page erase operation has completed, unless a new command write sequence has been buffered.

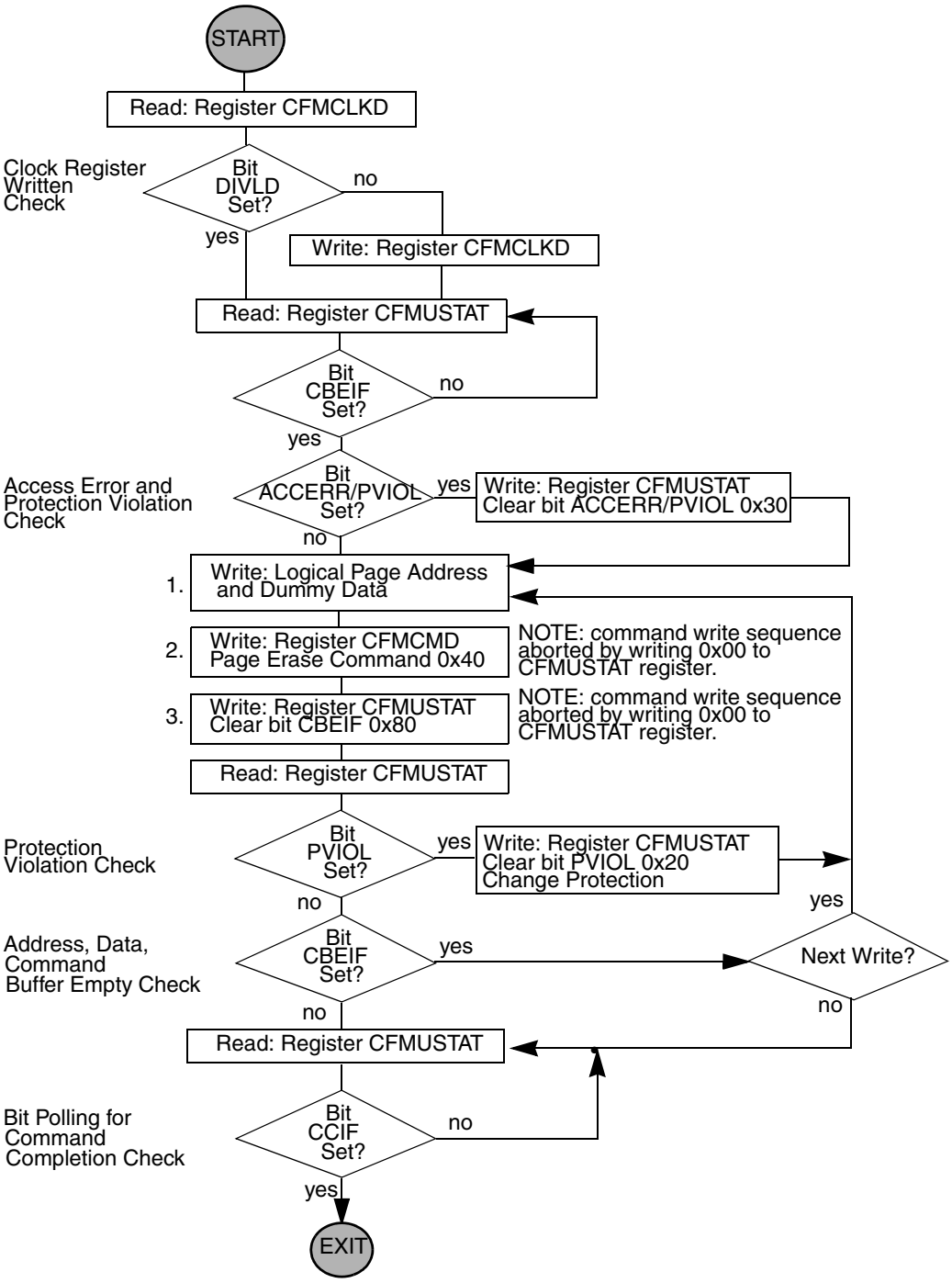


Figure 17-17. Example Page Erase Command Flow

Mass Erase

The mass erase operation erases all flash memory addresses using an embedded algorithm.

An example flow to execute the mass erase command is shown in Figure 17-18. The mass erase command write sequence is as follows:

1. Write to any flash memory address to start the command write sequence for the mass erase command. The specific address and data written during the mass erase command write sequence is ignored.
2. Write the mass erase command, $41, to the CFMCMD register.
3. Clear the CBEIF flag by writing a 1 to CBEIF to launch the mass erase command.

If any flash logical sector is protected, the PVIOL flag in the CFMUSTAT register sets during the command write sequence and the mass erase command does not launch. After the mass erase command has successfully launched, the CCIF flag in the CFMUSTAT register sets after the mass erase operation has completed, unless a new command write sequence has been buffered.

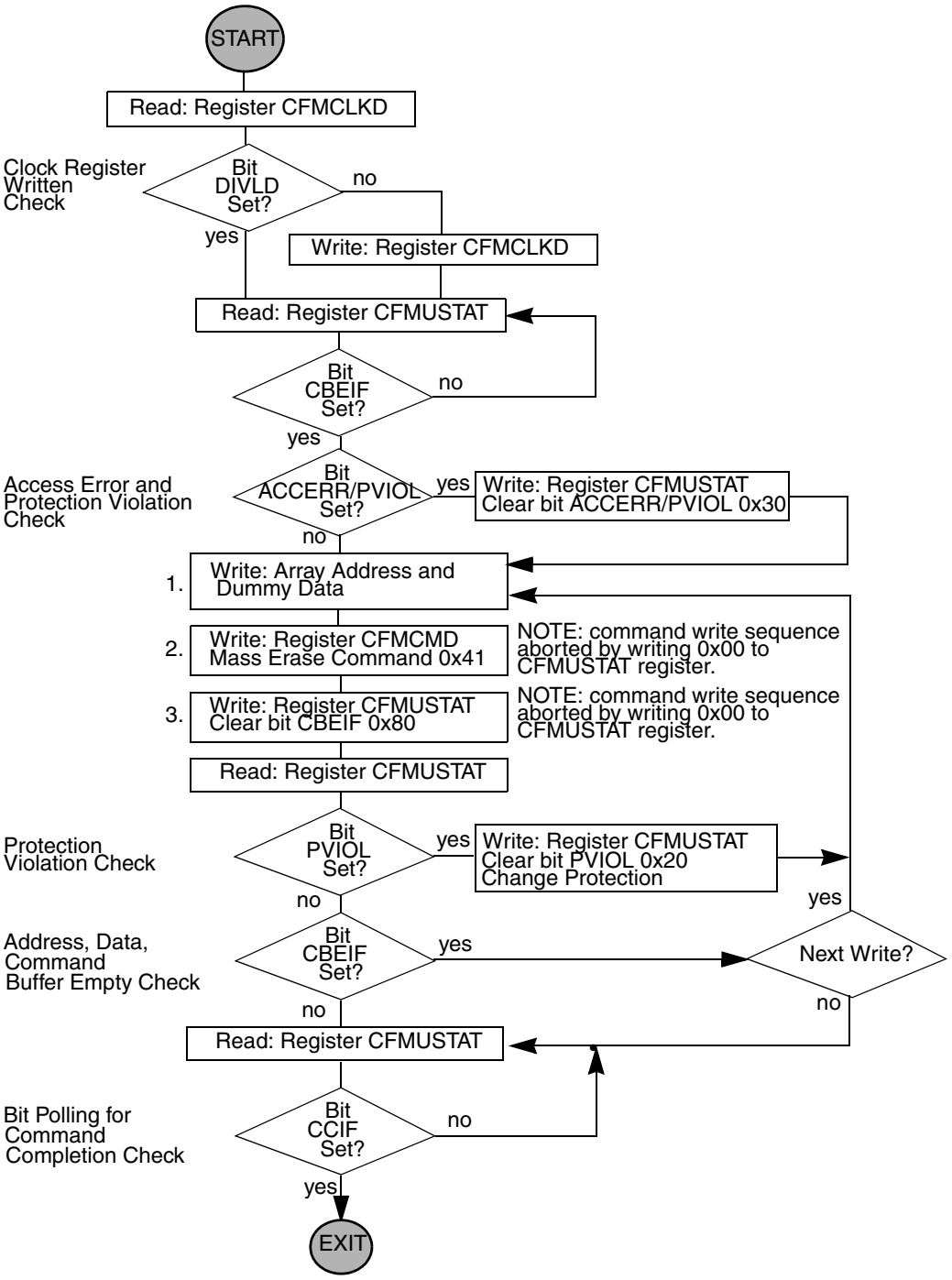


Figure 17-18. Example Mass Erase Command Flow

17.4.2.3.5 Flash Normal Mode Illegal Operations

The ACCERR flag is set during the command write sequence if any of the following illegal operations are performed, causing the command write sequence to immediately abort:

1. Writing to the flash memory before initializing CFMCLKD.
2. Writing to the flash memory while CBEIF is not set.

3. Writing to a flash block with a data size other than 32 bits.
4. After writing to the flash even block, writing an additional word to the flash memory during the flash command write sequence other than the flash odd block.
5. Writing an invalid flash normal mode command to the CFMCMD register.
6. Writing to any CFM register other than CFMCMD after writing to the flash memory.
7. Writing a second command to the CFMCMD register before executing the previously written command.
8. Writing to any CFM register other than CFMUSTAT (to clear CBEIF) after writing to the command register, CFMCMD.
9. The part enters stop mode and any command is in progress. Upon entering STOP mode, any active command is aborted.
10. Aborting a command write sequence by writing a 0 to the CBEIF flag after writing to the flash memory or after writing a command to the CFMCMD register but before the command is launched.

The PVIOL flag is set during the command write sequence if any of the following illegal operations are performed, causing the command write sequence to immediately abort:

1. Writing a program command if the address to program is in a protected flash logical sector.
2. Writing a page erase command if the address to erase is in a protected flash logical sector.
3. Writing a mass erase command while any protection is enabled.

If a read operation is attempted on a flash logical block while a command is active on that logical block (CCIF=0), the read operation returns invalid data and the ACCERR flag in the CFMUSTAT register is not set.

17.4.2.4 Stop Mode

If a command is active (CCIF=0) when the MCU enters stop mode, the flash command controller and flash memory controller perform the following:

1. The active command is aborted, and the data being programmed or erased is lost.
2. The high voltage circuitry to the flash arrays is switched off.
3. Any buffered command (CBEIF=0) is not executed after the MCU exits stop mode.
4. The CCIF and ACCERR flags are set if a command is active when the MCU enters stop mode.

CAUTION

As active commands are immediately aborted when the MCU enters stop mode, it is strongly recommended that the user does not execute the stop instruction during program and erase operations.

If a command is not active (CCIF=1) when the MCU enters stop mode, the ACCERR flag does not set.

17.4.3 Flash Security Operation

The CFM provides security information to the Integration module and the rest of the MCU. This security information is stored within a word in the flash configuration field. This security word is read automatically after each reset and stored in the CFMSEC register.

NOTE

Enabling flash security disables BDM communications.

In flash normal mode, the user can bypass the security via a backdoor access sequence using an 8-byte long key. Upon successful completion of the backdoor access sequence, the SECSTAT bit in the CFMSEC register is cleared indicating that the MCU is unsecured.

The CFM may be unsecured via one of the following methods:

1. Executing a backdoor access sequence.
2. Passing a blank check operation on the flash memory.

17.4.3.1 Backdoor Access Sequence

If the KEYEN bits in the CFMSEC register are set to the enabled state, the user can bypass security by performing the following:

1. Setting the KEYACC bit in the CFMMCR register.
2. Writing the correct 8-byte backdoor comparison key to the flash memory at offset 0x0400 - 0x0407. This operation must be composed of two 32-bit writes to address 0x0400 and 0x0404 in that order. The two backdoor write cycles can be separated by any number of internal flash bus cycles.

NOTE

Any attempt to use a key of all zeros or all ones locks the backdoor access sequence until the CFM is reset.

3. Clearing the KEYACC bit.
4. If all 8 bytes written match the flash memory content at offset 0x0400 - 0x0407, then security is bypassed until the next reset.

In the unsecured state, the user has full control of the contents of the 8-byte backdoor comparison key by programming the bytes at offset 0x0400 - 0x0407 of the flash configuration field. If at any time a key of all 0s or all 1s is received, the backdoor access sequence is terminated and cannot be successfully restarted until after the CFM is reset.

The security of the CFM as defined in the flash security word at address offset 0x0414 is not changed by the executing the backdoor access sequence to unsecure the device. After the next reset sequence, the CFM is secured again and the same backdoor key is in effect unless the flash configuration field was changed by program or erase prior to reset. The backdoor access sequence to unsecure the device has no effect on the program and erase protections defined in the CFM protection register.

The contents of the flash security word at address offset 0x0414 must be changed by programming that address when the device is unsecured and the sector containing the flash configuration field is unprotected.

17.4.3.2 Blank Check

A secured CFM can be unsecured by verifying that the entire flash memory is erased. If required, the mass erase command can be executed on the flash memory. The blank check command must then be executed on the flash memory. The CFM is unsecured if the blank check operation determines that the entire flash memory is erased. After the next reset sequence, the security state of the CFM is determined by the flash security word at address offset 0x0414. For further details on security, see the MCU security specification.

17.4.3.3 JTAG Lockout Recovery

A secured CFM can be unsecured by mass erasing the flash memory via a sequence of JTAG commands, as specified in the system level security documentation followed by a reset of the MCU.

17.4.3.4 EzPort Lockout Recovery

A secured CFM can also be unsecured by mass erasing the flash memory via the EzPort bulk erase (BE) command. Doing so clears the flash security (FS) bit in the EzPort status register (see Section 21.4.1.3, “Read Status Register”), after which a reset chip (RESET) command can be issued to regain access to the device.

.

Chapter 18
Fast Ethernet Controller (FEC)

This chapter provides a feature-set overview and a functional block diagram. Additionally, detailed descriptions of operation and the programming model are included.

18.1 Overview

The Ethernet Media Access Controller (MAC) is designed to support 10 and 100 Mbps Ethernet/IEEE 802.3 networks. The MAC must be used in conjunction with the on-board transceiver interface and transceiver function to complete the interface to the media.

18.1.1 Features

The FEC incorporates the following features:

- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbps throughput) with a minimum system clock rate of 50 MHz
- Support for half-duplex operation (100 Mbps throughput) with a minimum system clock rate of 25 MHz
- Retransmission from transmit FIFO following a collision (no processor bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)
- Address recognition
  - Frames with broadcast address may be always accepted or always rejected
  - Exact match for single 48-bit individual (unicast) address
  - Hash (64-bit hash) check of individual (unicast) addresses
  - Hash (64-bit hash) check of group (multicast) addresses
  - Promiscuous mode

18.2 Modes of Operation

The primary operational modes are described in this section.

18.2.1 Full and Half Duplex Operation

Full duplex mode is intended for use on point to point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Selection of the duplex mode is controlled by TCR[FDEN].

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE] and TCR[TFC_PAUSE] bits, the RCR[FCE] bit, and Section 18.4.10, “Full Duplex Flow Control,” for more details.

18.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 18.4.5, “Network Interface Options”.

18.2.2.1 10 Mbps and 100 Mbps MII Interface

MII is the Media Independent Interface defined by the IEEE 802.3 standard for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by asserting RCR[MII_MODE].

The speed of operation is determined by the ETXCLK and ERXCLK pins which are driven by the external transceiver. The transceiver auto-negotiates the speed or it may be controlled by software via the serial management interface (EMDC/EMDIO pins) to the transceiver. Refer to the MMFR and MSCR register descriptions as well as the section on the MII for a description of how to read and write registers in the transceiver via this interface.

18.2.3 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in Section 18.4.8, “Ethernet Address Recognition”.

18.2.4 Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in Section 18.4.13, “Internal and External Loopback”.

18.3 FEC Top-Level Functional Diagram

The block diagram of the FEC is shown below. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.

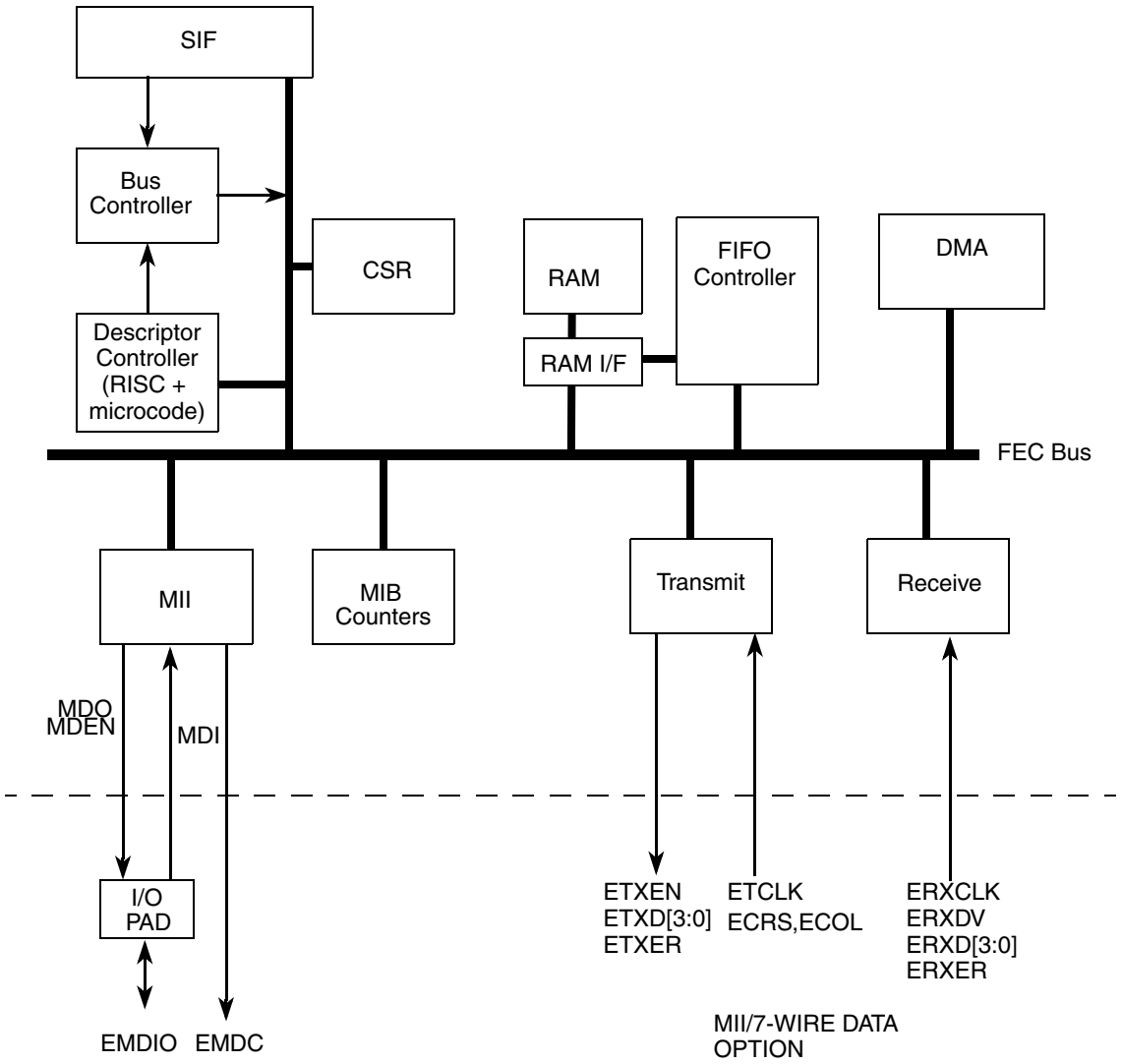


Figure 18-1. FEC Block Diagram

The descriptor controller is a RISC-based controller that provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine is for the transfer of FEC data only, and is not related to the DMA controller described in Chapter 20, "DMA Controller Module," nor to the DMA timers described in Chapter 24, "DMA Timers (DTIM0–DTIM3)."

The RAM is the focal point of all data flow in the Fast Ethernet Controller and is divided into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

The user controls the FEC by writing, through the SIF (Slave Interface) module, into control registers located in each block. The CSR (control and status register) block provides global control (e.g. Ethernet reset and enable) and interrupt handling registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the EMDC (Management Data Clock) and EMDIO (Management Data Input/Output) lines of the MII interface.

The DMA block provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The Transmit and Receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The Message Information Block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters. See Section 18.5.3, “MIB Block Counters Memory Map” for more information.

18.4 Functional Description

This section describes the operation of the FEC, beginning with the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FEC.

18.4.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations the user must initialize prior to enabling the FEC.

18.4.1.1 Hardware Controlled Initialization

In the FEC, registers and control logic that generate interrupts are reset by hardware. A hardware reset deasserts output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER_EN] bit is cleared. ECR[ETHER_EN] is deasserted by a hard reset or may be deasserted by software to halt operation. By deasserting ECR[ETHER_EN], the configuration control registers such as the TCR and RCR are not reset, but the entire data path is reset.

Table 18-1. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

18.4.2 User Initialization (Prior to Asserting ECR[ETHER_EN])

The user needs to initialize portions the FEC prior to setting the ECR[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in Table 18-2.

Table 18-2. User Initialization (Before ECR[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM (locations IPSBAR + 0x1200-0x12FC)

FEC FIFO/DMA registers that require initialization are defined in Table 18-3.

Table 18-3. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

18.4.3 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

Table 18-4 shows microcontroller initialization operations.

Table 18-4. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

18.4.4 User Initialization (After Asserting ECR[ETHER_EN])

After asserting ECR[ETHER_EN], the user can set up the buffer/frame descriptors and write to the TDAR and RDAR. Refer to Section 18.6, “Buffer Descriptors” for more details.

18.4.5 Network Interface Options

The FEC supports an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The interface mode is selected by the RCR[MII_MODE] bit. In MII mode (RCR[MII_MODE] = 1), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. These signals are shown in Table 18-5 below.

Table 18-5. MII Mode

Signal Description	EMAC pin
Transmit Clock	ETXCLK
Transmit Enable	ETXEN
Transmit Data	ETXD[3:0]
Transmit Error	ETXER
Collision	ECOL
Carrier Sense	ECRS
Receive Clock	ERXCLK
Receive Data Valid	ERXDV
Receive Data	ERXD[3:0]

Table 18-5. MII Mode (continued)

Signal Description	EMAC pin
Receive Error	ERXER
Management Data Clock	EMDC
Management Data Input/Output	EMDIO

The 7-wire serial mode interface (RCR[MII_MODE] = 0) operates in what is generally referred to as the AMD mode. 7-wire mode connections to the external transceiver are shown in Table 18-6.

Table 18-6. 7-Wire Mode Configuration

Signal description	EMAC Pin
Transmit Clock	ETXCLK
Transmit Enable	ETXEN
Transmit Data	ETXD[0]
Collision	ECOL
Receive Clock	ERXCLK
Receive Data Valid	ERXDV
Receive Data	ERXD[0]

18.4.6 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ECR[ETHER_EN] is asserted and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the TFWR), the MAC transmit logic asserts ETXEN and start transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (ECRS asserts). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, the transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See Section 18.4.14.1, “Transmission Errors” for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (Frame Check Sequence or 32-bit Cyclic Redundancy Check, CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of

the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end of frame buffer equals 1).

Buffer (TXB) and frame (TFINT) interrupts may be generated as determined by the settings in the EIMR.

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL, bytes the BABT interrupt is asserted; however, the entire frame is transmitted (no truncation).

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR register. When the TCR[GTS] is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame finishes or terminates with a collision. After the transmitter has stopped the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

18.4.6.1 Duplicate Frame Transmission

The FEC fetches transmit buffer descriptors (TxBDs) and the corresponding transmit data continuously until the transmit FIFO is full. It does not determine whether the TxBD to be fetched is already being processed internally (as a result of a wrap). As the FEC nears the end of the transmission of one frame, it begins to DMA the data for the next frame. To remain one BD ahead of the DMA, it also fetches the TxBD for the next frame. The FEC can fetch from memory a BD that has already been processed but not yet written back (it is read a second time with the R bit remaining set). In this case, the data is fetched and transmitted again.

Using at least three TxBDs fixes this problem for large frames, but not for small frames. To ensure correct operation for large or small frames, one of the following must be true:

- The FEC software driver ensures that there is always at least one TxBD with the ready bit cleared.
- Every frame uses more than one TxBD and every TxBD but the last is written back immediately after the data is fetched.
- The FEC software driver ensures a minimum frame size, *n*. The minimum number of TxBDs is then (Tx FIFO Size ÷ (*n* + 4)) rounded up to the nearest integer (though the result cannot be less than three). The default Tx FIFO size is 192 bytes; this size is programmable.

18.4.7 FEC Frame Reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by asserting ECR[ETHER_EN], it immediately starts processing receive frames. When ERXDV asserts, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the frame is processed by the receiver. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of RX_D0 following assertion of ERXDV are ignored. Following the first 16 bit times the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame have been received, the FEC performs address recognition on the frame.

After a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame is accepted and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to reject the frame. Thus, no collision fragments are presented to the user except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and after the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV and TR status bits, and the frame length. See Section 18.4.14.2, “Reception Errors” for more details.

Receive Buffer (RXB) and Frame Interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt occurs and the LG bit in the Receive Buffer Descriptor (RxBD) is set. See Section 18.6.2, “Ethernet Receive Buffer Descriptor (RxBD)” for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data LSB first.

18.4.8 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames is illustrated in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in Figure 18-2 illustrates the address recognition decisions made by the receive block, while Figure 18-3 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is deasserted, then the frame is accepted unconditionally, as shown in Figure 18-2. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 18-3.

If the DA is a group (multicast) address and flow control is disabled, the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

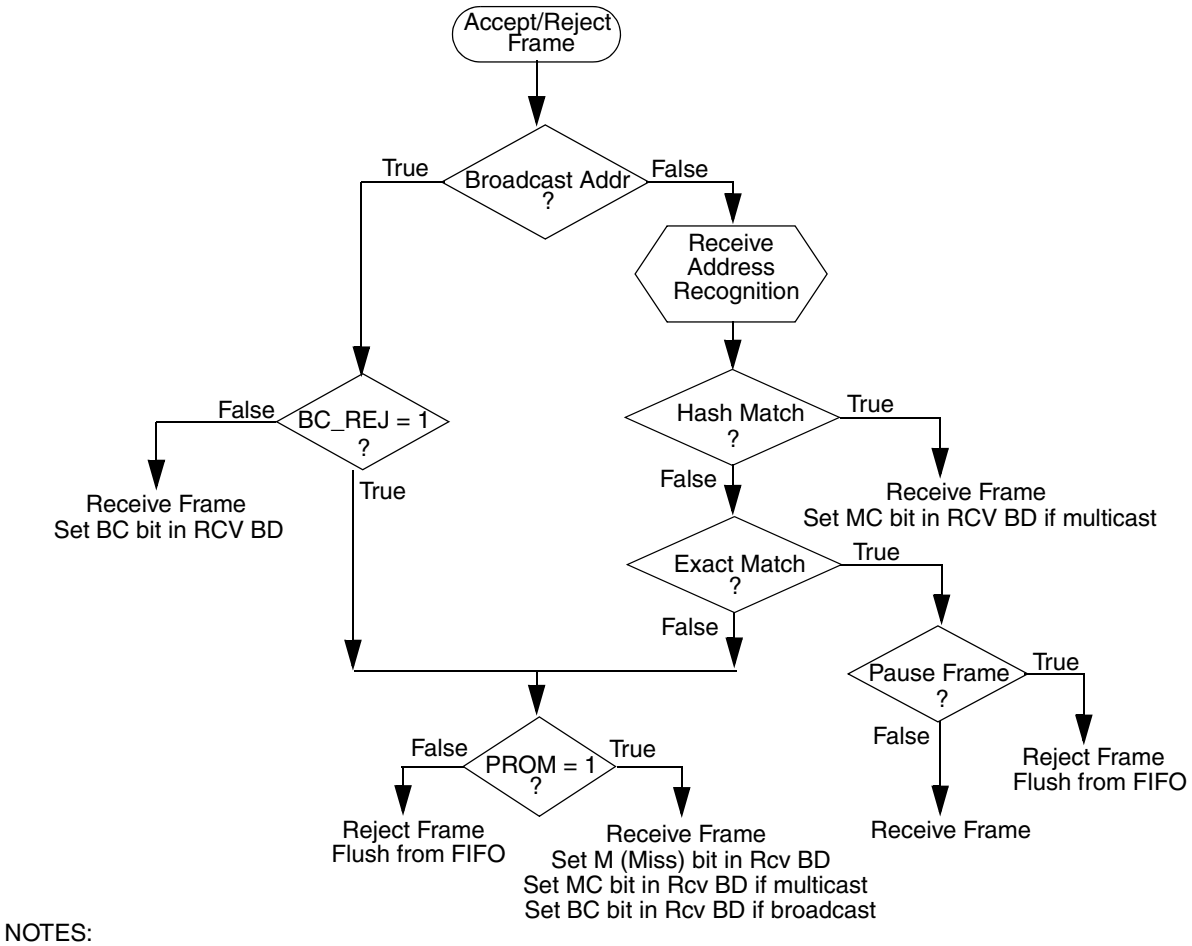
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines that the received frame is a valid PAUSE frame, the frame is rejected. The receiver detects a PAUSE frame with the DA field set to the designated PAUSE DA or the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or reject the frame based on PAUSE frame detection, shown in Figure 18-2.

If neither a hash match (group or individual) nor an exact match (group or individual) occur, then if promiscuous mode is enabled (RCR[PROM] = 1), the frame is accepted and the MISS bit in the receive buffer descriptor is set. Otherwise, the frame is rejected.

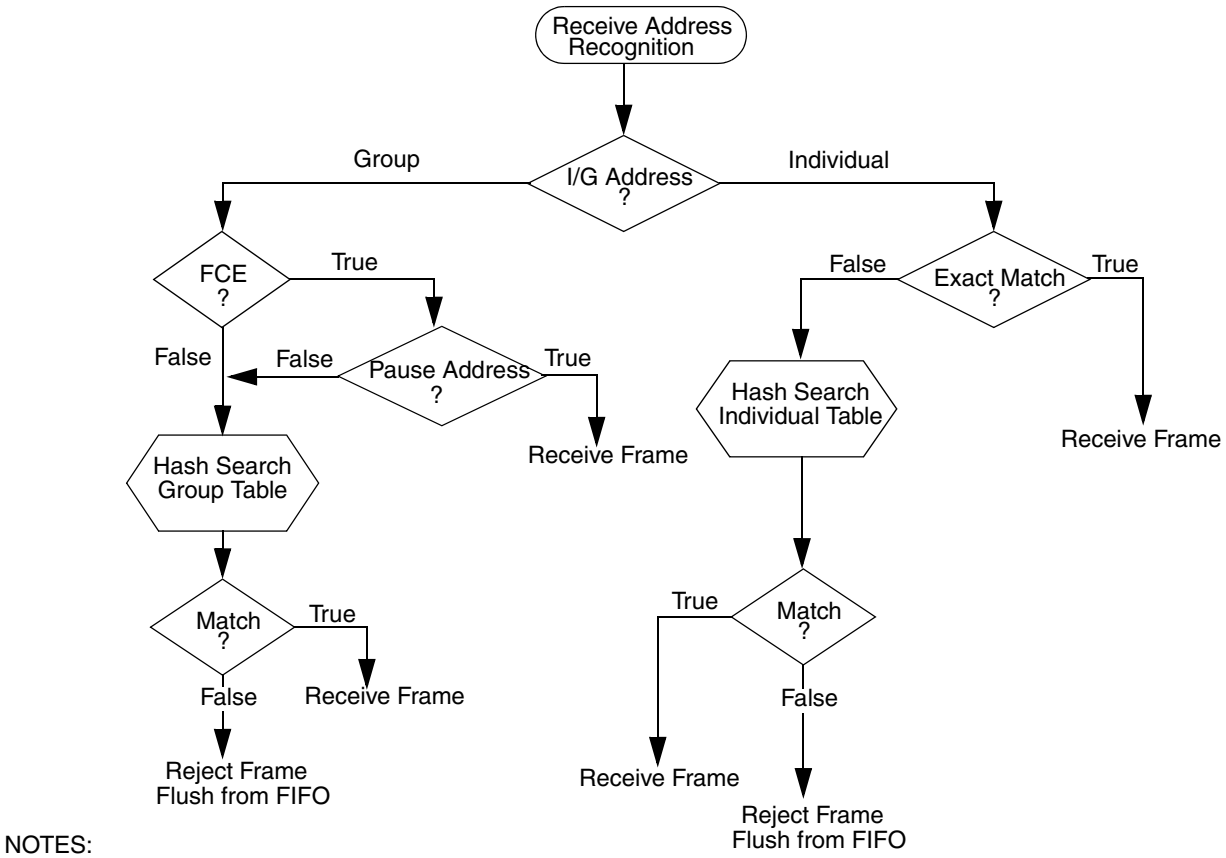
Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC_REJ]) is asserted, and promiscuous mode is enabled, the frame is accepted and the MISS bit in the receive buffer descriptor is set. Otherwise, the frame is rejected.

In general, when a frame is rejected, it is flushed from the FIFO.



NOTES:
BC_REJ - field in RCR register (BroadCast REJect)
PROM - field in RCR register (PROMiscous mode)
Pause Frame - valid PAUSE frame received

Figure 18-2. Ethernet Address Recognition—Receive Block Decisions



NOTES:
FCE - field in RCR register (Flow Control Enable)
I/G - Individual/Group bit in Destination Address (least significant bit in first byte received in MAC frame)

Figure 18-3. Ethernet Address Recognition—Microcode Decisions

18.4.9 Hash Algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, which are represented by 64 bits stored in GAUR, GALR (group address hash match) or IAUR, IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The MSB of the CRC result selects GAUR (MSB = 1) or GALR (MSB = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The hash table registers must be initialized by the user. The CRC32 polynomial to use in computing the hash is: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$.

A table of example Destination Addresses and corresponding hash values is included below for reference.

Table 18-7. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff:ff	0xa	10
8b:ff:ff:ff:ff:ff	0xb	11
0b:ff:ff:ff:ff:ff	0xc	12
3b:ff:ff:ff:ff:ff	0xd	13
7b:ff:ff:ff:ff:ff	0xe	14
5b:ff:ff:ff:ff:ff	0xf	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24
b9:ff:ff:ff:ff:ff	0x19	25
f9:ff:ff:ff:ff:ff	0x1a	26
c9:ff:ff:ff:ff:ff	0x1b	27
59:ff:ff:ff:ff:ff	0x1c	28

Table 18-7. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
79:ff:ff:ff:ff:ff	0x1d	29
29:ff:ff:ff:ff:ff	0x1e	30
19:ff:ff:ff:ff:ff	0x1f	31
d1:ff:ff:ff:ff:ff	0x20	32
f1:ff:ff:ff:ff:ff	0x21	33
b1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7f:ff:ff:ff:ff:ff	0x28	40
4f:ff:ff:ff:ff:ff	0x29	41
1f:ff:ff:ff:ff:ff	0x2a	42
3f:ff:ff:ff:ff:ff	0x2b	43
bf:ff:ff:ff:ff:ff	0x2c	44
9f:ff:ff:ff:ff:ff	0x2d	45
df:ff:ff:ff:ff:ff	0x2e	46
ef:ff:ff:ff:ff:ff	0x2f	47
93:ff:ff:ff:ff:ff	0x30	48
b3:ff:ff:ff:ff:ff	0x31	49
f3:ff:ff:ff:ff:ff	0x32	50
d3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3d:ff:ff:ff:ff:ff	0x38	56
0d:ff:ff:ff:ff:ff	0x39	57
5d:ff:ff:ff:ff:ff	0x3a	58
7d:ff:ff:ff:ff:ff	0x3b	59
fd:ff:ff:ff:ff:ff	0x3c	60

Table 18-7. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
dd:ff:ff:ff:ff:ff	0x3d	61
9d:ff:ff:ff:ff:ff	0x3e	62
bd:ff:ff:ff:ff:ff	0x3f	63

18.4.10 Full Duplex Flow Control

Full-duplex flow control allows the user to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] asserted) and flow control enable (RCR[FCE]) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in the table below. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 18-8. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_c200_0001 or Physical Address
48-bit Source Address	Any
16-bit Type	0x8808
16-bit Opcode	0x0001
16-bit PAUSE Duration	0x0000 to 0xFFFF

Pause frame detection is performed by the receiver and microcontroller modules. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR[GTS] is asserted by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. The pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is deasserted allowing MAC data frame transmission to resume. The receive flow control pause (TCR[RFC_PAUSE]) status bit is asserted while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and the user must assert flow control pause (TCR[TFC_PAUSE]). On assertion of transmit flow control pause (TCR[TFC_PAUSE]), the transmitter asserts TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts. Following EIR[GRA] assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause (TCR[TFC_PAUSE]) and TCR[GTS] are deasserted internally.

The user must specify the desired pause duration in the OPD register.

When the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause (TCR[TFC_PAUSE]) may be asserted and causes the transmission of a single pause frame. In this case, the EIR[GRA] interrupt is not asserted.

18.4.11 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the following frame may be discarded by the receiver.

18.4.12 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern is sent after the end of the preamble sequence.

If a collision occurs within 512 bit times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a Late Collision (LC) error indication.

18.4.13 Internal and External Loopback

Internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Internal and external loopback are configured using combinations of the LOOP and DRT bits in the RCR register and the FDEN bit in the TCR register.

For internal and external loopback, set FDEN equal to 1.

For internal loopback, set RCR[LOOP] equal to 1 and RCR[DRT] equal to 0. ETXEN and ETXER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the internal system clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMA’d to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback, set RCR[LOOP] and RCR[DRT] equal to 0 and configure the external transceiver for loopback.

18.4.14 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the EIR register, and the MIB block counters.

18.4.14.1 Transmission Errors

18.4.14.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame.

The UN interrupt is asserted if enabled in the EIMR register.

18.4.14.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the RL bit is set in the EIR. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame.

The RL interrupt is asserted if enabled in the EIMR register.

18.4.14.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the LC bit is set in the EIR register. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame.

The LC interrupt is asserted if enabled in the EIMR register.

18.4.14.1.4 Heartbeat

Some transceivers have a self-test feature called heartbeat or signal quality error. To signify a good self-test, the transceiver indicates a collision to the FEC within 4 microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the TCR register and the heartbeat condition is not detected by the FEC after a frame transmission, then a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets the HB bit in the EIR register, and generates the HBERR interrupt if it is enabled.

18.4.14.2 Reception Errors

18.4.14.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the RxBD. All subsequent data in the frame is discarded and subsequent frames may also be

discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.

18.4.14.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller handles up to seven dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, then the frame non-octet aligned (NO) error is reported in the RxBD. If there is no CRC error, then no error is reported.

18.4.14.2.3 CRC Error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

18.4.14.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt is generated, and the LG bit in the end of frame RxBD is set. The frame is not truncated unless the frame length exceeds 2047 bytes).

18.4.14.2.5 Truncation

When the receive frame length exceeds 2047 bytes the frame is truncated and the TR bit is set in the receive BD.

18.5 Programming Model

This section gives an overview of the registers, followed by a description of the buffers.

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control and to extract global status information. The descriptors are used to pass data buffers and related buffer information between the hardware and software.

18.5.1 Top Level Module Memory Map

The FEC implementation requires a 1-Kilobyte memory map space. This is divided into 2 sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. Table 18-9 defines the top level memory map.

Table 18-9. Module Memory Map

Address	Function
IPSBAR + 0x1000-11FF	Control/Status Registers
IPSBAR + 0x1200-12FF	MIB Block Counters

18.5.2 Detailed Memory Map (Control/Status Registers)

Table 18-10 shows the FEC register memory map with each register address, name, and a brief description.

Table 18-10. FEC Register Memory Map

IPSBAR Offset	Name	Width	Description
0x1004	EIR	32	Interrupt Event Register
0x1008	EIMR	32	Interrupt Mask Register
0x1010	RDAR	32	Receive Descriptor Active Register
0x1014	TDAR	32	Transmit Descriptor Active Register
0x1024	ECR	32	Ethernet Control Register
0x1040	MDATA	32	MII Data Register
0x1044	MSCR	32	MII Speed Control Register
0x1064	MIBC	32	MIB Control/Status Register
0x1084	RCR	32	Receive Control Register
0x10C4	TCR	32	Transmit Control Register
0x10E4	PALR	32	Physical Address Low Register
0x10E8	PAUR	32	Physical Address High+ Type Field
0x10EC	OPD	32	Opcode + Pause Duration
0x1118	IAUR	32	Upper 32 bits of Individual Hash Table
0x111C	IALR	32	Lower 32 Bits of Individual Hash Table
0x1120	GAUR	32	Upper 32 bits of Group Hash Table
0x1124	GALR	32	Lower 32 bits of Group Hash Table
0x1144	TFWR	32	Transmit FIFO Watermark
0x114C	FRBR	32	FIFO Receive Bound Register
0x1150	FRSR	32	FIFO Receive FIFO Start Registers
0x1180	ERDSR	32	Pointer to Receive Descriptor Ring
0x1184	ETDSR	32	Pointer to Transmit Descriptor Ring
0x1188	EMRBR	32	Maximum Receive Buffer Size

18.5.3 MIB Block Counters Memory Map

Table 18-11 defines the MIB Counters memory map which defines the locations in the MIB RAM space where hardware maintained counters reside. These fall in the 0x1200-0x13FF address offset range. The counters are divided into two groups.

RMON counters are included which cover the Ethernet Statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet Statistics group, a counter is included to count truncated frames as the FEC only supports frame lengths up to 2047 bytes. The RMON counters are implemented independently for transmit and receive to ensure accurate network statistics when operating in full duplex mode.

IEEE counters are included which support the Mandatory and Recommended counter packages defined in section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE Basic Package objects are supported by the FEC but do not require counters in the MIB block. In addition, some of the recommended package objects which are supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 18-11. MIB Counters Memory Map

IPSBAR Offset	Mnemonic	Description
0x1200	RMON_T_DROP	Count of frames not counted correctly
0x1204	RMON_T_PACKETS	RMON Tx packet count
0x1208	RMON_T_BC_PKT	RMON Tx Broadcast Packets
0x120C	RMON_T_MC_PKT	RMON Tx Multicast Packets
0x1210	RMON_T_CRC_ALIGN	RMON Tx Packets w CRC/Align error
0x1214	RMON_T_UNDERSIZE	RMON Tx Packets < 64 bytes, good crc
0x1218	RMON_T_OVERSIZE	RMON Tx Packets > MAX_FL bytes, good crc
0x121C	RMON_T_FRAG	RMON Tx Packets < 64 bytes, bad crc
0x1220	RMON_T_JAB	RMON Tx Packets > MAX_FL bytes, bad crc
0x1224	RMON_T_COL	RMON Tx collision count
0x1228	RMON_T_P64	RMON Tx 64 byte packets
0x122C	RMON_T_P65TO127	RMON Tx 65 to 127 byte packets
0x1230	RMON_T_P128TO255	RMON Tx 128 to 255 byte packets
0x1234	RMON_T_P256TO511	RMON Tx 256 to 511 byte packets
0x1238	RMON_T_P512TO1023	RMON Tx 512 to 1023 byte packets
0x123C	RMON_T_P1024TO2047	RMON Tx 1024 to 2047 byte packets
0x1240	RMON_T_P_GTE2048	RMON Tx packets w > 2048 bytes
0x1244	RMON_T_OCTETS	RMON Tx Octets
0x1248	IEEE_T_DROP	Count of frames not counted correctly
0x124C	IEEE_T_FRAME_OK	Frames Transmitted OK
0x1250	IEEE_T_1COL	Frames Transmitted with Single Collision
0x1254	IEEE_T_MCOL	Frames Transmitted with Multiple Collisions
0x1258	IEEE_T_DEF	Frames Transmitted after Deferral Delay
0x125c	IEEE_T_LCOL	Frames Transmitted with Late Collision
0x1260	IEEE_T_EXCOL	Frames Transmitted with Excessive Collisions
0x1264	IEEE_T_MACERR	Frames Transmitted with Tx FIFO Underrun
0x1268	IEEE_T_CSERR	Frames Transmitted with Carrier Sense Error
0x126C	IEEE_T_SQE	Frames Transmitted with SQE Error

Table 18-11. MIB Counters Memory Map (continued)

IPSBAR Offset	Mnemonic	Description
0x1270	IEEE_T_FDXFC	Flow Control Pause frames transmitted
0x1274	IEEE_T_OCTETS_OK	Octet count for Frames Transmitted w/o Error
0x1280	RMON_R_DROP	Count of frames not counted correctly
0x1284	RMON_R_PACKETS	RMON Rx packet count
0x1288	RMON_R_BC_PKT	RMON Rx Broadcast Packets
0x128C	RMON_R_MC_PKT	RMON Rx Multicast Packets
0x1290	RMON_R_CRC_ALIGN	RMON Rx Packets w CRC/Align error
0x1294	RMON_R_UNDERSIZE	RMON Rx Packets < 64 bytes, good crc
0x1298	RMON_R_OVERSIZE	RMON Rx Packets > MAX_FL bytes, good crc
0x129C	RMON_R_FRAG	RMON Rx Packets < 64 bytes, bad crc
0x12A0	RMON_R_JAB	RMON Rx Packets > MAX_FL bytes, bad crc
0x12A4	RMON_R_RESVD_0	
0x12A8	RMON_R_P64	RMON Rx 64 byte packets
0x12AC	RMON_R_P65TO127	RMON Rx 65 to 127 byte packets
0x12B0	RMON_R_P128TO255	RMON Rx 128 to 255 byte packets
0x12B4	RMON_R_P256TO511	RMON Rx 256 to 511 byte packets
0x12B8	RMON_R_P512TO1023	RMON Rx 512 to 1023 byte packets
0x12BC	RMON_R_P1024TO2047	RMON Rx 1024 to 2047 byte packets
0x12C0	RMON_R_P_GTE2048	RMON Rx packets w > 2048 bytes
0x12C4	RMON_R_OCTETS	RMON Rx Octets
0x12C8	IEEE_R_DROP	Count of frames not counted correctly
0x12CC	IEEE_R_FRAME_OK	Frames Received OK
0x12D0	IEEE_R_CRC	Frames Received with CRC Error
0x12D4	IEEE_R_ALIGN	Frames Received with Alignment Error
0x12D8	IEEE_R_MACERR	Receive Fifo Overflow count
0x12DC	IEEE_R_FDXFC	Flow Control Pause frames received
0x12E0	IEEE_R_OCTETS_OK	Octet count for Frames Rcvd w/o Error

18.5.4 Registers

The following sections describe each register in detail.

18.5.4.1 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the EIR, an interrupt is generated if the corresponding bit in the interrupt mask register (EIMR) is also set. The bit in the EIR is cleared if a one is written to that bit position; writing zero has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts because these errors are visible to network management via the MIB counters.

- HBERR - IEEE_T_SQE
- BABR - RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT - RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL - IEEE_T_LCOL
- COL_RETRY_LIM - IEEE_T_EXCOL
- XFIFO_UN - IEEE_T_MACERR

IPSBAR Access: User read/write

Offset: 0x1004 (EIR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EBERR	LC	RL	UN	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 18-4. Ethernet Interrupt Event Register (EIR)

Table 18-12. EIR Field Descriptions

Field	Description
31 HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR register and that the COL input was not asserted within the Heartbeat window following a transmission.
30 BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
29 BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR[MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffer(s). Truncation does not occur.

Table 18-12. EIR Field Descriptions (continued)

Field	Description
28 GRA	Graceful stop complete. This interrupt is asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. 1) A graceful stop, which was initiated by the setting of the TCR[GTS] bit is now complete. 2) A graceful stop, which was initiated by the setting of the TCR[TFC_PAUSE] bit is now complete. 3) A graceful stop, which was initiated by the reception of a valid full duplex flow control pause frame is now complete. Refer to the “Full Duplex Flow Control” section of the Functional Description chapter.
27 TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
26 TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.
25 RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.
24 RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
23 MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
22 EBERR	Ethernet bus error. This bit indicates that a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, ECR[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software needs to ensure that the FIFO controller and DMA are also soft reset.
21 LC	Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
20 RL	Collision retry limit. This bit indicates that a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame commences. Can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.
18–0	Reserved, should be cleared.

18.5.4.2 Interrupt Mask Register (EIMR)

The EIMR register controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared upon a hardware reset. If the corresponding bits in the EIR and EIMR registers are set, the interrupt is signalled to the CPU. The interrupt signal remains asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

IPSBAR Access: User read/write

Offset: 0x1008 (EIMR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HBERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EBERR	LC	RL	UN	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 18-5. Interrupt Mask Register (EIMR)

Table 18-13. EIMR Field Descriptions

Field	Description
31–19 (see Table 18-12 for the corresponding bit names)	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR samples the signal generated by the interrupting source. The corresponding EIR bit reflects the state of the interrupt signal even if the corresponding EIMR bit is set. 0 The corresponding interrupt source (see Table 18-12) is masked. 1 The corresponding interrupt source (see Table 18-12) is not masked.
18–0	Reserved, should be cleared.

18.5.4.3 Receive Descriptor Active Register (RDAR)

RDAR is a command register that indicates that the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and process receive frames (provided ECR[ETHER_EN] is also set). After the FEC polls a receive descriptor whose empty bit is not set, then the FEC clears the RDAR bit and cease receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors have been placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER_EN] is cleared.

IPSBAR Access: User read/write

Offset: 0x1010 (RDAR)

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

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

Figure 18-6. Receive Descriptor Active Register (RDAR)

Table 18-14. RDAR Field Descriptions

Field	Description
31–25	Reserved, should be cleared.
24 R_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device when no additional empty descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.
23–0	Reserved, should be cleared.

18.5.4.4 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register which should be written by the user to indicate that the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC polls the transmit descriptor ring and process transmit frames (provided ECR[ETHER_EN] is also set). After the FEC polls a transmit descriptor whose ready bit is not set, then the FEC clears the TDAR bit and cease transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

IPSBAR Access: User read/write

Offset: 0x1014 (TDAR)

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

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

Figure 18-7. Transmit Descriptor Active Register (TDAR)

Table 18-15. TDAR Field Descriptions

Field	Description
31–25	Reserved, should be cleared.
24 X_DES_ACTIVE	Set to one when this register is written, regardless of the value written. Cleared by the FEC device when no additional ready descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.
23–0	Reserved, should be cleared.

18.5.4.5 Ethernet Control Register (ECR)

ECR is a read/write user register, though both fields in this register may be altered by hardware as well. The ECR is used to enable/disable the FEC.

IPSBAR Access: User read/write

Offset: 0x1024 (ECR)

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

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

Figure 18-8. Ethernet Control Register (ECR)

Table 18-16. ECR Field Descriptions

Field	Description
1 ETHER_EN	When this bit is set, the FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any currently transmitted frame. The buffer descriptor(s) for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is deasserted, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions: • ECR[RESET] is set by software, in which case ETHER_EN is cleared • An error condition causes the EIR[EBERR] bit to set, in which case ETHER_EN is cleared
0 RESET	When this bit is set, a hardware reset of the FEC is performed. ETHER_EN is cleared and all other FEC registers are initialized to their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately 8 system clock cycles after RESET is written with a 1.

NOTE

The device may be operated with an external Ethernet PHY. However, due to the shared pin groups, the use of an external PHY limits ADC, interrupt, and QSPI functionality, and disables the UART0/1 and 16-bit timers (see Table 2-1).

18.5.4.6 MII Management Frame Register (MMFR)

The MMFR is accessed by the user and does not reset to a defined value. The MMFR register is used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR causes a management frame to be sourced unless the MSCR has been programmed to 0. In the case of writing to MMFR when MSCR equals 0, if the MSCR register is then written to a non-zero value, an MII frame is generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

Before accessing the MII registers via the MMFR, the software must poll EIR[MII] to make sure that an access is not currently in progress.

IPSBAR Access: User read/write
Offset: 0x1040 (MMFR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	ST		OP		PA					RA					TA	
Reset	Undefined															

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

Figure 18-9. MII Management Frame Register (MMFR)

Table 18-17. MMFR Field Descriptions

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
29–28 OP	Operation code. This field must be programmed to 10 (read) or 01 (write) to generate a valid MII management frame. A value of 11 produces read frame operation while a value of 00 produces write frame operation, but these frames are not MII compliant.
27–23 PA	PHY address. This field specifies one of up to 32 attached PHY devices.
22–18 RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.
17–16 TA	Turn around. This field must be set to 10 to generate a valid MII management frame.
15–0 DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

To perform a read or write operation on the MII Management Interface, the MMFR register must be written by the user. To generate a valid read or write management frame, the ST field must be written with

a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated but does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. After the write management frame operation has completed, the MII interrupt is generated. At this time, the contents of the MMFR register match the original value written.

To generate an MII Management Interface read frame (read a PHY register) the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a don't care). Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, the contents of the MMFR register are altered as the contents are serially shifted and are unpredictable if read by the user. After the read management frame operation has completed, the MII interrupt is generated. At this time, the contents of the MMFR register match the original value written except for the DATA field whose contents have been replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents are altered. Software should use the MII_STATUS register and/or the MII interrupt to avoid writing to the MMFR register while frame generation is in progress.

18.5.4.7 MII Speed Control Register (MSCR)

The MSCR provides control of the MII clock (EMDC pin) frequency, allows a preamble drop on the MII management frame, and provides observability (intended for manufacturing test) of an internal counter used in generating the EMDC clock signal.

IPSBAR Access: User read/write

Offset: 0x1044 (MSCR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 18-10. MII Speed Control Register (MSCR)

Table 18-18. MSCR Field Descriptions

Field	Description
31–8	Reserved, should be cleared.
7 DIS_PREAMBLE	Asserting this bit causes preamble (32 1's) not to be prefixed to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
6–1 MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (EMDC) relative to the system clock. A value of 0 in this field turns off the EMDC and leave it in low voltage state. Any non-zero value results in the EMDC frequency of 1/(MII_SPEED×2) of the system clock frequency.
0	Reserved, should be cleared.

The MII_SPEED field must be programmed with a value to provide an EMDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value to source a read or write management frame. After the management frame is complete the MSCR register may optionally be set to zero to turn off the EMDC. The EMDC generated has a 50% duty cycle except when MII_SPEED is changed during operation (change takes effect following a rising or falling edge of EMDC).

If the system clock is 25 MHz, programming this register to 0x0000_0005 results in an EMDC frequency of 25 MHz × 1/ (5 × 2) = 2.5 MHz. A table showing optimum values for MII_SPEED as a function of system clock frequency is provided below.

Table 18-19. Programming Examples for MSCR

System Clock Frequency	MII_SPEED (field in reg)	EMDC frequency
25 MHz	0x5	2.5 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.5 MHz
50 MHz	0xA	2.5 MHz
60 MHz	0xD	2.5 MHz

18.5.4.8 MIB Control Register (MIBC)

The MIBC is a read/write register used to provide control of and to observe the state of the MIB block. This register is accessed by user software if there is a need to disable the MIB block operation. For example, to clear all MIB counters in RAM the user should disable the MIB block, then clear all the MIB RAM locations, then enable the MIB block. The MIB_DISABLE bit is reset to 1. See Table 18-11 for the locations of the MIB counters.

IPSBAR Access: User read/write

Offset: 0x1064 (MIBC)

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

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

Figure 18-11. MIB Control Register (MIBC)

Table 18-20. MIBC Field Descriptions

Field	Description
31 MIB_DISABLE	A read/write control bit. If set, the MIB logic halts and does not update any MIB counters.
30 MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.
29–0	Reserved.

18.5.4.9 Receive Control Register (RCR)

The RCR is programmed by the user. The RCR controls the operational mode of the receive block and should be written only when ECR[ETHER_EN] equals 0 (initialization time).

IPSBAR Access: User read/write

Offset: 0x1084 (RCR)

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 18-12. Receive Control Register (RCR)

Table 18-21. RCR Field Descriptions

Field	Description
31–27	Reserved, should be cleared.
26–16 MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes the BABT interrupt to occur. Receive Frames longer than MAX_FL causes the BABR interrupt to occur and sets the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed by the user is 1518 or 1522 (if VLAN Tags are supported).
15–6	Reserved, should be cleared.
5 FCE	Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
4 BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = FF_FF_FF_FF_FF_FF are be rejected unless the PROM bit is set. If BC_REJ and PROM = 1, frames with broadcast DA are accepted and the M (MISS) bit is set in the receive buffer descriptor.
3 PROM	Promiscuous mode. All frames are accepted regardless of address matching.
2 MII_MODE	Media independent interface mode. Selects external interface mode. Setting this bit to one selects MII mode, setting this bit equal to zero selects 7-wire mode (used only for serial 10 Mbps). This bit controls the interface mode for transmit and receive blocks.
1 DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
0 LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The system clock is substituted for the ETXCLK when LOOP is asserted. DRT must be set to zero when asserting LOOP.

18.5.4.10 Transmit Control Register (TCR)

The TCR is read/write and is written by the user to configure the transmit block. This register is cleared at system reset. Bits 2 and 1 should be modified only when ECR[ETHER_EN] equals 0.

IPSBAR Access: User read/write

Offset: 0x10C4 (TCR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	RFC_PAUSE	TFC_PAUSE	FDEN	HBC	GTS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-13. Transmit Control Register (TCR)

Table 18-22. TCR Field Descriptions

Field	Description
31–5	Reserved, should be cleared.
4 RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmission of data frames after the current transmission is complete. At this time, the GRA interrupt in the EIR register is asserted. With transmission of data frames stopped, the MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resume transmitting data frames. If the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may continue transmitting a MAC Control PAUSE frame.
2 FDEN	Full duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should only be modified when ETHER_EN is deasserted.
1 HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ETHER_EN is deasserted.
0 GTS	Graceful transmit stop. When this bit is set, the MAC stops transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the EIR register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission has completed, a restart can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS equals 1, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that are transmitted when GTS is reasserted. To avoid this deassert ECR[ETHER_EN] following the GRA interrupt.

18.5.4.11 Physical Address Low Register (PALR)

The PALR is written by the user. This register contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the DA (Destination Address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte source address field when transmitting PAUSE frames. This register is not reset and must be initialized by the user.

IPSBAR Access: User read/write

Offset: 0x10E4 (PALR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PADDR1															
Reset	Undefined															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PADDR1															
Reset	Undefined															

Figure 18-14. Physical Address Low Register (PALR)

Table 18-23. PALR Field Descriptions

Field	Description
31–0 PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8) and 3 (bits 7:0) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.

18.5.4.12 Physical Address High Register (PAUR)

The PAUR is written by the user. This register contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR contain a constant type field (0x8808) used for transmission of PAUSE frames. This register is not reset and bits 31:16 must be initialized by the user.

IPSBAR Access: User read/write
Offset: 0x10E8 (PAUR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PADDR2															
Reset	Undefined															

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

Figure 18-15. Physical Address High Register (PAUR)

Table 18-24. PAUR Field Descriptions

Field	Description
31–16 PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
15–0 TYPE	Type field in PAUSE frames. These 16 bits are a constant value of 0x8808.

18.5.4.13 Opcode/Pause Duration Register (OPD)

The OPD is read/write accessible. This register contains the 16-bit opcode and 16-bit pause duration fields used in transmission of a PAUSE frame. The opcode field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. This register is not reset and must be initialized by the user.

IPSBAR Access: User read/write
Offset: 0x10EC (OPD)

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

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

Figure 18-16. Opcode/Pause Duration Register (OPD)

Table 18-25. OPD Field Descriptions

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause Duration field used in PAUSE frames.

18.5.4.14 Descriptor Individual Upper Address Register (IAUR)

The IAUR is written by the user. This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

IPSBAR Access: User read/write
Offset: 0x1118 (IAUR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IADDR1															
Reset	Undefined															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IADDR1															
Reset	Undefined															

Figure 18-17. Descriptor Individual Upper Address Register (IAUR)

Table 18-26. IAUR Field Descriptions

Field	Description
31–0 IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

18.5.4.15 Descriptor Individual Lower Address (IALR)

The IALR register is written by the user. This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

IPSBAR Access: User read/write
Offset: 0x111C (IALR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IADDR2															
Reset	Undefined															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IADDR2															
Reset	Undefined															

Figure 18-18. Descriptor Individual Lower Address Register (IALR)

Table 18-27. IALR Field Descriptions

Field	Description
31–0 IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

18.5.4.16 Descriptor Group Upper Address (GAUR)

The GAUR is written by the user. This register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

IPSBAR
Offset: 0x1120 (GAUR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	GADDR1															
Reset	Undefined															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	GADDR1															
Reset	Undefined															

Figure 18-19. Descriptor Group Upper Address Register (GAUR)

Table 18-28. GAUR Field Descriptions

Field	Description
31–0 GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

18.5.4.17 Descriptor Group Lower Address (GALR)

The GALR register is written by the user. This register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

IPSBAR Access: User read/write
Offset: 0x1124 (GALR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	GADDR2															
Reset	Undefined															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	GADDR2															
Reset	Undefined															

Figure 18-20. Descriptor Group Lower Address Register (GALR)

Table 18-29. GALR Field Descriptions

Field	Description
31–0 GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

18.5.4.18 FIFO Transmit FIFO Watermark Register (TFWR)

The TFWR is programmed by the user to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user to minimize transmit latency (TFWR = 0x) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

IPSBAR Access: User read/write

Offset: 0x1144 (TFWR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 18-21. FIFO Transmit FIFO Watermark Register (TFWR)

Table 18-30. TFWR Field Descriptions

Field	Description
31–2	Reserved, should be cleared.
1–0 X_WMRK	Number of bytes written to transmit FIFO before transmission of a frame begins 0x 64 bytes written 10 128 bytes written 11 192 bytes written

18.5.4.19 FIFO Receive Bound Register (FRBR)

The FRBR is an 8-bit register that the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR to appropriately divide the available FIFO RAM between the transmit and receive data paths.

IPSBAR Access: User read/write

Offset: 0x114C (FRBR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 18-22. FIFO Receive Bound Register (FRBR)

Table 18-31. FRBR Field Descriptions

Field	Description
31–10	Reserved, read as 0 (except bit 10, which is read as 1).
9–2 R_BOUND	Read-only. Highest valid FIFO RAM address.
1–0	Reserved, should be cleared.

18.5.4.20 FIFO Receive Start Register (FRSR)

The FRSR is an 8-bit register programmed by the user to indicate the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR. The receive FIFO uses addresses from FRSR to FRBR inclusive.

The FRSR register is initialized by hardware at reset. FRSR only needs to be written to change the default value.

IPSBAR Access: User read/write

Offset: 0x1150 (FRSR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

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

Figure 18-23. FIFO Receive Start Register (FRSR)

Table 18-32. FRSR Field Description

Field	Description
31–10	Reserved, read as 0 (except bit 10, which is read as 1).
9–2 R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs.
1–0	Reserved, read as 0.

18.5.4.21 Receive Descriptor Ring Start (ERDSR)

The ERDSR is written by the user. It provides a pointer to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized by the user prior to operation.

IPSBAR Access: User read/write
Offset: 0x1180 (ERDSR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	R_DES_START															
W																
Reset	Undefined															

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

Figure 18-24. Receive Descriptor Ring Start Register (ERDSR)

Table 18-33. ERDSR Field Descriptions

Field	Description
31–2 R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, should be cleared.

18.5.4.22 Transmit Buffer Descriptor Ring Start (ETSDR)

The ETSDR is written by the user. It provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). Bits 1 and 0 should be written to 0 by the user. Non-zero values in these two bit positions are ignored by the hardware.

This register is not reset and must be initialized by the user prior to operation.

IPSBAR Access: User read/write
Offset: 0x1184 (ETSDR)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	X_DES_START															
Reset	Undefined															

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

Figure 18-25. Transmit Buffer Descriptor Ring Start Register (ETDSR)

Table 18-34. ETDSR Field Descriptions

Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, should be cleared.

18.5.4.23 Receive Buffer Size Register (EMRBR)

The EMRBR is a 9-bit register programmed by the user. The EMRBR register dictates the maximum size of all receive buffers. Because receive frames are truncated at 2k-1 bytes, only bits 10–4 are used. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR must be set to RCR[MAX_FL] or larger. The EMRBR must be evenly divisible by 16. To ensure this, bits 3-0 are forced low. To minimize bus utilization (descriptor fetches) it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register does not reset, and must be initialized by the user.

IPSBAR Access: User read/write

Offset: 0x1188 (EMRBR)

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

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

Figure 18-26. Receive Buffer Size Register (EMRBR)

Table 18-35. EMRBR Field Descriptions

Field	Description
30–11	Reserved, should be cleared.
10–4 R_BUF_SIZE	Receive buffer size.
3–0	Reserved, should be cleared.

18.6 Buffer Descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

18.6.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. Associated with each buffer is a buffer descriptor (BD) which contains a starting address (pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read in by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit produces the buffer. Software writing to the TDAR or RDAR tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the RxBD[E] or TxBD[R] bit is cleared by hardware to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ECR[ETHER_EN] signal operates as a reset to the BD/DMA logic. When ECR[ETHER_EN] is deasserted the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before the ECR[ETHER_EN] bit is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the Wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

18.6.1.1 Driver/DMA Operation with Transmit BDs

Typically a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a 2nd buffer, IP header in a 3rd buffer, Ethernet/IEEE 802.3 header in a 4th buffer. The Ethernet MAC does not prefix the Ethernet header (destination address, source address, length/type field(s)), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD which must be set by the driver.

The driver (TxBD software producer) should set up Tx BDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the TxBD[R] bits should be set equal to 1 in reverse order (3rd, 2nd, 1st BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the 2nd was made available, potentially causing a transmit FIFO underrun.

In the FEC, the DMA is notified by the driver that new transmit frame(s) are available by writing to the TDAR register. When this register is written to (data value is not significant) the FEC RISC tells the DMA to read the next transmit BD in the ring. After started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers, until a transmit BD is encountered with the R bit equaling 0. At this point the FEC polls this BD one more time. If the R bit equals 0 the second time, the RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

18.6.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L (1 indicates last buffer in frame) bit, the frame status bits (if L = 1) and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end of frame buffers the receive BD is written with L equaling 1 and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame.

For simplicity the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out of spec implementation could result in giant frames. Frames of 2k (2048) bytes or larger are truncated by the FEC at 2047 bytes so software is guaranteed never to see a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received the FEC fills receive buffers and update the associated BDs, then read the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit equals 0, it polls this BD once more. If the BD equals 0 a second time the FEC stops reading receive BDs until the driver writes to RDAR.

18.6.2 Ethernet Receive Buffer Descriptor (RxBD)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in second longword. When the buffer has been DMA'd, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and write the length of the used portion of the buffer in the first longword. The M, BC,

MC, LG, NO, CR, OV and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

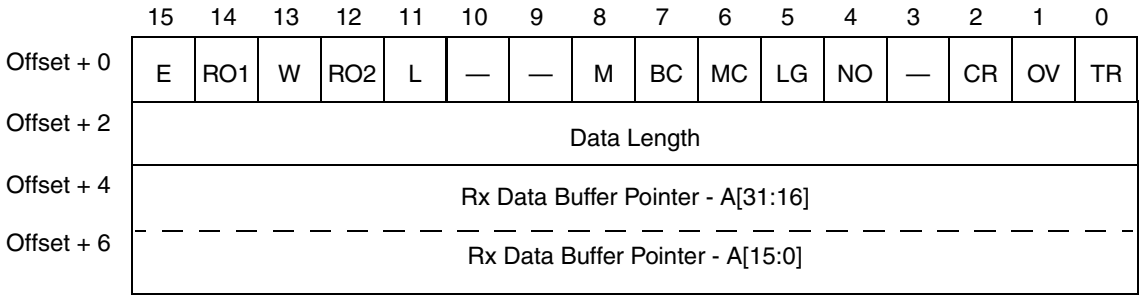


Figure 18-27. Receive Buffer Descriptor (RxBD)

Table 18-36. Receive Buffer Descriptor Field Definitions

Word	Location	Field Name	Description
Offset + 0	Bit 15	E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data, or data reception has been aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	Bit 14	RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	Bit 13	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ERDSR.
Offset + 0	Bit 12	RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	Bit 11	L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	Bits 10–9	—	Reserved.
Offset + 0	Bit 8	M	Miss. Written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a miss by the internal address recognition. Thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	Bit 7	BC	Set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	Bit 6	MC	Set if the DA is multicast and not BC.

Table 18-36. Receive Buffer Descriptor Field Definitions (continued)

Word	Location	Field Name	Description
Offset + 0	Bit 5	LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	Bit 4	NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. If this bit is set, the CR bit is not set.
Offset + 0	Bit 3	—	Reserved.
Offset + 0	Bit 2	CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	Bit 1	OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and is zero. This bit is valid only if the L-bit is set.
Offset + 0	Bit 0	TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set the frame should be discarded and the other error bits should be ignored as they may be incorrect.
Offset + 2	Bits [15:0]	Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD's data buffer if L equals 0 (the value is equal to EMRBR) or the length of the frame, including CRC if L equals 1. It is written by the FEC once as the BD is closed.
0ffset + 4	Bits [15:0]	A[31:16]	RX data buffer pointer, bits [31:16][1]
Offset + 6	Bits [15:0]	A[15:0]	RX data buffer pointer, bits [15:0]

[1] The receive buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

When the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDAR.

18.6.3 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (R bit) when DMA of the buffer is complete. In the TxBD the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword, and the buffer pointer in the second longword.

The FEC sets the R bit equal to 0 in the first longword of the BD when the buffer has been DMA'd. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See Section 18.5.3, "MIB Block Counters Memory Map" for more details.

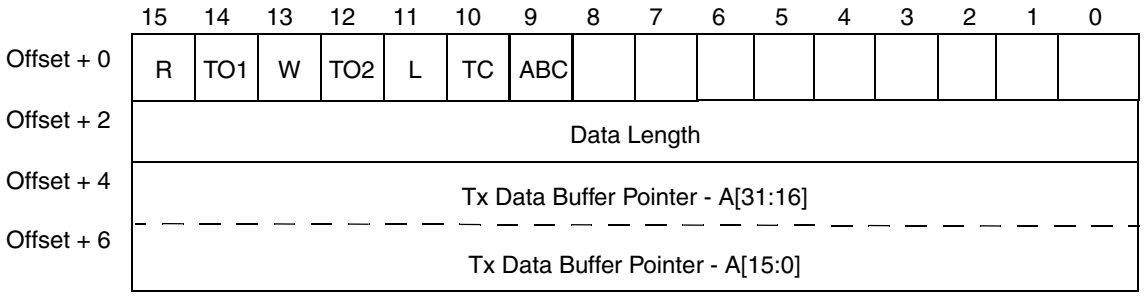


Figure 18-28. Transmit Buffer Descriptor (TxBD)

Table 18-37. Transmit Buffer Descriptor Field Definitions

Word	Location	Field Name	Description
Offset + 0	Bit 15	R	Ready. Written by the FEC and the user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD may be written by the user after this bit is set.
Offset + 0	Bit 14	TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	Bit 13	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.
Offset + 0	BIt 12	TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	Bit 11	L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.
Offset + 0	Bit 10	TC	Tx CRC. Written by user (only valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
Offset + 0	Bit 9	ABC	Append bad CRC. Written by user (only valid if L = 1). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value).
Offset + 0	Bits [8:0]	—	Reserved.
Offset + 2	Bits [15:0]	Data Length	Data Length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [15:5] are used by the DMA engine, bits[4:0] are ignored.

Table 18-37. Transmit Buffer Descriptor Field Definitions (continued)

Word	Location	Field Name	Description
Offset + 4	Bits [15:0]	A[31:16]	Tx data buffer pointer, bits [31:16][1]
Offset + 6	Bits [15:0]	A[15:0]	Tx data buffer pointer, bits [15:0].

[1] The transmit buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

After the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the R bit in the first BD for the frame. The driver should follow that with a write to TDAR which triggers the FEC to poll the next BD in the ring.

Chapter 19
Ethernet Physical Transceiver (EPHY)

19.1 Introduction

The Ethernet physical transceiver (Ethernet physical interface) is an IEEE 802.3 compliant 10BASE-T/100BASE-TX Ethernet PHY transceiver. The Ethernet physical interface module supports the medium-independent interface (MII) and the MII management interface. The EPHY requires a 25-MHz crystal for its basic operation.

19.1.1 Features

- IEEE 802.3 compliant
- Full-/half-duplex support in all modes
- Medium-independent interface (MII), which has these characteristics:
  - Capable of supporting 10 Mbps and 100 Mbps data rates
  - Data and delimiters are synchronous to clock references
  - Provides independent four-bit wide transmit and receive data paths
  - Provides a simple management interface
- Supports auto-negotiation
- Auto-negotiation next page ability
- Single RJ45 connection
- 1:1 common transformer
- Baseline wander correction
- Digital adaptive equalization
- Integrated wave-shaping circuitry
- Far-end fault detect
- MDC rates up to 25 MHz
- Supports MDIO preamble suppression
- Jumbo packet
- 2.5 V CMOS
- 2.5 V MII interface
- 125 MHz clock generator and timing recovery
- Loopback modes

19.1.2 Block Diagram

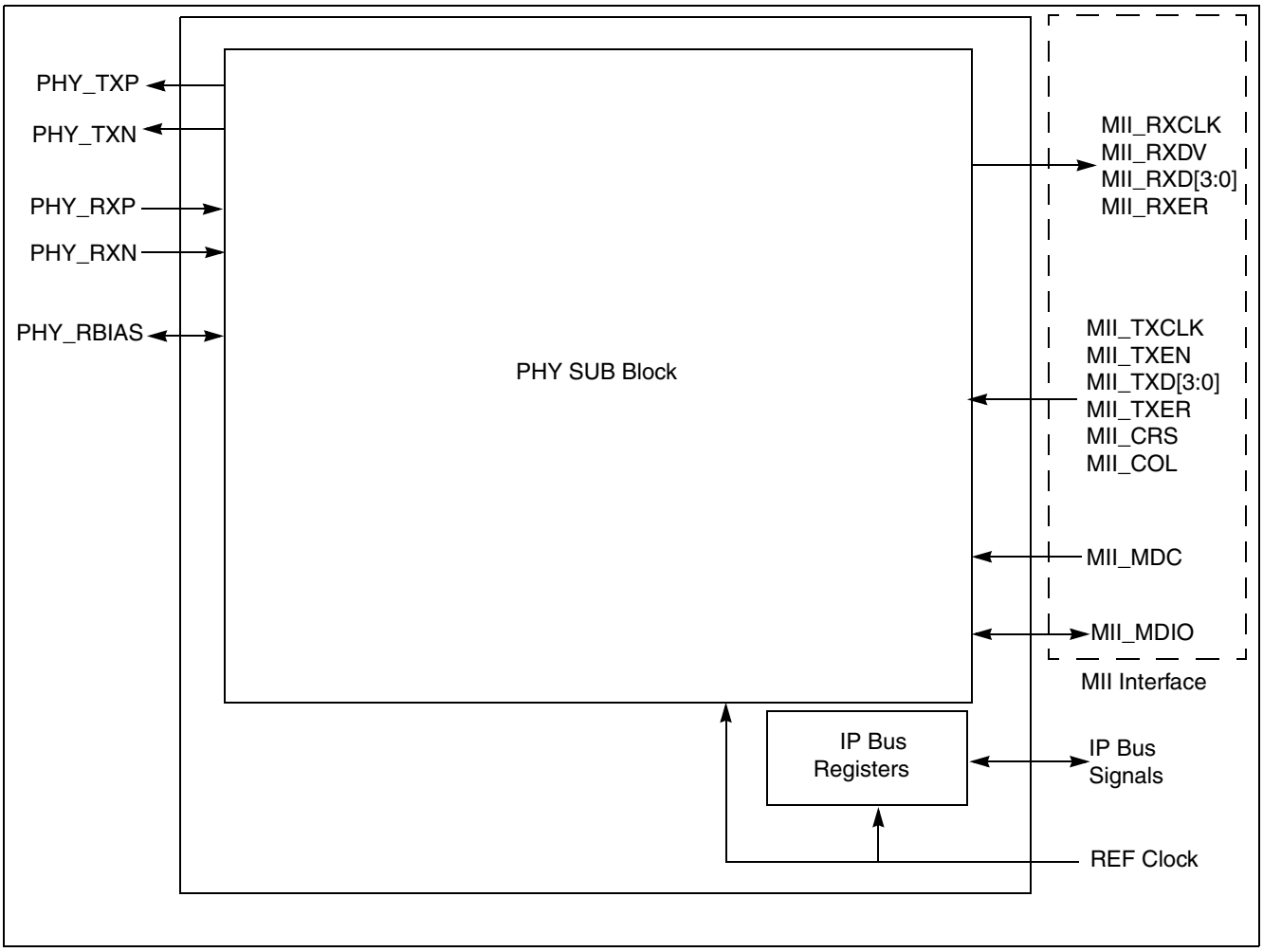


Figure 19-1. Ethernet Physical Transceiver (EPHY) Block Diagram

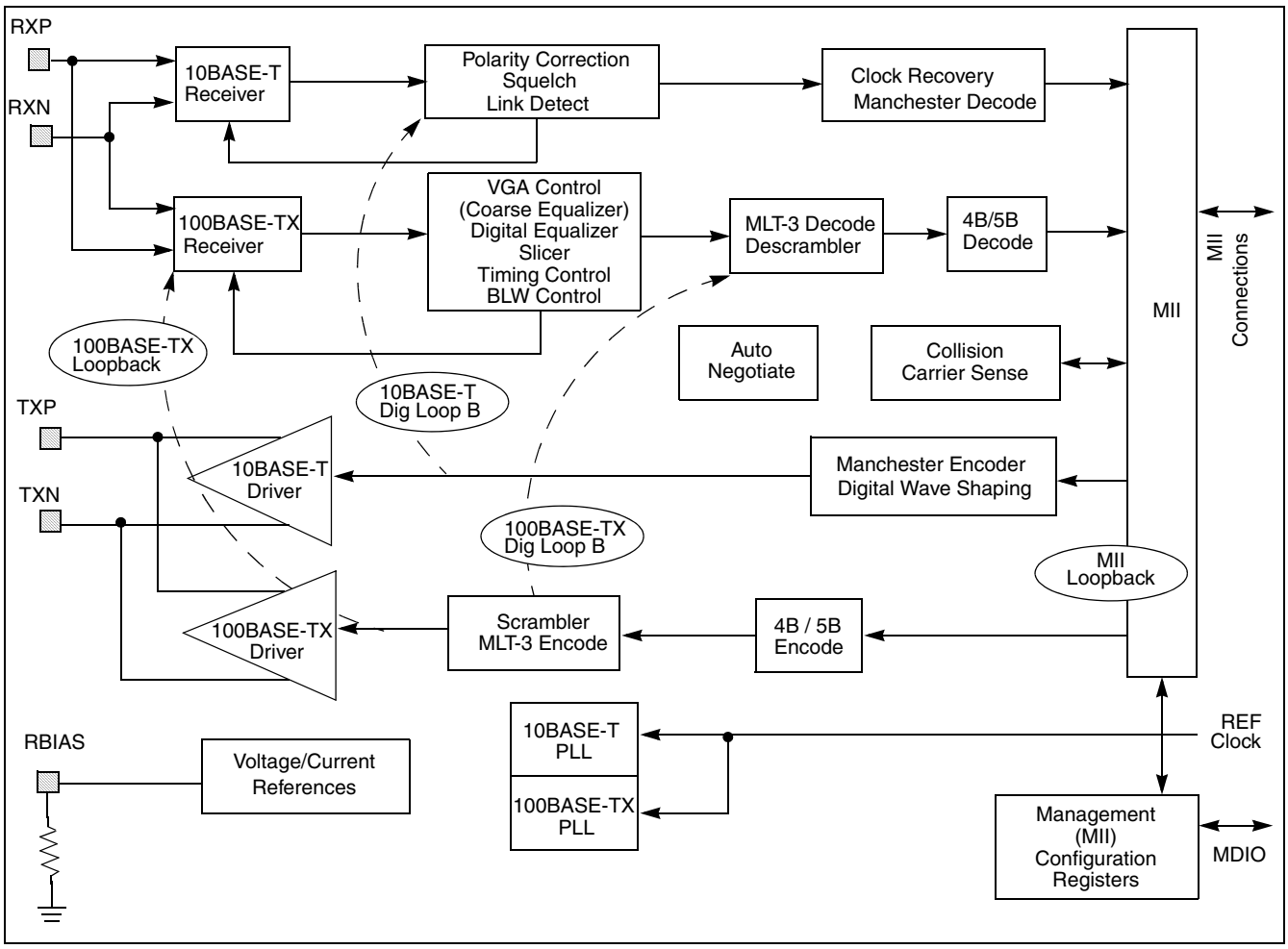


Figure 19-2. PHY Sub Block Diagram

19.2 External Signal Descriptions

This section contains the EPHY external pin descriptions.

19.2.1 PHY_TXP — EPHY Twisted Pair Output +

Ethernet twisted-pair output pin. This pin is high-impedance out of reset.

19.2.2 PHY_TXN — EPHY Twisted Pair Output –

Ethernet twisted-pair output pin. This pin is high-impedance out of reset.

19.2.3 PHY_RXP — EPHY Twisted Pair Input +

Ethernet twisted-pair input pin. This pin is high-impedance out of reset.

19.2.4 PHY_RXN — EPHY Twisted Pair Input –

Ethernet twisted-pair input pin. This pin is high-impedance out of reset.

19.2.5 PHY_RBIAS — EPHY Bias Control Resistor

Connect a 1.0% external resistor, RBIAS (see Electrical Characteristics chapter), between the PHY_RBIAS pin and analog ground. Place this resistor as near to the chip pin as possible. Stray capacitance must be kept to less than 10 pF (>50 pF causes instability). No high-speed signals are permitted in the region of RBIAS.

19.2.6 PHY_VDDRX, PHY_VSSRX — Power Supply Pins for EPHY Receiver

Power is supplied to the EPHY receiver through PHY_VDDRX and PHY_VSSRX. This 2.5 V supply is derived from the internal voltage regulator. There is no static load on those pins allowed. The internal voltage regulator is turned off, if V_{DDR} is tied to ground.

19.2.7 PHY_VDDTX, PHY_VSSTX — Power Supply Pins for EPHY Transmitter

External power is supplied to the EPHY transmitter through PHY_VDDTX and PHY_VSSTX. This 2.5 V supply is derived from the internal voltage regulator. There is no static load on those pins allowed. The internal voltage regulator is turned off, if V_{DDR} is tied to ground.

19.2.8 PHY_VDDA, PHY_VSSA — Power Supply Pins for EPHY Analog

Power is supplied to the EPHY PLLs through PHY_VDDA and PHY_VSSA. This 2.5 V supply is derived from the internal voltage regulator. There is no static load on those pins allowed. The internal voltage regulator is turned off, if V_{DDR} is tied to ground.

19.2.9 COLLED — Collision LED

Flashes when a collision occurs on a network in half duplex mode if EPHYCTL0 LEDEN bit is set.

19.2.10 DUPLED — Duplex LED

Indicates the duplex of the link, which can be full-duplex or half-duplex if EPHYCTL0 LEDEN bit is set.

19.2.11 SPDLED — Speed LED

Indicates the speed of a link, which can be 10 Mbps or 100 Mbps if EPHYCTL0 LEDEN bit is set.

19.2.12 LNKLED — Link LED

Indicates whether a link is established with another network device if EPHYCTL0 LEDEN bit is set.

19.2.13 ACTLEC — Activity LED

Flashes when data is received by the device if EPHYCTL0 LEDEN bit is set.

19.3 Memory Map and Register Descriptions

This section provides a detailed description of all registers accessible in the Ethernet physical interface.

19.3.1 Module Memory Map

Table 19-1 gives an overview of all registers in the Ethernet physical interface memory map. The Ethernet physical interface occupies 48 bytes in the memory space.

Table 19-1. EPHY Module Memory Map

IPSBAR Offset	Use	Access
0x1E_0000	Ethernet Physical Transceiver Control Register 0 (EPHYCTL0)	R/W
0x1E_0001	Ethernet Physical Transceiver Control Register 1 (EPHYCTL1)	R/W
0x1E_0002	Ethernet Physical Transceiver Status Register (EPHYSR)	R/W
0x1E_0003	RESERVED	R

19.3.2 Register Descriptions

19.3.2.1 Ethernet Physical Transceiver Control Register 0 (EPHYCTL0)

IPSBAR Access: User read/write

Offset: 0x1E_0000 (EPHYCTL0)

	7	6	5	4	3	2	1	0
R	EPHYEN	ANDIS	DIS100	DIS10	LEDEN	EPHYWAI	0	EPHYIEN
W								
Reset	0	1	1	1	0	0	0	0

Figure 19-3. Ethernet Physical Transceiver Control Register 0 (EPHYCTL0)

Table 19-2. EPHYCTL0 Field Descriptions

Field	Description
7 EPHYEN	EPHY Enable. This bit can be written anytime. 1 Enables EPHY 0 Disables EPHY
6 ANDIS	Auto Negotiation Disable. This bit can be written anytime, but the value is latched in the ANE bit of the MII PHY control register (MII address 0.12) only when the EPHYEN bit transitions from 0 to 1. 1 Auto negotiation is disabled after start-up. A 0 is latched in the ANE bit of the MII PHY control register (MII address 0.12), and upon completion of the start-up delay ($t_{Start-up}$), the EPHY bypasses auto-negotiation. The mode of operation is determined by the manual setting of MII registers. 0 Auto negotiation is enabled after start-up. A 1 is latched in the ANE bit of the MII PHY control register (MII address 0.12), and upon completion of the start-up delay ($t_{Start-up}$), the EPHY enters auto-negotiation. The mode of operation is automatically determined.
5 DIS100	Disable 100 BASE-TX PLL. This bit can be written anytime. Allows user to power down the clock generation PLL for 100BASE-TX clocks. 1 Disables 100BASE-TX PLL 0 100BASE-TX PLL state determined by EPHY operation mode
4 DIS10	Disable 10BASE-T PLL. This bit can be written anytime. Allows user to power down the clock generation PLL for 10BASE-T clocks. 1 Disables 10BASE-T PLL 0 10BASE-T PLL state determined by EPHY operation mode
3 LEDEN	LED Drive Enable. This bit can be written anytime. 1 Enables the EPHY to drive LED signals. 0 Disables the EPHY to drive LED signals.
2 EPHYWAI	EPHY Module Stops While in Wait. This bit can be written anytime. 1 Disables the EPHY module while the MCU is in wait mode. EPHY interrupts cannot be used to bring the MCU out of wait. 0 Allows the EPHY module to continue running during wait.
1	Reserved, should be cleared.
0 EPHYIEN	EPHY Interrupt Enable. This bit can be written anytime. 1 Enables EPHY module interrupts 0 Disables EPHY module interrupts

19.3.2.2 Ethernet Physical Transceiver Control Register 1 (EPHYCTL1)

IPSBAR
Offset: 0x1E_0001 (EPHYCTL1)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PHYADD4	PHYADD3	PHYADD2	PHYADD1	PHYADD0
W								
Reset	0	0	0	0	0	0	0	0

Figure 19-4. Ethernet Physical Transceiver Control Register 1 (EPHYCTL1)

Table 19-3. EPHYCTL1 Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4–0 PHYADD*n*	EPHY Address for MII Requests. These bits can be written anytime, but the EPHY address is latched to the MII PHY address register (MII address 21.4:0) only when the EPHYEN bit transitions from 0 to 1. PHYADD4 is the MSB of the of the EPHY address.

19.3.2.3 Ethernet Physical Transceiver Status Register (EPHYSR)

IPSBAR
Offset: 0x1E_0002 (EPHYSR)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	100DIS	10DIS	0	0	0	EPHYIF
W								
Reset	0	0	1	1	0	0	0	0

Figure 19-5. Ethernet Physical Transceiver Status Register (EPHYSR)

Table 19-4. EPHYSR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 100DIS	EPHY Port 100BASE-TX mode status. This bit is not writable — i.e., read-only. Output to indicate EPHY port Base100-TX mode status. 1 EPHY port 100BASE-TX disabled 0 EPHY port 100BASE-TX enabled
4 10DIS	EPHY Port 10BASE-T mode status. This bit is not writable. Output to indicate EPHY port 10BASE-T mode status. 1 EPHY port 10BASE-T disabled 0 EPHY port 10BASE-T enabled

Table 19-4. EPHYSR Field Descriptions (continued)

Field	Description
3–1	Reserved, should be cleared.
0 EPHYIF	EPHY Interrupt Flag. EPHYIF indicates that interrupt conditions have occurred. To clear the interrupt flag, write a 1 to this bit after reading the interrupt control register via the MII management interface. 1 EPHY interrupt has occurred 0 EPHY interrupt has not occurred

19.3.3 MII Registers

Table 19-5 gives an overview of all registers in the Ethernet physical interface that are accessible via the MII management interface. These registers are not part of the MCU memory map.

Table 19-5. MII Registers

MII Register Address		Use	Access
0	0x00	Control Register	Read/Write
1	0x01	Status Register	Read/Write[1]
2	0x02	PHY Identification Register 1	Read/Write[1]
3	0x03	PHY Identification Register 2	Read/Write[1]
4	0x04	Auto-Negotiation Advertisement Register	Read/Write
5	0x05	Auto-Negotiation Link Partner Ability Register	Read/Write[1]
6	0x06	Auto-Negotiation Expansion Register	Read/Write[1]
7	0x07	Auto-Negotiation Next Page Transmit	Read/Write
8	0x08	RESERVED	Read/Write[2]
9	0x09	RESERVED	Read/Write[2]
10	0x0A	RESERVED	Read/Write[2]
11	0x0B	RESERVED	Read/Write[2]
12	0x0C	RESERVED	Read/Write[2]
13	0x0D	RESERVED	Read/Write[2]
14	0x0E	RESERVED	Read/Write[2]
15	0x0F	RESERVED	Read/Write[2]
16	0x10	Interrupt Control Register	Read/Write
17	0x11	Proprietary Status Register	Read/Write[1]
18	0x12	Proprietary Control Register	Read/Write

1 Write has no effect.

2 Always reads 0x00.

NOTE

Bit notation for MII registers is: Bit 20.15 refers to MII register address 20 and bit number 15.

19.3.3.1 EPHY Control Register

MII Register Address: 0x00 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RESET	LOOP BACK	DATA RATE	ANE	PDWN	ISOL	RAN	DPLX	COL TEST	0	0	0	0	0	0	0
W																
Reset	0	0	1	X	0	0	0	1	0	0	0	0	0	0	0	0

Figure 19-6. Control Register

Table 19-6. Control Register Field Descriptions

Field	Description
15 RESET	EPHY Reset bit. Resetting a port is accomplished by setting this bit to 1. 1 The PHY resets the port's status and registers to the default values. The PHY also resets the PHY to its initial state. After the reset is complete, the PHY clears this bit automatically. The reset process is completed within 1.3 ms of this bit being set. While the preamble is suppressed, the management interface must not receive an ST within three MDC clock cycles following a software reset. 0 No effect
14 LOOPBACK	Digital Loopback Mode bit. Determines Digital Loopback Mode 1 Enables digital loopback mode. Port is placed in loopback mode. Loopback mode allows the TXD data to be sent to the RXD data circuitry within 512 bit times. The PHY is isolated from the medium (no transmit or receive to the medium allowed) and the MII_COL signal remains de-asserted, unless this bit is set. 0 Disables digital loopback mode
13 DATARATE	Speed Selection bit. The link speed is selected through the auto-negotiation process or by manual speed selection. ANE allows manual speed selection while it is set to 0. While auto-negotiation is enabled, DATARATE can be read or written but its value is not required to reflect speed of the link. 1 While auto-negotiation is disabled, selects 100 Mbps operation 0 While auto-negotiation is disabled, selects 10 Mbps operation
12 ANE	Auto-Negotiation Enable bit. This bit determines whether the A/N process is enabled. When auto-negotiation is disabled, DATARATE and DPLX determine the link configuration. While auto-negotiation is enabled, bits DATARATE and DPLX do not affect the link. 1 Enables auto-negotiation 0 Disables auto-negotiation
11 PDWN	Power Down bit. When this bit is set, the port is placed in a low power consumption mode. 1 Port is placed in a low power consumption mode. Normal operation is allowed within 0.5 s after PDWN and ISOL are changed to 0. During a transition to power-down mode (or if already in power down mode), the port responds only to management function requests through the MI interface. All other port operations are disabled. When power-down mode is exited, all register values are maintained. The port starts its operation based on the register values. 0 Normal operation
10 ISOL	Isolate bit. 1 Isolates the port's data path signals from the MII. The port does not respond to changes on MII_TXDx, MII_TXEN, and MII_TXER inputs, and it presents high impedance on MII_TXCLK, MII_RXCLK, MII_RXDV, MII_RXER, MII_RXDx, MII_COL, and MII_CRS outputs. The port responds to management transactions while in isolate mode. 0 Normal operation

Table 19-6. Control Register Field Descriptions (continued)

Field	Description
9 RAN	Restart Auto-Negotiation bit. This bit determines when the A/N process can start processing. 1 When auto-negotiation is enabled (ANE=1), the auto-negotiation process is restarted. After auto-negotiation indicates that it has been initialized, this bit is cleared. When bit ANE is cleared to indicate auto-negotiation is disabled, RAN must also be 0. 0 Normal operation
8 DPLX	Duplex Mode bit. This mode can be selected by the auto-negotiation process or manual duplex selection. Manual duplex selection is allowed only while the auto-negotiation process is disabled (ANE=0). While the auto-negotiation process is enabled (ANE = 1), the state of DPLX has no effect on the link configuration. While loopback mode is asserted (LOOPBACK =1), the value of DPLX has no effect on the PHY. 1 Indicates full-duplex mode 0 Indicates half-duplex mode
7 COLTEST	Collision Test bit. The collision test function is enabled only if the loopback mode of operation is also selected (LOOPBACK = 1). 1 Forces the PHY to assert the MII_COL signal within 512 bit times from the assertion of MII_TXEN and de-assert MII_COL within 4 bit times of MII_TXEN being de-asserted. 0 Normal operation
6–0	Reserved, should be cleared.

19.3.3.2 Status Register

This register advertises the capabilities of the port to the MII.

MII Register Address: 0x01 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	100 T4	100X FD	100X HD	10T FD	10THD	0	0	0	0	SUP PRE	AN COMP	REM FLT	AN ABL	LNK STST	JAB DT	EX CAP
W																
Reset	0	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1

Figure 19-7. Status Register

Table 19-7. Status Register Field Descriptions

Field	Description
15 100T4	100BASE-T4 1 Indicates PHY supports 100BASE-T4 transmission 0 Indicates the PHY does not support 100BASE-T4 transmission **Note:** This function is not implemented in the EPHY module.
14 100XFD	100BASE-TX Full-Duplex 1 Indicates PHY supports 100BASE-TX full-duplex mode 0 Indicates PHY does not support 100BASE-TX full-duplex mode
13 100XHD	100BASE-TX Half-Duplex 1 Indicates the PHY supports 100BASE-TX half-duplex mode 0 Indicates the PHY does not support 100BASE-TX half-duplex mode
12 10TFD	10BASE-T Full-Duplex 1 Indicates the PHY supports 10BASE-T full-duplex mode 0 Indicates the PHY does not support 10BASE-T full-duplex mode
11 10THD	10BASE-T Half-Duplex 1 Indicates the PHY supports 10BASE-T half-duplex mode 0 Indicates the PHY does not support 10BASE-T half-duplex mode
10–7	Reserved, should be cleared.
6 SUPPRE	MF Preamble Suppression 1 Indicates that management frames are not required to contain the preamble stream 0 Indicates that management frames are required to contain the preamble stream
5 ANCOMP	Auto-Negotiation Complete bit. To inform the management interface (MI) that it has completed processing, ANCOMP is set by the A/N process. After it has been started, the auto-negotiation process uses link code words to exchange capability information and establish the highest common denominator (HCD) for link transactions. 1 Indicates that the auto-negotiation process has completed and that the contents of registers 4 through 7 are valid. 0 Indicates that the auto-negotiation process has not completed and that the contents of registers 4 through 7 are not valid

Table 19-7. Status Register Field Descriptions (continued)

Field	Description
4 REMFLT	Remote Fault bit. Possible remote faults (RF) • The link partner transmits the RF bit (5.13=1) • Link partner protocol is not 00001 (5.4:0) • Link partner advertises only T4 capability (5.9:5) • No common operation mode found between PHY and the link partner. After it is set, REMFLT is cleared each time register 1 is read via the management interface. REMFLT is also cleared by a PHY reset. 1 Indicates that a remote fault condition has been detected. 0 No fault detected
3 ANABL	Auto-Negotiation Ability 1 Indicates that PHY has auto-negotiation ability 0 Indicates that PHY does not have auto-negotiation ability
2 LNKSTST	Link Status The PHY sets this bit when it determines that a valid link has been established. The occurrence of a link failure causes LNKSTST to be cleared. After it has been cleared, it remains cleared until it is read via the management interface. 1 Indicates a valid link has been established 0 Indicates a valid link has NOT been established
1 JABDT	Jabber Detect After it is set, JABDT is cleared each time register 1 is read via the management interface. JABDT is also cleared by a PHY reset. For 100BASE-TX operation, this signal is always cleared. 1 Indicates that a jabber condition has been detected 0 Indicates that no jabber condition has been detected
0 EXCAP	Extended Capability 1 Indicates that the extended register set (registers 2–31) has been implemented in the PHY. 0 Indicates that the extended register set (registers 2–31) has NOT been implemented in the PHY

19.3.3.3 EPHY Identifier Register 1

The EPHY Identifier Registers 1 and 2 provide the PHY identification code.

MII Register Address: 0x02 Access: User read/write

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

Figure 19-8. EPHY Identifier Register 1

Table 19-8. EPHY Identifier Register 1 Field Descriptions

Field	Description
15–0 PHYID	PHY ID Number. Composed of bits 3:18 of the organization unique identifier (OUI).

19.3.3.4 EPHY Identifier Register 2

The EPHY Identifier Registers 1 and 2 provide the PHY identification code.

MII Register Address: 0x03 Access: User read/write

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

Figure 19-9. EPHY Identifier Register 2

Table 19-9. EPHY Identifier Register 2 Field Descriptions

Field	Description
15–10 PHYID	PHY ID number organization unique identifier. Composed of bits 19:24.
9–4 MODELNUMBER	Manufacturers model number. Composed of bits 9:4.
3–0 REVISIONNUMBER	Manufacturers revision number. Composed of bits 3:0.

19.3.3.5 Auto-Negotiate (A/N) Advertisement Register

The auto-negotiation (A/N) process requires four registers to communicate link information with its link partner: A/N advertisement register (MII register 4), A/N link partner ability register (MII register 5), A/N expansion register (MII register 6), and the A/N next page transmit register (MII register 7).

Figure 19-10 shows the contents of the A/N advertisement register. On power-up, before A/N starts, the register sets the selector field, bits 4.4:0, to 00001 to indicate that it is IEEE Standard 802.3 compliant. The

technology ability fields (4.9:5) are set according to the values in the MII status register (1.15:11). The MI can set the technology ability field bits before renegotiations to allow management to auto-negotiate to an alternate common mode.

MII Register Address: 0x04 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NXTP	0	RFLT	0	0	FLCTL	0	TAF 100FD	TAF 100HD	TAF 10FD	TAF 10HD	SELECTORFIELD				
W																
Reset	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1

Figure 19-10. Auto Negotiate Advertisement Register

Table 19-10. Auto Negotiate Advertisement Register Field Descriptions

Field	Description
15 NXTP	Next Page 1 Capable of sending next pages 0 Not capable of sending next pages
14	Reserved, should be cleared.
13 RFLT	Remote Fault 1 Remote fault 0 No remote fault
12	Reserved, should be cleared.
11	Reserved, should be cleared.
10 FLCTL	Flow Control 1 Advertise implementation of the optional MAC control sublayer and pause function as specified in IEEE standard clause 31 and anex 31B of 802.3. Setting FLCTL has no effect except to set the corresponding bit in the FLP stream 0 No MAC-based flow control
9	Reserved, should be cleared.
8 TAF100FD	100BASE-TX Full-Duplex 1 100BASE-TX full -duplex capable 0 Not 100BASE-TX full-duplex capable
7 TAF100HD	100BASE-TX Half-Duplex 1 100BASE-TX half-duplex capable 0 Not 100BASE-TX half-duplex capable
6 TAF10FD	10BASE-T Full-Duplex 1 10BASE-T full-duplex capable 0 Not 10BASE-T full-duplex capable
5 TAF10HD	10BASE-T Half-Duplex 1 10BASE-T half-duplex capable 0 Not 10BASE-T half-duplex capable
4–0 SELECTORFIELD	Selector field. This field is set to 0b1 at power-up to indicate that the module complies with the IEEE 802.3 standard.

19.3.3.6 Auto Negotiation Link Partner Ability (Base Page)

Figure 19-11 shows the contents of the A/N link partner ability register. The register can only be read by the MI and is written by the auto-negotiation process when it receives a link code word advertising the capabilities of the link partner. This register has a dual purpose: exchange of base page information as shown in Figure 19-11, and exchange of next page information as shown in Figure 19-12.

MII Register Address: 0x05 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NXTP	ACK0	RFLT	0	0	FLCTL	TAF 100T4	TAF 100FD	TAF 100HD	TAF 10FD	TAF 10HD	SELECTORFIELD				
W																
Reset	X	X	X	0	0	X	X	X	X	X	X	X	X	X	X	X

Figure 19-11. Auto Negotiation Link Partner Ability Register (Base Page)

Table 19-11. Auto Negotiation Link Partner Ability Register (Base Page) Field Descriptions

Field	Description
15 NXTP	Next Page 1 Link partner capable of sending next pages 0 Link partner not capable of sending next pages
14 ACK0	Acknowledge 1 Link Partner has received link code word 0 Link Partner has not received link code word
13 RFLT	Remote Fault 1 Remote fault 0 No remote fault
12–11	Reserved, should be cleared.
10 FLCTL	Flow Control 1 Advertise implementation of the optional MAC control sublayer and pause function as specified in IEEE standard clause 31 and anex 31B of 802.3. Setting FLCTL has no effect except to set the corresponding bit in the FLP stream 0 No MAC-based flow control
9 TAF100T4	100BASE-T4 Full-Duplex 1 Link partner is 100BASE-T4 capable 0 Link partner is not 100BASE-T4 capable **Note:** This function is not implemented in the EPHY.
8 TAF100FD	100BASE-TX Full-Duplex 1 Link partner is 100BASE-TX full-duplex capable 0 Link partner is not 100BASE-TX full-duplex capable
7 TAF100HD	100BASE-TX Half-Duplex 1 Link partner is 100BASE-TX half-duplex capable 0 Link partner is not 100BASE-TX half-duplex capable
6 TAF10FD	10BASE-T Full-Duplex 1 Link partner is10BASE-T full-duplex capable 0 Link partner is not 10BASE-T full-duplex capable

Table 19-11. Auto Negotiation Link Partner Ability Register (Base Page) Field Descriptions (continued)

Field	Description
5 TAF10HD	10BASE-T Half-Duplex 1 Link partner is 10BASE-T half-duplex capable 0 Link partner is not 10BASE-T half-duplex capable
4–0 SELECTORFIELD	Selector field. The valid values are given below. 00001IEEE 802.3 standard 00010IEEE 802.9 ISLAN-16T standard

19.3.3.7 Auto Negotiation Link Partner Ability (Next Page)

MII Register Address: 0x05 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NXTP	ACK	MSGP	ACK2	TGL	CODEFIELD										
W																
Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 19-12. Auto Negotiation Link Partner Ability Register (Next Page)

Table 19-12. Auto Negotiation Link Partner Ability Register (Next Page) Field Descriptions

Fields	Description
15 NXTP	Next Page 1 Additional next pages follow 0 Last page transmitted
14 ACK	Acknowledge ACK is used to acknowledge receipt of information. 1 Link partner has received link code word 0 Link partner has not received link code word
13 MSGP	Message Page 1 Message page 0 Unformatted page
12 ACK2	Acknowledge 2 ACK2 is used to indicate that the receiver is able to act on the information (or perform the task) defined in the message. 1 Receiver is able to perform the task defined in the message 0 Receiver is unable to perform the task defined in the message
11 TGL	Toggle 1 Previous value of the transmitted link code word equalled 0 0 Previous value of the transmitted link code word equalled 1
10–0 CODEFIELD	Message/Unformatted Code Field Message code field — Predefined code fields defined in IEEE 802.3u-1995 Annex 28C Unformatted code filed — 11-bit field containing an arbitrary value

19.3.3.8 Auto-Negotiation Expansion Register

Figure 19-13 shows the contents of the A/N expansion register. The MI process can only read this register. This register contains information about the A/N capabilities of the port's link partner and information on the status of the parallel detection mechanism.

MII Register Address: 0x06 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	PDFLT	LPNPA	NXTPA	PRCVD	LPANA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 19-13. Auto-Negotiation Expansion Register

Table 19-13. Auto-Negotiation Expansion Register Field Descriptions

Field	Description
15–5	Reserved, should be cleared.
4 PDFLT	Parallel Detection Fault This bit is used to indicate that zero or more than one of the NLP receive link integrity test function for 100BASE-TX have indicated that the link is ready (link_status=READY) when the A/N wait timer has expired. PDFLT is reset to 0 after a read of register 6. 1 Parallel detection fault has occurred 0 Parallel detection fault has not occurred
3 LPNPA	Link Partner Next Page Able Bit to indicate whether the link partner has the capability of using NP. 1 Link partner is next page able 0 Link partner is not next page able
2 NXTPA	Next Page Able This bit is used to inform the MI and the link partner whether the port has next page capabilities. 1 The port has next page capabilities 0 The port does not have next page capabilities
1 PRCVD	Page Received Bit is used to indicate whether a new link code word has been received and stored in the A/N link partner ability register (MII register 5). PRCVD is reset to 0 after register 6 is read. 1 Three identical and consecutive link code words have been received from link partner 0 Three identical and consecutive link code words have not been received from link partner
0 LPANA	Link Partner A/N Able Indicates whether the link partner has A/N capabilities. 1 Link partner is A/N able 0 Link partner is not A/N able

19.3.3.9 Auto Negotiation Next Page Transmit

Figure 19-14 shows the contents of the A/N next page transmit register. The MI writes to this register if it needs to exchange more information with the link partner. The PHY defaults to sending only a NULL message page to the link partner unless the STA overrides the values in the register. Next pages are transmitted until the link partner has no more pages to transmit and bit 7.15 has been cleared by the STA.

MII Register Address: 0x07 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	NXTP	0	MSGP	ACK2	TGL	CODEFIELD										
W																
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 19-14. Auto Negotiation Next Page Transmit Register

Table 19-14. Auto Negotiation Next Page Transmit Register Field Descriptions

Field	Description
15 NXTP	Next Page 1 Additional next pages follow 0 Last page to transmit
14	Reserved, should be cleared.
13 MSGP	Message Page 1 Message page 0 Unformatted page
12 ACK2	Acknowledge 2 ACK2 is used to indicate that the receiver is able to act on the information (or perform the task) defined in the message. 1 Receiver is able to perform the task defined in the message 0 Receiver is unable to perform the task defined in the message
11 TGL	Toggle 1 Previous value of the transmitted link code word equalled 0 0 Previous value of the transmitted link code word equalled 1
10–0 CODEFIELD	Message/Unformatted Code Field Message code field — Predefined code fields defined in IEEE 802.3u-1995 Annex 28C Unformatted code field — Eleven bit field containing an arbitrary value

19.3.4 PHY-Specific Registers

PHY also contains a number of registers to set its internal mode of operation. These registers can be set through the external management interface to determine capabilities such as speed, test-mode, circuit bypass mode, interrupt setting, etc. The PHY register set includes registers 16 through 29. These registers are not part of the MCU memory map.

19.3.4.1 Interrupt Control Register

MII Register Address: 0x10 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	ACKIE	PRIE	LCIE	ANIE	PDFIE	RFIE	JABIE	0	ACKR	PGR	LKC	ANC	PDF	RMTF	JABI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-15. Interrupt Control Register

Table 19-15. Interrupt Control Register Field Descriptions

Field	Description
15	Reserved, should be cleared.
14 ACKIE	Acknowledge Bit Received Interrupt Enable 1 Enable interrupt when the acknowledge bit is received from the link partner 0 Disable interrupt when acknowledge bit is received
13 PRIE	Page Received INT Enable 1 Enable interrupt when a new page is received 0 Disable interrupt when a page is received
12 LCIE	Link Changed Enable 1 Enable interrupt when the link status changes 0 Disable interrupt when the link status changes
11 ANIE	Auto-Negotiation Changed Enable 1 Enable interrupt when the state of the auto-negotiation state machine has changed since the last access of this register 0 Disable interrupt when the state of the auto-negotiation state machine has changed since the last access of this register
10 PDFIE	Parallel Detect Fault Enable 1 Enable interrupt on a parallel detect fault 0 Disable interrupt on a parallel detect fault
9 RFIE	Remote Fault Interrupt Enable 1 Enable interrupt on a parallel detect fault 0 Disable interrupt on a parallel detect fault
8 JABIE	Jabber Interrupt Enable 1 Enable setting interrupt on detection of a jabber condition 0 Disable setting interrupt on detection of a jabber condition
7	Reserved, should be cleared.
6 ACKR	Acknowledge Bit Received 1 Acknowledge bit has been received from the link partner 0 Acknowledge bit has not been received since the last access of this register. (ACK bit 14 of the auto-negotiation link partner ability register was set by receipt of link code word)
5 PGR	Page Received 1 A new page has been received from the link partner 0 A new page has not been received from the link partner since the last access of this register (Bit 1 was set by a page received event)

Table 19-15. Interrupt Control Register Field Descriptions (continued)

Field	Description
4 LKC	Link Changed 1 The link status has changed since the last access of this register 0 The link status has not changed since the last access of this register. (LNK bit 14 of the proprietary status register was changed)
3 ANC	Auto-Negotiation Changed 1 The auto-negotiation status has changed since the last access of this register 0 The auto-negotiation status has not changed since the last access of this register
2 PDF	Parallel Detect Fault 1 A parallel-detect fault has occurred since the last access of this register 0 A parallel-detect fault has not been detected since the last access of this register. (Bit 4 was set by rising edge of parallel detection fault)
1 RMTF	Remote Fault 1 A remote fault condition has been detected since the last access of this register 0 A remote fault condition has not been detected since the last access of this register. (RMTF bit 4 of the status register was set by rising edge of a remote fault)
0 JABI	Jabber Interrupt 1 A jabber condition has been detected since the last access of this register 0 A jabber condition has not been detected since the last access of this register (JABD bit 1 of the status register was set by rising edge of jabber condition)

19.3.4.2 Proprietary Status Register

MII Register Address: 0x11 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	LNK	DPMD	SPD	0	ANNC	PRCVD	ANC MODE	0	0	PLR	0	0	0	0	0
W																
Reset	0	1	1	1	0	0	0	1[1]	0	0	0	0	0	0	0	0

[1] This bit is valid only when ANNC is set.

Figure 19-16. Proprietary Status Register

Table 19-16. Proprietary Status Register Field Descriptions

Field	Description
15	Reserved, should be cleared.
14 LNK	Link Status This is a duplicate of LNKSTAT bit 2 of the status register (1.2). 1 Link is down 0 Link is up
13 DPMD	Duplex Mode 1 Full-duplex 0 Half-duplex

Table 19-16. Proprietary Status Register Field Descriptions (continued)

Field	Description
12 SPD	Speed 1 100 Mbps 0 10 Mbps
11	Reserved, should be cleared.
10 ANNC	Auto-Negotiation Complete This is a duplicate of ANCOMP bit 5 of the status register (1.5) 1 A-N complete 0 A-N not complete
9 PRCVD	Page Received 1 Three identical and consecutive link code words have been received 0 Three identical and consecutive link code words have not been received
8 ANCMODE	Auto-Negotiation (A-N) Common Operating Mode This bit is only valid when ANNC is set. 1 A common operation mode was not found 0 A-N is complete and a common operation mode has been found
7	Reserved, should be cleared.
6	Reserved, should be cleared.
5 PLR	Polarity Reversed (10BASE-T) 1 10BASE-T receive polarity is reversed 0 10BASE-T receive polarity is normal
4–0	Reserved, should be cleared.\

19.3.4.3 Proprietary Control Register

MII Register Address: 0x12 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	FE FLTD	MIILBD	0	1	JBDE	LNK TSTD	POL CORD	ALGD	ENC BYP	SCR BYP	TRD ANALB	TR TST	0	0	0
W																
Reset	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0

Figure 19-17. Proprietary Control Register

The miscellaneous (EMISC) register provides visibility of internal counters used by the EMAC.

Table 19-17. Proprietary Control Register Field Descriptions

Field	Description
15	Reserved, should be cleared.
14 FEFLTD	Far End Fault Disable 1 Far end fault detect is disabled 0 Far end fault detect on receive and transmit is enabled. This applies only while auto-negotiation is disabled

Table 19-17. Proprietary Control Register Field Descriptions (continued)

Field	Description
13 MIILBD	MII Loopback Disable 1 Disable MII loopback 0 MII transmit data is looped back to the MII receive pins
12	Reserved, should be cleared.
11	Reserved, should be set.
10 JBDE	Jabber Detect Enable (10BASE-T) 1 Enable jabber detection 0 Disable jabber detection
9 LNKTSTD	Link Test Disable (10BASE-T) 1 Disable 10BASE-T link integrity test 0 10BASE-T link integrity test enabled
8 POLCORD	Disable Polarity Correction (10BASE-T) 1 10BASE-T receive polarity correction is disabled 0 10BASE-T receive polarity is automatically corrected
7 ALGD	Disable Alignment 1 Un-aligned mode. Available only in symbol mode 0 Aligned mode
6 ENCBYP	Encoder Bypass 1 Symbol mode and bypass 4B/5B encoder and decoder 0 Normal mode
5 SCRBYP	Scrambler Bypass Mode (100BASE-TX) 1 Bypass the scrambler and de-scrambler 0 Normal
4 TRDANALB	Transmit and Receive Disconnect and Analog Loopback 1 High-impedance twisted pair transmitter. Analog loopback mode overrides and forces this bit 0 Normal operation
3 TRTST	Transmit and Receive Test (100BASE-TX) 1 Transmit and receive data regardless of link status 0 Normal operation
2–0	Reserved, should be cleared.

19.4 Functional Description

The Ethernet physical interface is an IEEE 802.3 compliant 10/100 Ethernet physical transceiver. The Ethernet physical interface can be configured to support 10BASE-T or 100BASE-TX applications. The Ethernet physical interface is configurable via internal registers which are accessible through the MII management interface as well as limited configurability using the EPHY register map.

There are five basic modes of operation for the EPHY:

- Power down/initialization
- Auto-negotiate
- 10BASE-T
- 100BASE-TX
- Low-power

19.4.1 Power Down/Initialization

Upon reset, the EPHYEN bit, in the Ethernet physical transceiver control register 0 (EPHYCTL0), is cleared and EPHY is in its lowest power consumption state. All analog circuits are powered down. The twisted-pair transmitter and receiver pins (PHY_TXP, PHY_TXN, PHY_RXP, and PHY_RXN) are high-impedance. The MII management interface is not accessible. All MII registers are initialized to their reset state. The ANDIS, DIS100, and DIS10 bits, in the EPHYCTL0 register, have no effect until the EPHYEN bit is set.

The EPHYEN bit can be set or cleared by a register write at any time. Prior to enabling the EPHY, setting EPHYEN to 1, the MII PHY address PHYADD[4:0] must be set in the Ethernet physical transceiver control register 1 (EPHYCTL1), and the ANDIS, DIS100, DIS10 bits, in the EPHYCTL0 register, must be configured for the desired start-up operation. When the EPHYEN bit transitions from 0 to 1, MDIO communications must be delayed until completion of a start-up delay period ($t_{Start-up}$, see Figure 19-19).

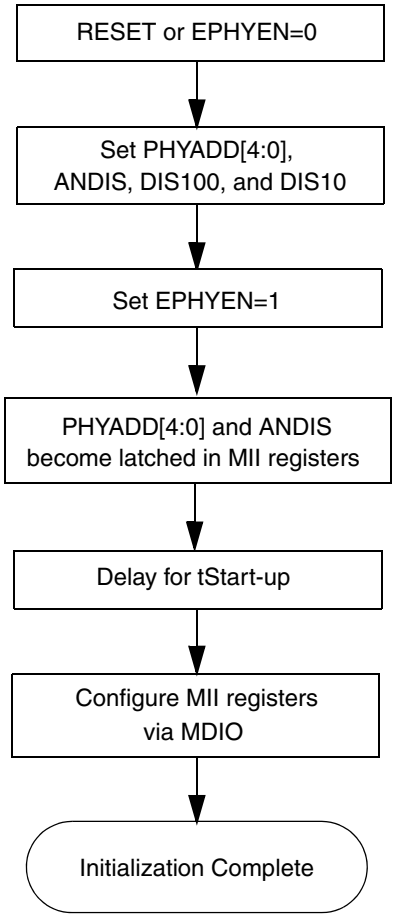


Figure 19-18. EPHY Start-Up / Initialization Sequence

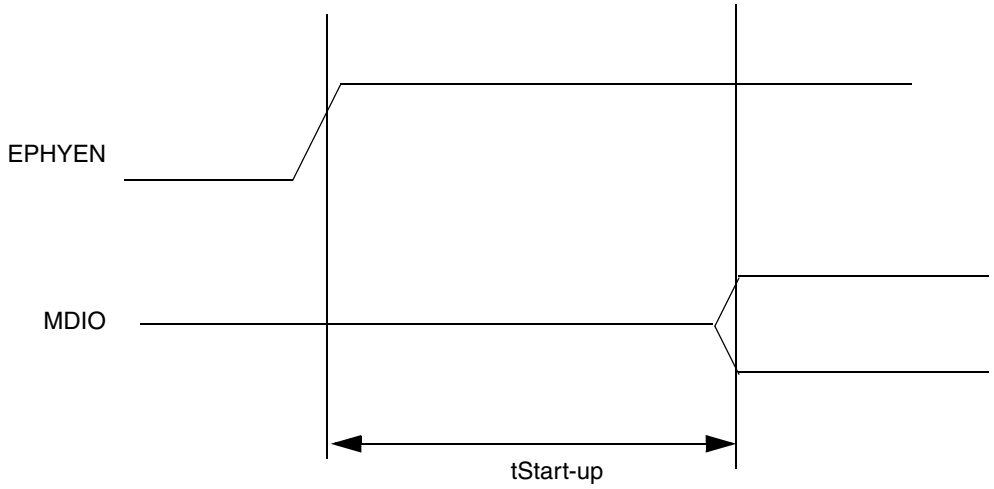


Figure 19-19. EPHY Start-Up Delay

If the auto-negotiation mode of operation is desired, the ANDIS bit in the EPHYCTL0 must be set to 0 and the DIS100 and DIS10 bits must be cleared prior to setting EPHYEN to 1. Refer to Section 19.4.2, "Auto-Negotiation," for more information on auto-negotiation operation.

If the mode of operation is set manually, the ANDIS bit must be set to 1 in the EPHYCTL0 register and the DIS100 and DIS10 bits must be cleared prior to setting EPHYEN to 1. After the EPHYEN bit has been set and the start-up delay period is completed, the mode of operation can be configured through the MII registers. Table 19-18 summarizes the MII register configuration and operational modes.

Table 19-18. Operational Configuration While Auto-Negotiation is Disabled[1]

Bit 0.12 Auto Neg.	Bit 0.13 Data Rate	Bit 0.8 Duplex	Bit 18.6 Encoder Bypass	Bit 18.5 Scrambler Bypass	Bit 18.7 Symbol Unalign	Operation
0	0	1	X	X	X	10BASE-T full-duplex
0	0	0	X	X	X	10BASE-T half-duplex
0	1	1	0	0	0	100BASE-TX full-duplex
0	1	1	1	0	0	100BASE-TX full-duplex with encoder bypass (symbol mode) — aligned
0	1	1	1	0	1	100BASE-TX full-duplex with encoder bypass (symbol mode) — unaligned
0	1	1	1	1	0	100BASE-TX full-duplex with scrambler and encoder bypassed (symbol mode), aligned
0	1	1	1	1	1	100BASE-TX full-duplex with scrambler and encoder bypassed (symbol mode), unaligned
0	1	0	0	0	0	100BASE-TX half-duplex

[1] Symbol mode is not supported.

19.4.2 Auto-Negotiation

Auto-negotiation is used to determine the capabilities of the link partner. Auto-negotiation is compliant with IEEE 802.3 clause 28. In this case, the PHY transmits fast link pulse (FLP) bursts to share its capabilities with the link partner.

If the link partner is also capable of performing auto-negotiation, it also sends FLP bursts. The information shared through the FLP bursts allows both link partners to find the highest common mode (if it exists).

If no common mode is found, the remote fault bit (1.4) is set. A remote fault is defined as a condition in which the PHY and the link partner cannot establish a common operating mode. Configuring auto-negotiation advertisement register sets the different auto-negotiation advertisement modes.

If the link partner does not support auto-negotiation, it transmits normal link pulses (NLP) for 10 Mbps operation or 100 Mbps idle symbols. Based on the received signal, the PHY determines whether the link partner is 10 Mbps capable or 100 Mbps capable. The ability to do this is called parallel detection. If using parallel detection, the link is configured as a half-duplex link. After parallel detection has established the link configuration, the remote fault bit is set if the operating mode does not match the pre-set operating modes.

Figure 19-20 shows the main blocks used in the auto-negotiation function. The transmit block allows transmission of fast link pulses to establish communications with partners that are auto-negotiation able. The receive block determines the capabilities of the link partner and writes to the link partner ability register (register 5). The arbitration block determines the highest common mode of operation to establish the link.

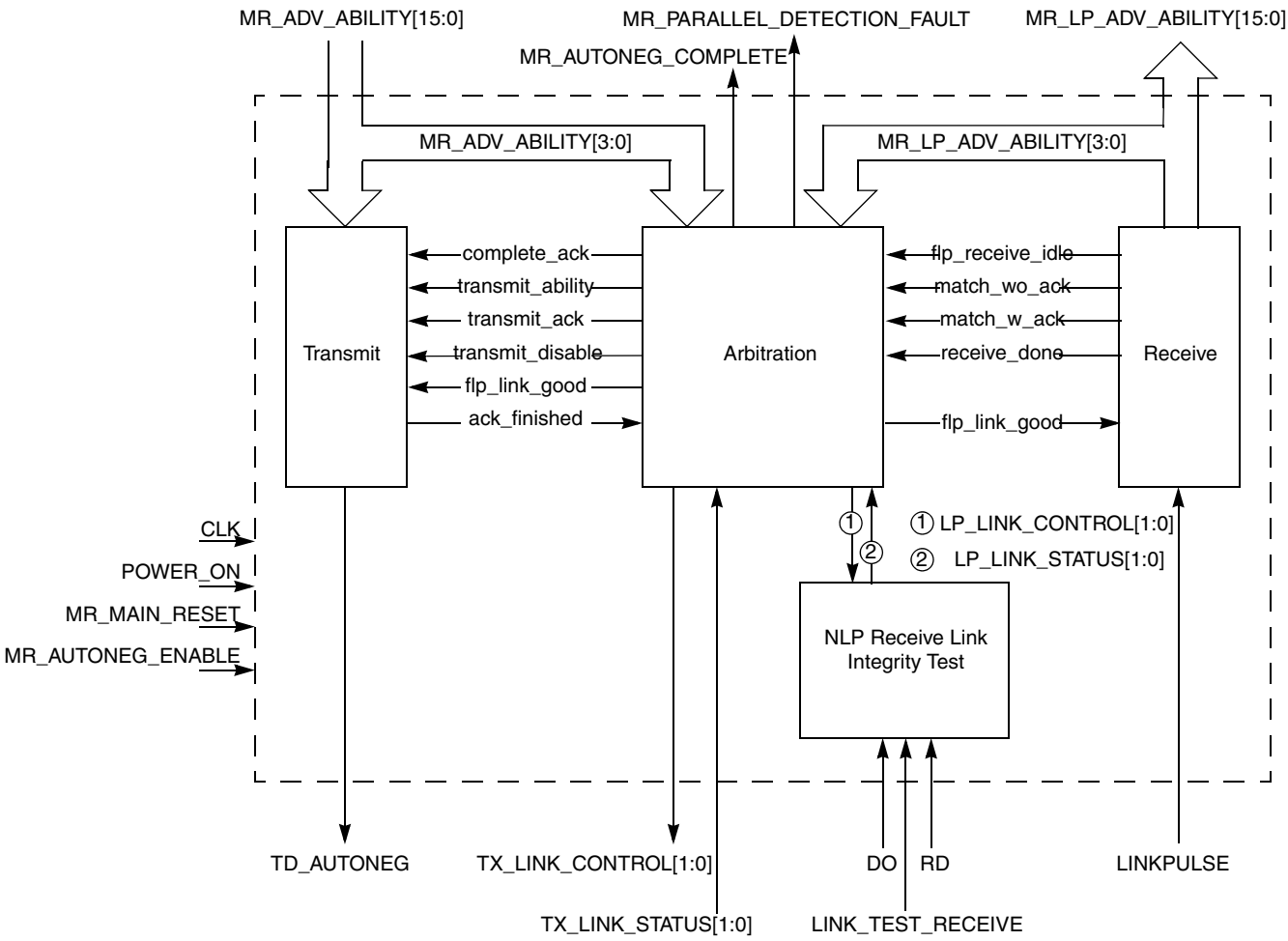


Figure 19-20. Auto-Negotiation

19.4.3 10BASE-T

The 10BASE-T interface implements the physical layer specification for a 10 Mbps over two pairs of twisted-pair cables. The specifications are given in clause 14 of the IEEE 802.3 standard.

In 10BASE-T mode, Manchester encoding is used. When transmitting, nibbles from the MII are converted to a serial bit stream and then Manchester encoded. When receiving, the Manchester encoded bit stream is decoded and converted to nibbles for presentation to the MII.

A 2.5 MHz internal clock is used for nibble wide transactions. A 10 MHz internal clock is used for serial transactions.

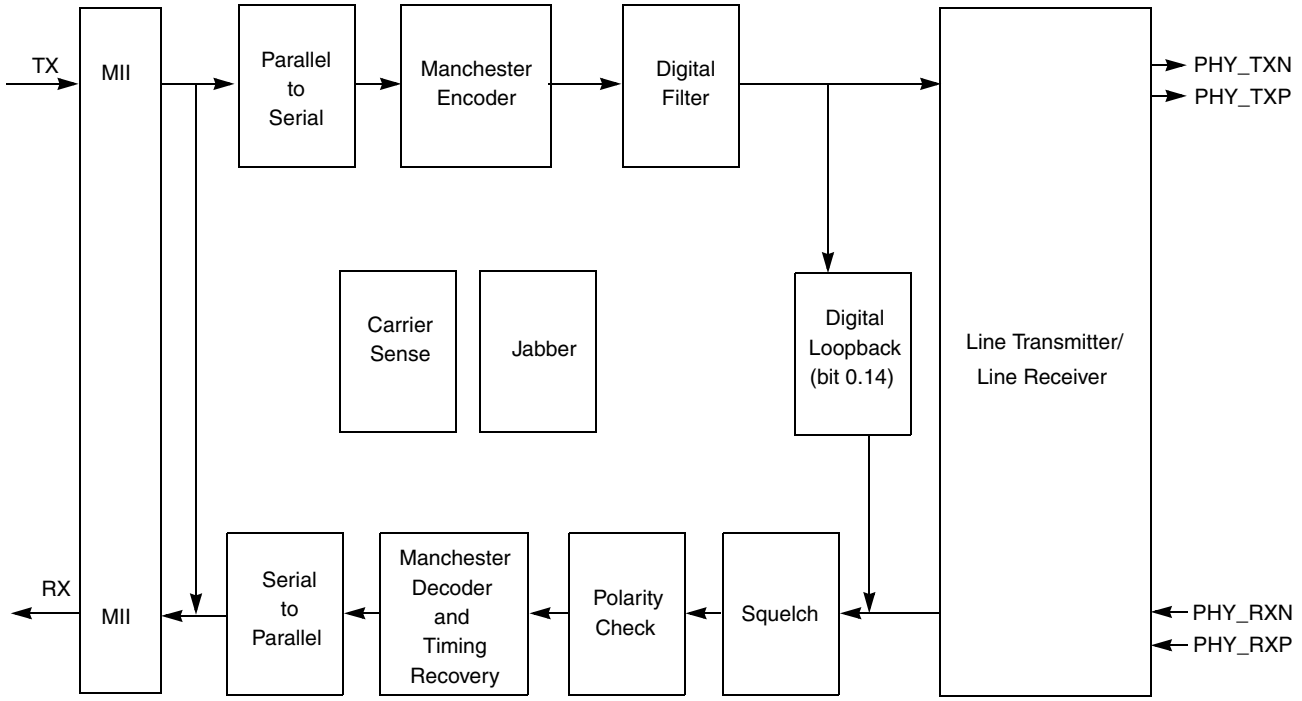


Figure 19-21. 10BASE-T Block Diagram

Parallel to Serial: Converts the 4-bit wide nibbles from the MII to serial format before the information is processed by subsequent blocks.

Manchester Encoder: Allows encoding of the clock and data in one bit stream. A logical one is encoded as a zero when the clock is high and a one when the clock is low. A logical zero is encoded as a one when the clock is high and a zero when the clock is low.

Digital Filter: Performs pre-emphasis and low pass filtering of the input Manchester data.

DAC: Converts the digital data to an analog format before transmission on the media.

Carrier Sense: In half-duplex operation, carrier is asserted when the transmit or receive medium is active. In full-duplex operation, carrier asserted only on reception of data. During receive, carrier sense is asserted during reception of a valid preamble, and de-asserted after reception of an EOF.

Loopback: Enabled when bit 0.14 is asserted. This loopback mode allows for the Manchester encoded and filtered data to be looped back to the squelch block in the receive path. All the 10BASE-T digital functions are exercised during this mode. The transmit and receive channels are disconnected from the media.

MII loopback (18.13) must be disabled to allow for correct operation of the digital loopback (0.14).

Link Generator: Generates a 100 ns duration pulse at the end of every 12 ms period of the transmission path being idle (TXEN de-asserted). This pulse is used to keep the 10BASE-T link operational in the absence of data transmission.

Link Integrity Test: Used to determine whether the 10BASE-T link is operational. If neither data nor a link pulse is received for 64 ms, then the link is considered down. While the link is down, the transmit,

loopback, collision detect, and SQE functions are disabled. The link down state is exited after receiving data or four link pulses.

Jabber: Prevents the transmitter from erroneously transmitting for too long a period. The maximum time the device can transmit is 50,000 bit times. When the jabber timer is exceeded, the transmit output goes idle for 0.525 s.

This function can be disabled with the jabber inhibit register bit (18.10).

Squelch: Used to determine whether active data, a link pulse, or an idle condition exists on the 10BASE-T receive channel. While an idle or link pulse condition exists, a higher squelch level is used for greater noise immunity. The squelch output is used to determine when the Manchester decoder should operate. The output is also used to determine when an end of packet is received.

Polarity Check: By examining the polarity of the received link pulses, EPHY can determine whether the received signal is inverted. If the pulses are inverted, this function changes the polarity of the signal.This feature is activated if eight inverted link pulses are received or four frames with inverted EOF are encountered.

Manchester Decoder and Timing Recovery: Decodes the Manchester encoded data. The receive data and clock are recovered during this process.

Serial to Parallel: Converts the serial bit stream from the Manchester decoder to the required MII parallel format.

PMD Sublayer: Transmits and receives signals compliant with IEEE 802.3, Section 14.

Line Transmitter and Line Receiver: These analog blocks allow the EPHY to drive and receive data from the 10BASE-T media.

19.4.4 100BASE-TX

100BASE-TX specifies operation over two pairs of category 5 unshielded twisted-pair cable (UTP).

The EPHY implementation includes the physical coding sublayer (PCS), the physical medium attachment (PMA), and the physical medium dependent (PMD) sublayer.

The block diagram for 100BASE-TX operation is shown in Figure 19-22.

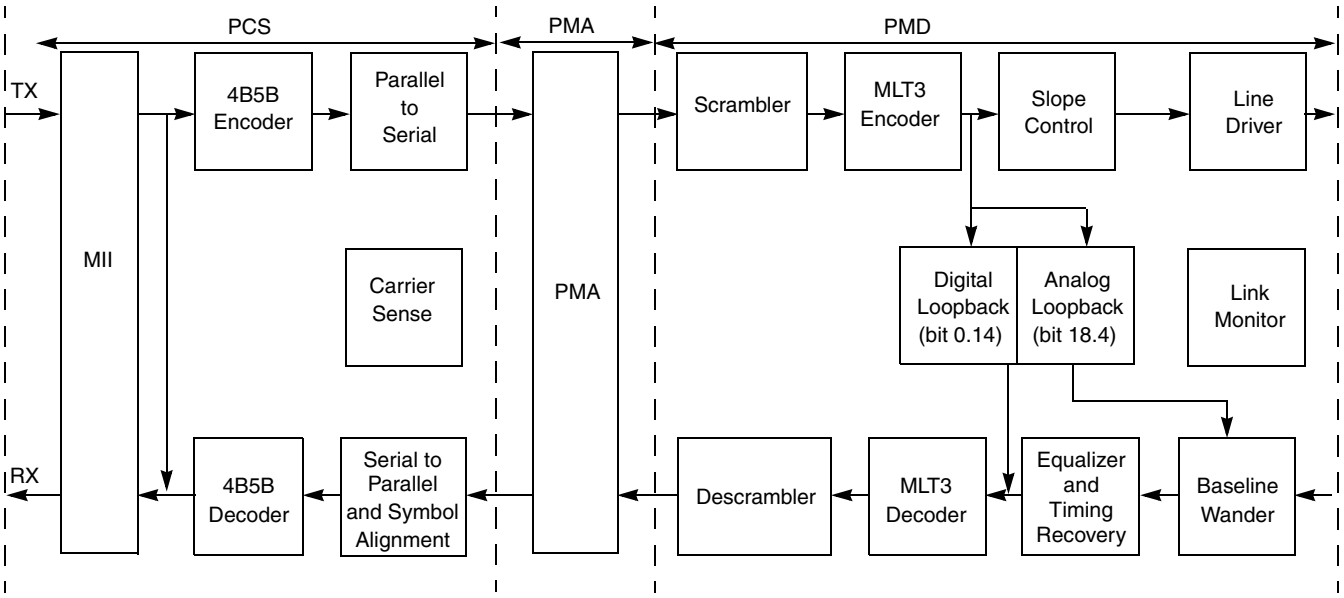


Figure 19-22. 100BASE-TX Block Diagram

19.4.4.1 Sublayers

19.4.4.1.1 PCS Sublayer

The PCS sublayer is the MII interface that provides a uniform interface to the reconciliation sublayer.

The services provided by the PCS include:

- Encoding/decoding of MII data nibbles to/from 5-bit code-groups (4B/5B)
- Carrier sense and collision indications
- Serialization/deserialization of code-groups for transmission/reception on the PMA
- Mapping of transmit, receive, carrier sense, and collision detection between the MII and the underlying PMA

Serial to Parallel and Symbol Alignment: This block looks for the occurrence of the JK symbol to align the serial bit stream and convert it to a parallel format.

Carrier Sense: In full-duplex mode, carrier sense is only asserted while the receive channel is active**.** The carrier sense examines the received data bit stream looking for the SSD, the JK symbol pair. In the idle state, IDLE symbols (all logic ones) are received. If the first 5-bit symbols received after an idle stream forms the J symbol (11000) it asserts the CRS signal. At this point the second symbol is checked to confirm the K symbol (10001). If successful, the following aligned data (symbols) are presented to the 4B/5B decoder. If the JK pair is not confirmed, the false carrier detect is asserted and the idle state is re-entered.

Carrier sense is de-asserted when the ESD (end-of-stream) delimiter, the TR symbol pair, is found, or when an idle state is detected.

In half-duplex, CRS is also asserted on transmit.

Parallel to Serial: This block takes parallel data and converts it to serial format.

4B/5B Encoder/Decoder: The 4B/5B encoder converts the 4-bit nibbles from the reconciliation sublayer to a 5-bit code group.

19.4.4.1.2 PMA Sublayer

The PMA provides medium-independent means for the PCS and other bit-oriented clients (e.g., repeaters) to support the use of a range of physical media. For 100BASE-TX the PMA performs these functions:

- Mapping of transmit and receive code-bits between the PMA's client and the underlying PMD
- Generating a control signal indicating the availability of the PMD to a PCS or other client
- Synchronization with the auto-negotiation function
- Generating indications of carrier activity and carrier errors from the PMD
- Recovery of clock from the NRZI data supplied by the PMD

19.4.4.1.3 PMD Sublayer

For 100BASE-TX, the ANSI X3.263: 199X (TP-PMD) standard is used. These signalling standards, called PMD sublayers, define 125 Mbps, full-duplex signalling systems that use STP and UTP wiring.

Scrambler/De-scrambler: The scrambler and de-scrambler used in EPHY meet the ANSI Standard X3.263-1995 FDDI TP-PMD. The purpose of the scrambler is to randomize the 125 Mbps data on transmission resulting in a reduction of the peak amplitudes in the frequency spectrum. The de-scrambler restores the received 5-bit code groups to their unscrambled values.

The scrambler input data (plaintext) is encoded by modulo 2 addition of a key stream to produce a ciphertext bit stream. The key stream is a periodic sequence of 2047 bits generated by the recursive linear function X[n] = X[n-11] + X[n-9] (modulo 2).

If not transmitting data, the scrambler encodes and transmits idles. This allows a pattern to use by the de-scrambler to synchronize and decode the scrambled data.

The implementation of the scrambler and de-scrambler is as shown in Appendix G of the ANSI Standard X3.263-1995.

For test, the scrambler can be bypassed by setting bit 18.5. Scrambler bypass mode is a special type of interface for 100BASE-TX operation that bypasses the scrambler and de-scrambler operation. This mode is typically used for test so that input and output test vectors match. In this mode, idles are not sent and the MAC must provide idles.

MLT-3 Encoder/Decoder: An MLT-3 encoder is used in the transmit path to convert NRZ bit stream data from the PMA sublayer into a three-level code. This encoding results in a reduction in the energy over the critical frequency range. The MLT-3 decoder converts the received three-level code back to an NRZ bit stream.

Baseline Wander: The use of the scrambler and MLT-3 encoding can cause long run lengths of 0s and 1s that can produce a DC component. The DC component cannot be transmitted through the isolation transformers and results in baseline wander. Baseline wander reduces noise immunity because the base line moves nearer to the positive or negative signal comparators. To correct for this EPHY uses DC

restoration to restore the lost DC component of the recovered digital data to correct the baseline wander problem.

Timing Recovery: The timing recovery block locks onto the incoming data stream, extracts the embedded clock, and presents the data synchronized to the recovered clock.

In the event that the receive path is unable to converge to the receive signal, it resets the MSE-good (bit 25.15) signal. The clock synthesizer provides a center frequency reference for operation of the clock recovery circuit in the absence of data.

Adaptive Equalizer: At a data rate of 125 Mbps, the cable introduces significant distortion due to high frequency roll off and phase shift. The high frequency loss is mainly due to skin effect, which causes the conductor resistance to rise as the square of the frequency.

The adaptive equalizer compensates for signal amplitude and phase distortion incurred from transmitting with different cable lengths.

Loopback: If asserted by bit 0.14, data encoded by the MLT3 encoder block is looped back to the MLT3 decoder block while the transmit and receive paths are disconnected from the media.

A second loopback mode for 100BASE-TX is available by setting bit 18.13 (MII loopback) to a logical 1. This loopback mode takes the MII transmit data and loops it directly back to the MII receive pins. Again, the transmit and receive paths are disconnected from the media.

MII loopback has precedence over the digital loopback if both are enabled at the same time.

A third loopback mode is available by setting bit 18.4 high. This analog loopback mode takes the MLT3 encoded data and loops it back through the base line wander and analog receive circuits.

Line Transmitter and Line Receiver: These analog blocks allow EPHY to drive and receive data to/from the 100BASE-TX media. The transmitter is designed to drive a 100-Ω UTP cable.

Link Monitor: The link monitor process is responsible for determining whether the underlying receive channel is providing reliable data. If a failure is found, normal operation is disabled. As specified in the IEEE 802.3 standard, the link is operating reliably if a signal is detected for a period of 330 μs.

Far End Fault: While the auto-negotiation function is disabled, this function is used to exchange fault information between the PHY and the link partner.

19.4.5 Low Power Modes

There are several reduced power configurations available for the EPHY.

19.4.5.1 Stop Mode

If the MCU executes a STOP instruction, the EPHY is powered down and all internal MII registers reset to their default state. Upon exiting stop mode, the EPHY exits the power-down state and latch the values previously written to the EPHYCTL0 and EPHYCTL1 registers. The MII registers have to be re-initialized after the start-up delay ($t_{Start-up}$) has expired.

19.4.5.2 Wait Mode

If the MCU executes a WAIT instruction with the EPHYWAI bit set, the EPHY is powered down and all internal MII registers reset to their default state. Upon exiting STOP mode the EPHY exits the power-down state and latch the values previously written to the EPHYCTL0 and EPHYCTL1 registers. The MII registers must be re-initialized after the start-up delay ($t_{Start\text{-}up}$) has expired.

19.4.5.3 MII Power Down

This mode disconnects the PHY from the network interface (three-state receiver and driver pins).

Setting bit 0.11 of the port enters this mode. In this mode, the management interface is accessible but all internal chip functions are in a zero power state.

In this mode all analog blocks except the PLL clock generator and band gap reference are in low power mode. All digital blocks except the MDIO interface and management registers are inactive.

Chapter 20
DMA Controller Module

20.1 Introduction

This chapter describes the direct memory access (DMA) controller module. It provides an overview of the module and describes in detail its signals and registers. The latter sections of this chapter describe operations, features, and supported data transfer modes in detail.

NOTE

The designation *n* is used throughout this section to refer to registers or signals associated with one of the four identical DMA channels: DMA0, DMA1, DMA2, or DMA3.

20.1.1 Overview

The DMA controller module enables fast transfers of data, providing an efficient way to move blocks of data with minimal processor interaction. The DMA module, shown in Figure 20-1, has four channels that allow byte, word, longword, or 16-byte burst data transfers. Each channel has a dedicated source address register (SAR*n*), destination address register (DAR*n*), byte count register (BCR*n*), control register (DCR*n*), and status register (DSR*n*). Transfers are dual address to on-chip devices, such as UART and GPIOs.

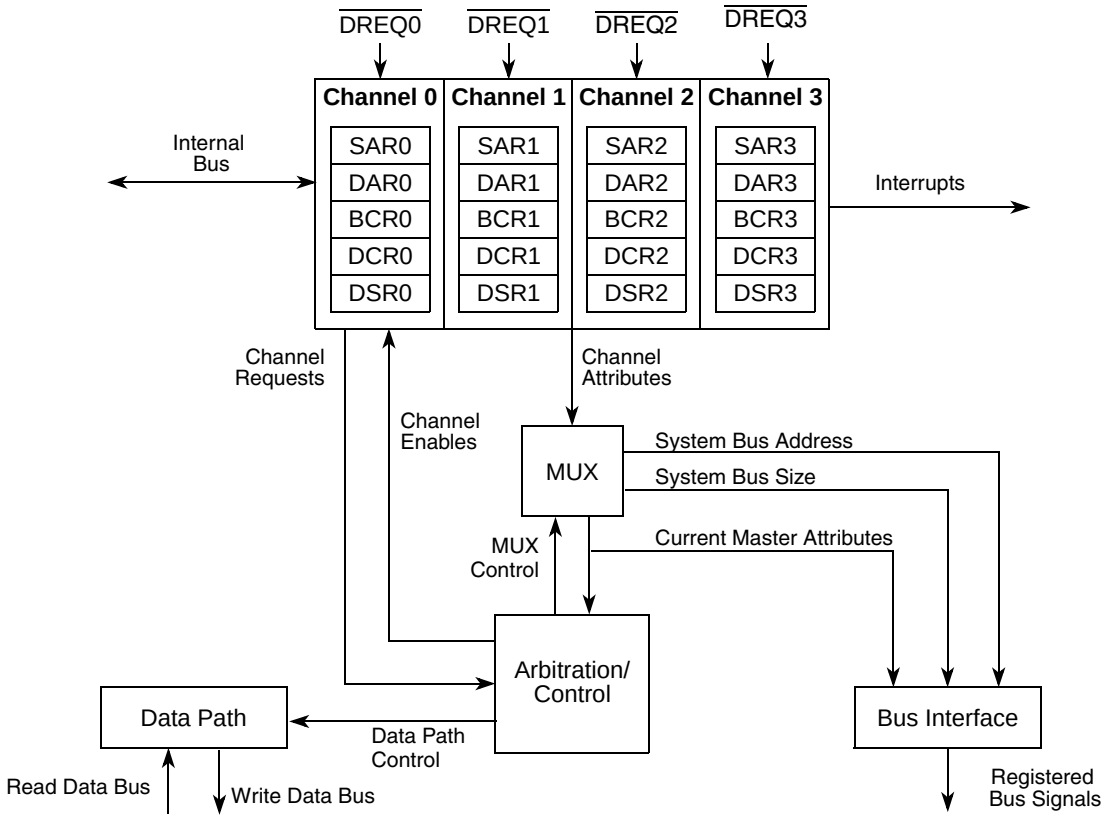


Figure 20-1. DMA Signal Diagram

NOTE

Throughout this chapter, the terms external request and DREQ are used to refer to a DMA request from one of the on-chip UARTS, DMA timers or $\overline{\text{DREQ}}$ signals. For details on the connections associated with DMA request inputs, see Section 20.3.1, "DMA Request Control (DMAREQC)."

20.1.2 Features

The DMA controller module features:

- Four independently programmable DMA controller channels
- Auto-alignment for source or destination accesses
- Dual-address transfers
- Channel arbitration on transfer boundaries
- Data transfers in 8-, 16-, 32-, or 128-bit blocks using a 16-byte buffer
- Continuous-mode or cycle-steal transfers
- Independent transfer widths for source and destination
- Independent source and destination address registers
- Modulo addressing on source and destination addresses
- Automatic channel linking

20.2 DMA Transfer Overview

The DMA module can move data within system memory (including memory and peripheral devices) with minimal processor intervention, greatly improving overall system performance. The DMA module consists of four independent, functionally equivalent channels, so references to DMA in this chapter apply to any of the channels. It is not possible to implicitly address all four channels at once.

The processor generates DMA requests internally by setting DCR[START]; the UART modules and DMA timers can generate a DMA request by asserting internal DREQ signals. The processor can program bus bandwidth for each channel. The channels support cycle-steal and continuous transfer modes; see Section 20.4.1, "Transfer Requests (Cycle-Steal and Continuous Modes)."

The DMA controller supports dual-address transfers. The DMA channels support up to 32 data bits.

- Dual-address transfers—A dual-address transfer consists of a read followed by a write and is initiated by an internal request using the START bit or by a peripheral DMA request. Two types of transfer can occur: a read from a source device or a write to a destination device. See Figure 20-2 for more information.

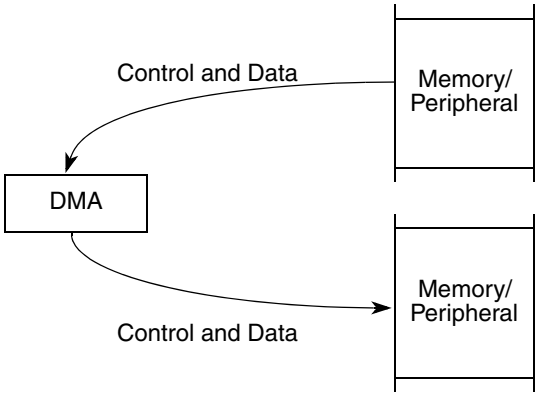


Figure 20-2. Dual-Address Transfer

Any operation involving the DMA module follows the same three steps:

1. Channel initialization—Channel registers are loaded with control information, address pointers, and a byte-transfer count.
2. Data transfer—The DMA accepts requests for operand transfers and provides addressing and bus control for the transfers.
3. Channel termination—Occurs after the operation is finished, successfully or due to an error. The channel indicates the operation status in the channel's DSR, described in Section 20.3.4, "Byte Count Registers (BCRn) and DMA Status Registers (DSRn)."

20.3 Memory Map/Register Definition

This section describes each internal register and its bit assignment. Modifying DMA control registers during a DMA transfer can result in undefined operation. Table 20-1 shows the mapping of DMA controller registers.

Table 20-1. DMA Controller Memory Map

IPSBAR Offset	Register	Width	Access	Reset Value	Section/Page
0x00_0014	DMA request control register (DMAREQC)[1]	32	R/W	0x0000_0000	20.3.1/20-4
0x00_0100 + *n* * 0x10	Source address register *n* (SAR*n*) where *n* = 0–3	32	R/W	0x0000_0000	20.3.2/20-5
0x00_0104 + *n* * 0x10	Destination address register *n* (DAR*n*) where *n* = 0–3	32	R/W	0x0000_0000	20.3.3/20-6
0x00_0108 + *n* * 0x10	DMA status (DSR*n*) and byte count register *n* (BCR*n*) where *n* = 0–3	32	R/W	0x0000_0000	20.3.4/20-6
0x00_010C + *n* * 0x10	DMA control register *n* (DCR*n*) where *n* = 0–3	32	R/W	0x0000_0000	20.3.5/20-8

[1] Located within the SCM, but listed here for clarity.

20.3.1 DMA Request Control (DMAREQC)

The DMAREQC register provides a software-controlled connection matrix for DMA requests. It logically routes DMA requests from the DMA timers and UARTs to the four channels of the DMA controller. Writing to this register determines the exact routing of the DMA request to the four channels of the DMA modules.

If DCR*n*[EEXT] is set and the channel is idle, the assertion of the appropriate external $\overline{\text{DREQ}n}$ signal activates channel *n*.

IPSBAR Access: read/write
Offset: 0x00_0014 (DMAREQC)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMAC3				DMAC2				DMAC1				DMAC0			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-3. DMA Request Control Register (DMAREQC)

Table 20-2. DMAREQC Field Description

Field	Description
15–0 DMAC*n*	DMA channel *n*. Each four bit field defines the logical connection between the DMA requesters and that DMA channel. There are ten possible requesters (4 DMA Timers and 6 UARTs). Any request can be routed to any of the DMA channels. Effectively, the DMAREQC provides a software-controlled routing matrix of the 10 DMA request signals to the 4 channels of the DMA module. DMAC3 controls DMA channel 3, DMAC2 controls DMA channel 2, etc. 0100 DMA Timer 0 0101 DMA Timer 1 0110 DMA Timer 2 0111 DMA Timer 3 1000 UART0 Receive 1001 UART1 Receive 1010 UART2 Receive 1100 UART0 Transmit 1101 UART1 Transmit 1110 UART2 Transmit All other values are reserved and do not generate a DMA request.

20.3.2 Source Address Registers (SAR*n*)

SAR*n*, shown in Figure 20-4, contains the address from which the DMA controller requests data.

IPSBAR 0x00_0100 (SAR0) Access: read/write
Offset: 0x00_0110 (SAR1)
0x00_0120 (SAR2)
0x00_0130 (SAR3)

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

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

Figure 20-4. Source Address Registers (SAR*n*)

NOTE

The backdoor enable bit must be set in the SCM RAMBAR, as well as the secondary port valid bit in the core RAMBAR to enable backdoor accesses from the DMA to SRAM. See Section 14.5.2, “Memory Base Address Register (RAMBAR)and Section 11.2.1, “SRAM Base Address Register (RAMBAR),” for more details.

20.3.3 Destination Address Registers (DAR*n*)

DAR*n* holds the address to which the DMA controller sends data.

IPSBAR 0x00_0104 (DAR0) Access: Read/write
Offset: 0x00_0114 (DAR1)
0x00_0124 (DAR2)
0x00_0134 (DAR3)

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

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

Figure 20-5. Destination Address Registers (DAR*n*)

20.3.4 Byte Count Registers (BCR*n*) and DMA Status Registers (DSR*n*)

The BCR*n* and DSR*n* registers are two logical registers that occupy one 32-bit register, as shown in Figure 20-6. The address used to access both registers is the same; DSR*n* occupies bits 31–24, and BCR*n* occupies bits 23–0. BCR*n* contains the number of bytes yet to be transferred for a given block. BCR*n* decrements on the successful completion of the address transfer of a write transfer. BCR*n* decrements by 1, 2, 4, or 16 for byte, word, longword, or line accesses, respectively.

IPSBAR 0x00_0108 (BCR0/DSR0) Access: Read/write
Offset: 0x00_0118 (BCR1/DSR1)
0x00_0128 (BCR2/DSR2)
0x00_0138 (BCR3/DSR3)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DSR								BCR							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Figure 20-6. Byte Count Registers (BCR*n*) and DMA Status Registers (DSR*n*)

The fields of the DSR*n* register (bits 31–24 in Figure 20-6) are shown in Figure 20-7. In response to an event, the DMA controller writes to the appropriate DSR*n* bit. Only a write to DSR*n*[DONE] results in action. DSR*n*[DONE] is set when the block transfer is complete.

When a transfer sequence is initiated and BCR*n*[BCR] is not a multiple of 16, 4, or 2 when the DMA is configured for line, longword, or word transfers, respectively, DSR*n*[CE] is set and no transfer occurs.

IPSBAR 0x00_0108 (DSR0) Access: Read/write
Offsets: 0x00_0118 (DSR1)
0x00_0128 (DSR2)
0x00_0138 (DSR3)

	7	6	5	4	3	2	1	0
R	0	CE	BES	BED	0	REQ	BSY	DONE
W								
Reset:	0	0	0	0	0	0	0	0

Figure 20-7. DMA Status Registers (DSR*n*)

Table 20-3. DSR*n* Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CE	Configuration error. Occurs when BCR, SAR, or DAR does not match the requested transfer size, or if BCR equals 0 when the DMA receives a start condition. CE is cleared at hardware reset or by writing a 1 to DSR[DONE]. 0 No configuration error exists. 1 A configuration error has occurred.
5 BES	Bus error on source 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the read portion of a transfer.
4 BED	Bus error on destination 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the write portion of a transfer.
3	Reserved, should be cleared.
2 REQ	Request 0 No request is pending or the channel is currently active. Cleared when the channel is selected. 1 The DMA channel has a transfer remaining and the channel is not selected.
1 BSY	Busy 0 DMA channel is inactive. Cleared when the DMA has finished the last transaction. 1 BSY is set the first time the channel is enabled after a transfer is initiated.
0 DONE	Transactions done. Set when all DMA controller transactions complete, as determined by transfer count or error conditions. When BCR reaches zero, DONE is set when the final transfer completes successfully. DONE can also be used to abort a transfer by resetting the status bits. When a transfer completes, software must clear DONE before reprogramming the DMA. 0 Writing or reading a 0 has no effect. 1 DMA transfer completed. Writing a 1 to this bit clears all DMA status bits and can be used in an interrupt handler to clear the DMA interrupt and error bits.

20.3.5 DMA Control Registers (DCR*n*)

The DMA control registers (DCR*n*) are described in Figure 20-8 and Table 20-4.

IPSBAR 0x00_010C (DCR0)
Offsets: 0x00_011C (DCR1)
0x00_012C (DCR2)
0x00_013C (DCR3)

Access: Read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	INT	EEXT	CS	AA	BWC			0	0	SINC	SSIZE		DINC	DSIZE		0
W																START
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SMOD				DMOD				D_REQ	0	LINKCC		LCH1		LCH2	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-8. DMA Control Registers (DCR*n*)

Table 20-4. DCR*n* Field Descriptions

Field	Description
31 INT	Interrupt on completion of transfer. Determines whether an interrupt is generated by completing a transfer or by the occurrence of an error condition. 0 No interrupt is generated. 1 Internal interrupt signal is enabled.
30 EEXT	Enable external request. Care should be taken because a collision can occur between the START bit and $\overline{\text{DREQ}n}$ when EEXT equals 1. 0 External request is ignored. 1 Enables external request to initiate transfer. The internal request (initiated by setting the START bit) is always enabled.
29 CS	Cycle steal. 0 DMA continuously makes read/write transfers until the BCR decrements to 0. 1 Forces a single read/write transfer per request.
28 AA	Auto-align. AA and SIZE determine whether the source or destination is auto-aligned, that is, transfers are optimized based on the address and size. See Section 20.4.4.1, “Auto-Alignment.” 0 Auto-align disabled 1 If SSIZE indicates a transfer no smaller than DSIZE, source accesses are auto-aligned; otherwise, destination accesses are auto-aligned. Source alignment takes precedence over destination alignment. If auto-alignment is enabled, the appropriate address register increments, regardless of DINC or SINC.

Table 20-4. DCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>27–25
BWC</td><td>Bandwidth control. Indicates the number of bytes in a block transfer. When the byte count reaches a multiple of the BWC value, the DMA releases the bus.
<table>
<tr><th>BWC</th><th>Number of kilobytes per block</th></tr>
<tr><td>000</td><td>DMA has priority and does not negate its request until transfer completes.</td></tr>
<tr><td>001</td><td>16 Kbytes</td></tr>
<tr><td>010</td><td>32 Kbytes</td></tr>
<tr><td>011</td><td>64 Kbytes</td></tr>
<tr><td>100</td><td>128 Kbytes</td></tr>
<tr><td>101</td><td>256 Kbytes</td></tr>
<tr><td>110</td><td>512 Kbytes</td></tr>
<tr><td>111</td><td>1024 Kbytes</td></tr>
</table></td></tr>
<tr><td>24–23</td><td>Reserved, should be cleared.</td></tr>
<tr><td>22
SINC</td><td>Source increment. Controls whether a source address increments after each successful transfer.
0 No change to SAR after a successful transfer.
1 The SAR increments by 1, 2, 4, or 16, as determined by the transfer size.</td></tr>
<tr><td>21–20
SSIZE</td><td>Source size. Determines the data size of the source bus cycle for the DMA control module.
00 Longword
01 Byte
10 Word
11 Line (16-byte burst)</td></tr>
<tr><td>19
DINC</td><td>Destination increment. Controls whether a destination address increments after each successful transfer.
0 No change to the DAR after a successful transfer.
1 The DAR increments by 1, 2, 4, or 16, depending upon the size of the transfer.</td></tr>
<tr><td>18–17
DSIZE</td><td>Destination size. Determines the data size of the destination bus cycle for the DMA controller.
00 Longword
01 Byte
10 Word
11 Line (16-byte burst)</td></tr>
<tr><td>16
START</td><td>Start transfer.
0 DMA inactive
1 The DMA begins the transfer in accordance to the values in the control registers. START is cleared automatically after one system clock and is always read as logic 0.</td></tr>
</table>

Table 20-4. DCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–12
SMOD</td><td>Source address modulo. Defines the size of the source data circular buffer used by the DMA Controller. If enabled (SMOD is non-zero), the buffer base address is located on a boundary of the buffer size. The value of this boundary is based upon the initial source address (SAR). The base address should be aligned to a 0-modulo-circular buffer size boundary. Misaligned buffers are not possible. The boundary is forced to the value determined by the upper address bits in the field selection.
<table>
<tr><th>SMOD</th><th>Circular Buffer Size</th></tr>
<tr><td>0000</td><td>Buffer Disabled</td></tr>
<tr><td>0001</td><td>16 Bytes</td></tr>
<tr><td>0010</td><td>32 Bytes</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>1111</td><td>256 Kbytes</td></tr>
</table></td></tr>
<tr><td>11–8
DMOD</td><td>Destination address modulo. Defines the size of the destination data circular buffer used by the DMA Controller. If enabled (DMOD value is non-zero), the buffer base address is located on a boundary of the buffer size. The value of this boundary depends on the initial destination address (DAR). The base address should be aligned to a 0-modulo- circular buffer size boundary. Misaligned buffers are not possible. The boundary is forced to the value determined by the upper address bits in the field selection.
<table>
<tr><th>DMOD</th><th>Circular Buffer Size</th></tr>
<tr><td>0000</td><td>Buffer Disabled</td></tr>
<tr><td>0001</td><td>16 Bytes</td></tr>
<tr><td>0010</td><td>32 Bytes</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>1111</td><td>256 Kbytes</td></tr>
</table></td></tr>
<tr><td>7
D_REQ</td><td>Disable request. DMA hardware automatically clears the corresponding DCRn[EEXT] bit when the byte count register reaches zero.
0 EEXT bit is not affected.
1 EEXT bit is cleared when the BCR is exhausted.</td></tr>
<tr><td>6</td><td>Reserved; should be cleared.</td></tr>
</table>

Table 20-4. DCR*n* Field Descriptions (continued)

Field	Description
5–4 LINKCC	Link channel control. Allows DMA channels to have their transfers linked. The current DMA channel triggers a DMA request to the linked channels (LCH1 or LCH2) depending on the condition described by the LINKCC bits. 00 No channel-to-channel linking 01 Perform a link to channel LCH1 after each cycle-steal transfer followed by a link to LCH2 after the BCR decrements to zero. 10 Perform a link to channel LCH1 after each cycle-steal transfer 11 Perform a link to channel LCH1 after the BCR decrements to zero If not in cycle steal mode (DCR*n*[CS]=0) and LINKCC equals 01 or 10, no link to LCH1 occurs. If LINKCC equals 01, a link to LCH1 is created after each cycle-steal transfer performed by the current DMA channel is completed. As the last cycle-steal is performed and the BCR reaches zero, then the link to LCH1 is closed and a link to LCH2 is created. If the LINKCC field is non-zero, the contents of the bandwidth control field (DCRn[BWC]) are ignored and effectively forced to zero by the DMA hardware. This is done to prevent any non-zero bandwidth control settings from allowing channel arbitration while any type of link is to be performed.
3-2 LCH1	Link channel 1. Indicates the DMA channel assigned as link channel 1. The link channel number cannot be the same as the currently executing channel, and generates a configuration error if this is attempted (DSR*n*[CE] is set). 00 DMA Channel 0 01 DMA Channel 1 10 DMA Channel 2 11 DMA Channel 3
1-0 LCH2	Link channel 2. Indicates the DMA channel assigned as link channel 2. The link channel number cannot be the same as the currently executing channel, and generates a configuration error if this is attempted (DSR*n*[CE] is set). 00 DMA Channel 0 01 DMA Channel 1 10 DMA Channel 2 11 DMA Channel 3

20.4 Functional Description

In the following discussion, the term DMA request implies that DCR*n*[START or EEXT] is set, followed by assertion of an internal or external DMA request. The START bit is cleared when the channel begins an internal access.

Before initiating a dual-address access, the DMA module verifies that DCR*n*[SSIZE,DSIZE] are consistent with the source and destination addresses. If they are not consistent, the configuration error bit, DSR*n*[CE], is set. If misalignment is detected, no transfer occurs, DSR*n*[CE] is set, and, depending on the DCR configuration, an interrupt event is issued. If the auto-align bit, DCR*n*[AA], is set, error checking is performed on the appropriate registers.

A read/write transfer reads bytes from the source address and writes them to the destination address. The number of bytes is the larger of the sizes specified by DCR*n*[SSIZE] and DCR*n*[DSIZE]. See 20.3.5, “DMA Control Registers (DCRn)”.

Source and destination address registers (SAR*n* and DAR*n*) can be programmed in the DCR*n* to increment at the completion of a successful transfer.

20.4.1 Transfer Requests (Cycle-Steal and Continuous Modes)

The DMA channel supports internal and external requests. A request is issued by setting DCR*n*[START] or when a UART or DMA timer asserts a DMA request. Setting DCR*n*[EEXT] enables recognition of external DMA requests. Selecting between cycle-steal and continuous modes minimizes bus usage for internal or external requests.

- Cycle-steal mode (DCR*n*[CS] = 1)—Only one complete transfer from source to destination occurs for each request. If DCR*n*[EEXT] is set, a request can be internal or external. An internal request is selected by setting DCR*n*[START]. An external request is initiated by an on-chip peripheral while DCR*n*[EEXT] is set.
- Continuous mode (DCR*n*[CS] = 0)—After an internal or external request, the DMA continuously transfers data until BCR*n* reaches zero or a multiple of DCR*n*[BWC] or until DSR*n*[DONE] is set. If BCR*n* is a multiple of BWC, the DMA request signal is negated until the bus cycle terminates to allow the internal arbiter to switch masters. DCR*n*[BWC] equaling 000 specifies the maximum transfer rate; other values specify a transfer rate limit.

  The DMA performs the specified number of transfers, then relinquishes bus control. The DMA negates its internal bus request on the last transfer before BCR*n* reaches a multiple of the boundary specified in BWC. Upon completion, the DMA reasserts its bus request to regain mastership at the earliest opportunity. The DMA loses bus control for a minimum of one bus cycle.

20.4.2 Dual-Address Data Transfer Mode

Each channel supports dual-address transfers. Dual-address transfers consist of a source data read and a destination data write. The DMA controller module begins a dual-address transfer sequence during a DMA request. If no error condition exists, DSR*n*[REQ] is set.

- Dual-address read—The DMA controller drives the SAR*n* value onto the internal address bus. If DCR*n*[SINC] is set, the SAR*n* increments by the appropriate number of bytes upon a successful read cycle. When the appropriate number of read cycles complete (multiple reads if the destination size is larger than the source), the DMA initiates the write portion of the transfer.

  If a termination error occurs, DSR*n*[BES, DONE] are set and DMA transactions stop.
- Dual-address write—The DMA controller drives the DAR*n* value onto the address bus. If DCR*n*[DINC] is set, DAR*n* increments by the appropriate number of bytes at the completion of a successful write cycle. BCR*n* decrements by the appropriate number of bytes. DSR*n*[DONE] is set when BCR*n* reaches zero. If the BCR*n* is greater than zero, another read/write transfer is initiated. If the BCR*n* is a multiple of DCR*n*[BWC], the DMA request signal is negated until termination of the bus cycle to allow the internal arbiter to switch masters.

  If a termination error occurs, DSR*n*[BED, DONE] are set and DMA transactions stop.

20.4.3 Channel Initialization and Startup

Before a block transfer starts, channel registers must be initialized with information describing configuration, request-generation method, and the data block.

20.4.3.1 Channel Prioritization

The four DMA channels are prioritized in ascending order (channel 0 having highest priority and channel 3 having the lowest) or in an order determined by DCR*n*[BWC]. If the BWC encoding for a DMA channel is 000, that channel has priority only over the channel immediately preceding it. For example, if DCR3[BWC] equals 000, DMA channel 3 has priority over DMA channel 2 (assuming DCR2[BWC] ≠ 000), but not over DMA channel 1.

If DCR0[BWC] equals 000 and DCR1[BWC] equals 000, DMA0 continues having priority over DMA1. In this case, DCR1[BWC] equals 000 does not affect prioritization.

Simultaneous external requests are prioritized in ascending order or in an order determined by each channel's DCR*n*[BWC] bits.

20.4.3.2 Programming the DMA Controller Module

General guidelines for programming the DMA are:

- No mechanism exists within the DMA module itself to prevent writes to control registers during DMA accesses.
- If the DCR*n*[BWC] value of sequential channels are equal, the channels are prioritized in ascending order.

The DMAREQC register is configured to assign peripheral DMA requests to the individual DMA channels.

The SAR*n* is loaded with the source (read) address. If the transfer is from a peripheral device to memory, the source address is the location of the peripheral data register. If the transfer is from memory to a peripheral device or memory, the source address is the starting address of the data block. This can be any aligned byte address.

The DAR*n* should contain the destination (write) address. If the transfer is from a peripheral device to memory, or from memory to memory, the DAR*n* is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, DAR*n* is loaded with the address of the peripheral data register. This address can be any aligned byte address.

SAR*n* and DAR*n* change after each cycle depending on DCR*n*[SSIZE,DSIZE, SINC,DINC,SMOD,DMOD] and on the starting address. Increment values can be 1, 2, 4, or 16 for byte, word, longword, or 16-byte line transfers, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the data transfer.

BCR*n*[BCR] must be loaded with the number of byte transfers to occur. It is decremented by 1, 2, 4, or 16 at the end of each transfer, depending on the transfer size. DSR*n*[DONE] must be cleared for channel startup.

As soon as the channel has been initialized, it is started by writing a one to DCR*n*[START], or a peripheral DMA request, depending on the status of DCR*n*[EEXT]. Programming the channel for internal requests causes the channel to request the bus and start transferring data immediately. If the channel is programmed for external request, a peripheral DMA request must be asserted before the channel requests the bus.

Changes to DCR*n* are effective immediately while the channel is active. To avoid problems with changing a DMA channel setup, write a one to DSR*n*[DONE] to stop the DMA channel.

20.4.4 Data Transfer

This section describes auto-alignment and bandwidth control for DMA transfers.

.

20.4.4.1 Auto-Alignment

Auto-alignment allows block transfers to occur at the optimal size based on the address, byte count, and programmed size. To use this , DCR*n*[AA] must be set. The source is auto-aligned if DCR*n*[SSIZE] indicates a transfer size larger than DCR*n*[DSIZE]. Source alignment takes precedence over the destination when the source and destination sizes are equal. Otherwise, the destination is auto-aligned. The address register chosen for alignment increments regardless of the increment value. Configuration error checking is performed on registers not chosen for alignment.

If BCR*n* is greater than 16, the address determines transfer size. Bytes, words, or longwords are transferred until the address is aligned to the programmed size boundary, at which time accesses begin using the programmed size.

If BCR*n* is less than 16 at the start of a transfer, the number of bytes remaining dictates transfer size. For example, AA equals 1, SAR*n* equals 0x0001, BCR*n* equals 0x00F0, SSIZE equals 00 (longword), and DSIZE equals 01 (byte). Because SSIZE > DSIZE, the source is auto-aligned. Error checking is performed on destination registers. The access sequence is as follows:

1. Read byte from 0x0001—write 1 byte, increment SAR*n*.
2. Read word from 0x0002—write 2 bytes, increment SAR*n*.
3. Read longword from 0x0004—write 4 bytes, increment SAR*n*.
4. Repeat longwords until SAR*n* = 0x00F0.
5. Read byte from 0x00F0—write byte, increment SAR*n*.

If DSIZE is another size, data writes are optimized to write the largest size allowed based on the address, but not exceeding the configured size.

20.4.4.2 Bandwidth Control

Bandwidth control makes it possible to force the DMA off the bus to allow access to another device. DCR*n*[BWC] provides seven levels of block transfer sizes. If the BCR*n* decrements to a multiple of the decode of the BWC, the DMA bus request negates until the bus cycle terminates. If a request is pending, the arbiter may then pass bus mastership to another device. If auto-alignment is enabled,

DCR*n*[AA] equals 1, the BCR*n* may skip over the programmed boundary, in which case, the DMA bus request is not negated.

If BWC equals 000, the request signal remains asserted until BCR*n* reaches zero. DMA has priority over the core. In this scheme, the arbiter can always force the DMA to relinquish the bus.

20.4.5 Termination

An unsuccessful transfer can terminate for one of the following reasons:

- Error conditions—When the DMA encounters a read or write cycle that terminates with an error condition, DSR*n*[BES] is set for a read and DSR*n*[BED] is set for a write before the transfer is halted. If the error occurred in a write cycle, data in the internal holding register is lost.
- Interrupts—If DCR*n*[INT] is set, the DMA drives the appropriate internal interrupt signal. The processor can read DSR*n* to determine whether the transfer terminated successfully or with an error. DSR*n*[DONE] is then written with a one to clear the interrupt and the DONE and error bits.

Chapter 21
EzPort

EzPort is a serial flash programming interface that allows the flash memory contents on a 32-bit general purpose microcontroller to be read, erased, and programmed from off-chip in a compatible format to many standalone flash memory chips.

21.1 Features

The EzPort includes the following features:

- Serial interface that is compatible with a subset of the SPI format
- Ability to read, erase, and program flash memory
- Ability to reset the micro-controller, allowing it to boot from the flash memory after the memory has been configured

The EzPort allows the flash memory internal to the controller to be programmed like standard SPI flash memories available from ST Microelectronics, Macronix, Spansion, and other vendors. The EzPort implements the same command set as devices from these vendors, so existing microcontroller or automated test equipment code used to program these devices can also be used to program the device with little or no modification. In essence, the EzPort eliminates the need to use the background debug mode interface to download and run user-developed flash programming code to initialize

21.2 Modes of Operation

The EzPort can operate in one of two different modes:

- Enabled—When enabled, the EzPort steals access to the flash memory, preventing access from other cores or peripherals. The rest of the micro-controller is disabled when the EzPort is enabled to avoid conflicts.
- Disabled—When the EzPort is disabled, the rest of the micro-controller can access flash memory as normal.

Figure 21-1 is a block diagram of the EzPort.

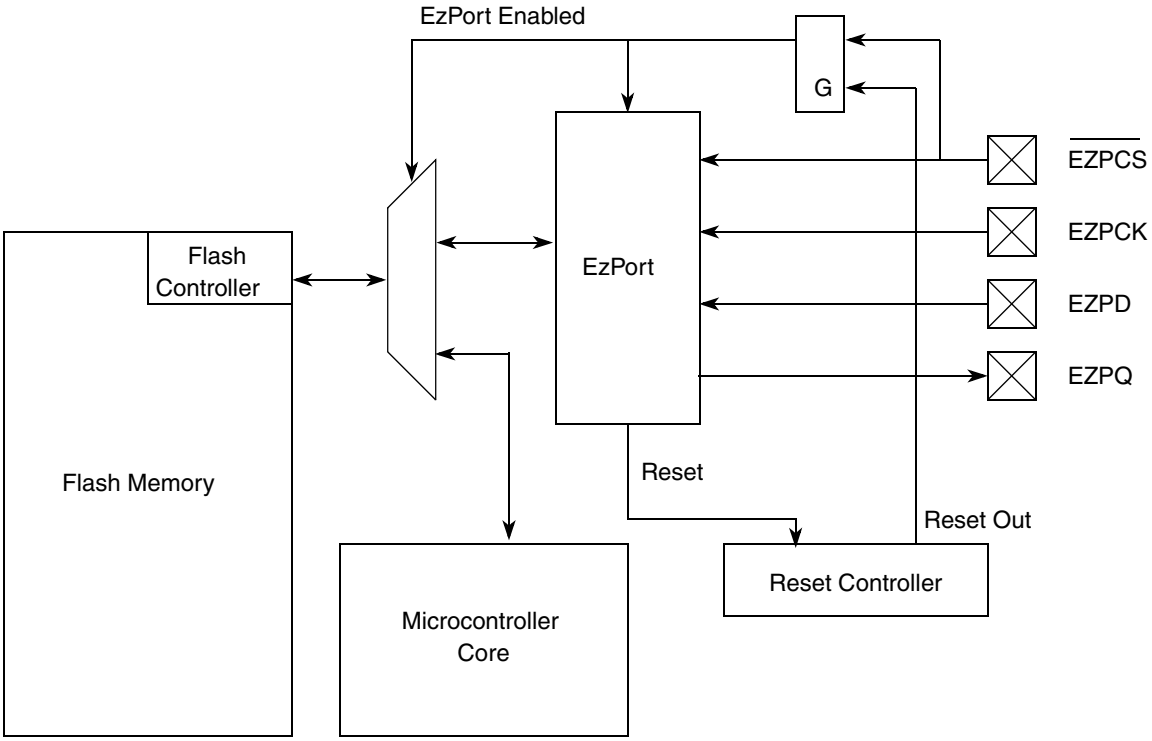


Figure 21-1. EzPort Block Diagram

21.3 External Signal Description

21.3.1 Overview

Table 21-1 contains a list of EzPort external signals.

Table 21-1. Signal Descriptions

Name	Description	I/O
EZPCK	EzPort Clock	Input
$\overline{\text{EZPCS}}$	EzPort Chip Select	Input
EZPD	EzPort Serial Data In	Input
EZPQ	EzPort Serial Data Out	Output

21.3.2 Detailed Signal Descriptions

21.3.2.1 EZPCK — EzPort Clock

EzPort clock (EZPCK) is the serial clock for data transfers. Serial data in (EZPD) and chip select ($\overline{\text{EZPCS}}$) are registered on the rising edge of EZPCK while serial data out (EZPQ) is driven on the falling edge of

EZPCK.The maximum frequency of the EzPort clock is half the system clock frequency for all commands except when executing the read data command. When executing the Read Data command, the EzPort clock has a maximum frequency of one eighth the system clock frequency.

21.3.2.2 $\overline{\text{EZPCS}}$ — EzPort Chip Select

EzPort chip select ($\overline{\text{EZPCS}}$) is the chip select for signalling the start and end of serial transfers. If $\overline{\text{EZPCS}}$ is asserted during and when the micro-controller's reset out signal is negated, then EzPort is enabled out of reset; otherwise it is disabled. After EzPort is enabled, asserting $\overline{\text{EZPCS}}$ commences a serial data transfer, which continues until $\overline{\text{EZPCS}}$ is negated again. The negation of $\overline{\text{EZPCS}}$ indicates the current command is finished and resets the EzPort state machine so that it is ready to receive the next command.

21.3.2.3 EZPD — EzPort Serial Data In

EzPort serial data in (EZPD) is the serial data in for data transfers. It is registered on the rising edge of EZPCK. All commands, addresses, and data are shifted in most significant bit first. When EzPort is driving output data on EZPQ, the data shifted in EZPD is ignored.

21.3.2.4 EZPQ — EzPort Serial Data Out

EzPort serial data out (EZPQ) is the serial data out for data transfers. It is driven on the falling edge of EZPCK. It is tri-stated, unless $\overline{\text{EZPCS}}$ is asserted and the EzPort is driving data out. All data is shifted out most significant bit first.

21.4 Command Definition

The EzPort receives commands from an external device and translates those commands into flash memory accesses. Table 21-2 lists the supported commands.

Table 21-2. EzPort Commands

Command	Description	Code	Address Bytes	Dummy Bytes	Data Bytes	Compatible Commands[1]
WREN	Write Enable	0x06	0	0	0	WREN
WRDI	Write Disable	0x04	0	0	0	WRDI
RDSR	Read Status Register	0x05	0	0	1	RDSR
WRCR	Write Config Register	0x01	0	0	1	WRSR
READ	Read Data	0x03	3	0	1+	READ
FAST_READ	Read Data at High Speed	0x0B	3	1	1+	FAST_READ
PP	Page Program	0x02	3	0	4 to 256	PP
SE	Sector Erase	0xD8	3	0	0	SE
BE	Bulk Erase	0xC7	0	0	0	BE
RESET	Reset Chip	0xB9	0	0	0	DP

[1]Lists the compatible commands on the ST Microelectronics Serial Flash Memory parts.

21.4.1 Command Descriptions

21.4.1.1 Write Enable

The Write Enable command sets the write enable register bit in the status register. The write enable bit must be set for a Write Configuration Register (WRCR), Page Program (PP), Sector Erase (SE), or Bulk Erase (BE) command to be accepted. The write enable register bit clears on reset, on a Write Disable command, and at the completion of a write, program, or erase command.

This command should not be used if a write is already in progress.

21.4.1.2 Write Disable

The Write Disable command clears the write enable register bit in the status register.

This command should not be used if a write is already in progress.

21.4.1.3 Read Status Register

The Read Status Register command returns the contents of the EzPort Status register.

IPSBAR Offset: Access: read/write

	7	6	5	4	3	2	1	0
R	FS	WEF	CRL				WEN	WIP
W								
Reset:	0/1[1]	0	0	0	0	0	0	0

Figure 21-2. EzPort Status Register

[1]Reset value reflects if flash security is enabled or disabled out of reset.

Table 21-3. EzPort Status Register Field Description

Field	Descriptions
7 FS	Flash Security. Status flag that indicates if the flash memory is in secure mode. In secure mode, the following commands are not accepted: Read (READ), Fast Read (FAST_READ), Page Program (PP), Sector Erase (SE). Secure mode can be exited by performing a Bulk Erase (BE) command, which erases the entire contents of the flash memory. 0 Flash is not in secure mode. 1 Flash is in secure mode.
6 WEF	Write Error Flag. Status flag that indicates if there has been an error with an erase or program instruction inside the flash controller due to attempting to program or erase a protected sector, or if there is an error in the flash memory after performing a Bulk Erase command. The flag clears after a Read Status Register (RDSR) command. 0 No error on previous erase/program command. 1 Error on previous erase/program command.

Table 21-3. EzPort Status Register Field Description (continued)

Field	Descriptions
5 CRL	Configuration Register Loaded. Status flag that indicates if the configuration register has been loaded. The configuration register initializes the flash controllers clock configuration register to generate a divided down clock from the system clock that runs at a frequency of 150 kHz to 200 kHz. This register must be initialized before any erase or program commands are accepted. 0 Configuration register has not been loaded; erase and program commands are not accepted. 1 Configuration register has been loaded; erase and program commands are accepted.
4–2 —	Reserved, should be cleared.
1 WEN	Write Enable. Control bit that must be set before a Write Configuration Register (WRCR), Page Program (PP), Sector Erase (SE), or Bulk Erase (BE) command is accepted. Is set by the Write Enable (WREN) command and cleared by reset or a Write Disable (WRDI) command. It also clears on completion of a write, erase, or program command. 0 Disables the following write, erase, or program command. 1 Enables the following write, erase, or program command.
0 WIP	Write In Progress. Status flag that sets after a Write Configuration Register (WRCR), Page Program (PP), Sector Erase (SE), or Bulk Erase (BE) command is accepted and clears after the flash memory erase or program is completed. Only the Read Status Register (RDSR) command is accepted while a write is in progress. 0 Write is not in progress. Accept any command. 1 Write is in progress. Only accept RDSR command.

21.4.1.4 Write Configuration Register

The Write Configuration Command updates the flash controller's clock configuration register. The clock configuration register divides down the flash controller's internal system clock to a 150 kHz to 200 kHz clock. This register must be initialized before any erase or program commands are issued to the flash controller.

This command should not be used if the write error flag is set, a write is in progress, or the configuration register has already been loaded (as it is a write-once register).

IPSBAR Offset: Access: read/write

	7	6	5	4	3	2	1	0
R								
W		PRDIV8	DIV[5:0]					
Reset:	0	0	0	0	0	0	0	0

Figure 21-3. EzPort Configuration Register

Table 21-4. EzPort Configuration Register Field Description

Field	Descriptions
7 —	Reserved, should be cleared.
6 PRDIV	Enables prescaler divide by 8. 0 The system clock is fed directly into the divider. 1 Enables a prescaler that divides the system clock by 8 before it enters the divider.
5–0 DIV[5:0]	Clock divider field. The combination of PRDIV8 and DIV[5:0] effectively divides the system clock down to a frequency between 150 kHz and 200 kHz.

21.4.1.5 Read Data

The Read Data command returns data from the flash memory, starting at the address specified in the command word. Data continues being returned for as long as the EzPort chip select ($\overline{\text{EZPCS}}$) is asserted, with the address automatically incrementing. When the address reaches the highest flash memory address, it wraps around to the lowest flash memory address. In this way, the entire contents of the flash memory can be returned by one command.

For this command to return the correct data, the EzPort Clock (EZPCK) must run at no more than divide by eight of the internal system clock.

This command should not be used if the write error flag is set, or a write is in progress. This command is not accepted if flash security is enabled.

21.4.1.6 Read Data at High Speed

This command is identical to the Read Data command, except for the inclusion of a dummy byte following the address bytes and before the first data byte is returned.

This allows the command to run at any frequency of the EzPort Clock (EZPCK) up to and including half the internal system clock frequency of the micro-controller.This command should not be used if the write error flag is set, or a write is in progress. This command is not accepted if flash security is enabled.

21.4.1.7 Page Program

The Page Program command programs locations in flash memory that have previously been erased. The starting address of the memory to program is sent after the command word and must be a 32-bit aligned address (the two LSBs must be zero). After every four bytes of data are received by the EzPort, that 32-bit word is programmed into flash memory with the address automatically incrementing after each write. For this reason, the number of bytes to program must be a multiple of four. Only a maximum of 256 bytes can be programmed at a time; when the address reaches the highest address within any given 256-byte space of memory, it wraps around to the lowest address in that same space.

This command should not be used if the write error flag is set, a write is in progress, the write enable bit is not set, or the configuration register has not been written.This command is not accepted if flash security is enabled.

The write error flag sets if there is an attempt to program a protected area of the flash memory.

21.4.1.8 Sector Erase

The Sector Erase command erases the contents of a 2-Kbyte space of flash memory. The 3-byte address sent after the command byte can be any address within the space to erase.

This command should not be used if the write error flag is set, a write is in progress, the write enable bit is not set, or the configuration register has not been written. This command is not accepted if flash security is enabled.

The write error flag sets if there is an attempt to erase a protected area of the flash memory.

21.4.1.9 Bulk Erase

The Bulk Erase command erases the entire contents of flash memory, ignoring any protected sectors or flash security. The write error flag sets if the Bulk Erase command does not successfully erase the entire contents of flash memory. Flash security is disabled if the Bulk Erase command is followed by a Reset Chip command.

This command should not be used if the write error flag is set, a write is in progress, the write enable bit is not set, or the configuration register has not been written.

21.4.1.10 Reset Chip

The Reset Chip command forces the chip into the reset state. If the EzPort chip select ($\overline{\text{EZPCS}}$) pin is asserted at the end of the reset period, then EzPort is enabled; otherwise it is disabled.

This command allows the chip to boot up from flash memory after it has been programmed by an external source.

This command should not be used if a write is in progress.

21.5 Functional Description

The EzPort provides a simple interface to connect an external device to the flash memory on board a 32 bit microcontroller.

The interface itself is compatible with the SPI interface (with the EzPort operating as a slave) running in either of the two following modes with data transmitted most significant bit first:

- CPOL = 0, CPHA = 0
- CPOL = 1, CPHA = 1

Commands are issued by the external device to erase, program, or read the contents of the flash memory. The serial data out from the EzPort is tri-stated unless data is being driven, allowing the signal to be shared among several different EzPort (or compatible) devices in parallel, provided they have different chip selects.

21.6 Initialization/Application Information

Prior to issuing any program or erase commands, the CFMCLKD register (see Section 17.3.3.2, “CFMCLKD — CFM Clock Divider Register,” on page 17-8) must be written to set the flash state machine clock (FCLK). The flash controller module runs at the system clock frequency divided by 2, but FCLK must be divided down from this frequency to a frequency between 150 kHz and 200 kHz. Use the following procedure to set the PRDIV8 and DIV[5:0] bits in the clock configuration register.

1. If $f_{SYS/2}$ is greater than 25.6 MHz, PRDIV8 equals 1; otherwise, PRDIV8 equals 0.
2. Determine DIV[5:0] by using the following equation. Keep only the integer portion of the result and discard any fraction. Do not round the result.

$$DIV = \frac{Fsys}{2 \times 200kHz \times (1 + (PRDIV8 \times 7))}$$

3. The flash state machine clock is:

$$FCLK = \frac{Fsys}{2 \times (DIV + 1) \times (1 + (PRDIV8 \times 7))}$$

For Fsys equaling 60 MHz, DIV equals 18 (0b00010010) using the above equations, and writing 0x52 (0b01010010, i.e. including the PRDIV8 bit) to CFMCLKD sets FCLK to 197.37 kHz. This is a valid frequency for the timing of program and erase operations.

For proper program and erase operations, it is critical to set FCLK between 150 kHz and 200 kHz. Array damage due to overstress can occur when FCLK is less than 150 kHz. Incomplete programming and erasure can occur when FCLK is greater than 200 kHz.

Chapter 22
Programmable Interrupt Timers (PIT0–PIT1)

22.1 Introduction

This chapter describes the operation of the two programmable interrupt timer modules: PIT0–PIT1.

22.1.1 Overview

Each PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can count down from the value written in the modulus register or it can be a free-running down-counter.

22.1.2 Block Diagram

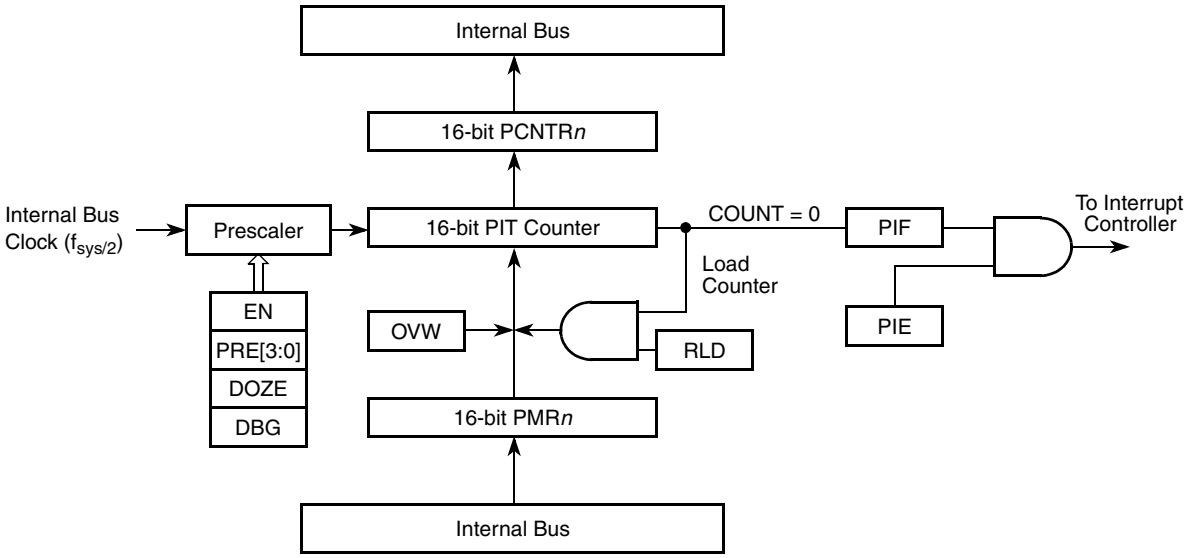


Figure 22-1. PIT Block Diagram

22.1.3 Low-Power Mode Operation

This subsection describes the operation of the PIT modules in low-power modes and debug mode of operation. Low-power modes are described in the power management module, Chapter 9, "Power Management." Table 22-1 shows the PIT module operation in low-power modes and how it can exit from each mode.

NOTE

The low-power interrupt control register (LPICR) in the system control module specifies the interrupt level at or above which the device can be brought out of a low-power mode.

Table 22-1. PIT Module Operation in Low-power Modes

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSR*n*[DOZE] cleared, stopped otherwise	Any interrupt at or above level in LPICR, exit doze mode if PCSR*n*[DOZE] is set. Otherwise interrupt assertion has no effect.
Stop	Stopped	No
Debug	Normal if PCSR*n*[DBG] cleared, stopped otherwise	No. Any interrupt is serviced upon normal exit from debug mode

In wait mode, the PIT module continues to operate as in run mode and can be configured to exit the low-power mode by generating an interrupt request. In doze mode with the PCSR*n*[DOZE] bit set, PIT module operation stops. In doze mode with the PCSR*n*[DOZE] bit cleared, doze mode does not affect PIT operation. When doze mode is exited, PIT continues operating in the state it was in prior to doze mode. In stop mode, the internal bus clock is absent and PIT module operation stops.

In debug mode with the PCSR*n*[DBG] bit set, PIT module operation stops. In debug mode with the PCSR*n*[DBG] bit cleared, debug mode does not affect PIT operation. When debug mode is exited, the PIT continues to operate in its pre-debug mode state, but any updates made in debug mode remain.

22.2 Memory Map/Register Definition

This section contains a memory map (see Table 22-2) and describes the register structure for PIT0–PIT1.

Table 22-2. Programmable Interrupt Timer Modules Memory Map

IPSBAR Offset PIT 0 PIT 1	Register	Width (bits)	Access[1]	Reset Value	Section/Page
Supervisor Access Only Registers[2]					
0x15_0000 0x16_0000	PIT Control and Status Register (PCSR*n*)	16	R/W	0x0000	22.2.1/22-3
0x15_0002 0x16_0002	PIT Modulus Register (PMR*n*)	16	R/W	0xFFFF	22.2.2/22-4
User/Supervisor Access Registers					
0x15_0004 0x16_0004	PIT Count Register (PCNTR*n*)	16	R	0xFFFF	22.2.3/22-5

1 Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

[2] User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

22.2.1 PIT Control and Status Register (PCSR*n*)

The PCSR*n* registers configure the corresponding timer's operation.

IPSBAR 0x15_0000 (PCSR0)
Offset: 0x16_0000 (PCSR1)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PRE				0	DOZE	DBG	OVW	PIE	PIF	RLD	EN
W														w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-2. PCSR*n* Register

Table 22-3. PCSR*n* Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 PRE	Prescaler. The read/write prescaler bits select the internal bus clock divisor to generate the PIT clock. To accurately predict the timing of the next count, change the PRE[3:0] bits only when the enable bit (EN) is clear. Changing PRE[3:0] resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Setting the EN bit and writing to PRE[3:0] can be done in this same write cycle. Clearing the EN bit stops the prescaler counter.
7	Reserved, must be cleared.
6 DOZE	Doze Mode Bit. The read/write DOZE bit controls the function of the PIT in doze mode. Reset clears DOZE. 0 PIT function not affected in doze mode 1 PIT function stopped in doze mode. When doze mode is exited, timer operation continues from the state it was in before entering doze mode.

PRE	Internal Bus Clock Divisor	Decimal Equivalent
0000	2^0	1
0001	2^1	2
0010	2^2	4
...	...	...
1101	2^{13}	8192
1110	2^{14}	16384
1111	2^{15}	32768

Table 22-3. PCSR*n* Field Descriptions (continued)

Field	Description
5 DBG	Debug mode bit. Controls the function of PIT in halted/debug mode. Reset clears DBG. During debug mode, register read and write accesses function normally. When debug mode is exited, timer operation continues from the state it was in before entering debug mode, but any updates made in debug mode remain. 0 PIT function not affected in debug mode 1 PIT function stopped in debug mode **Note:** Changing the DBG bit from 1 to 0 during debug mode starts the PIT timer. Likewise, changing the DBG bit from 0 to 1 during debug mode stops the PIT timer.
4 OVW	Overwrite. Enables writing to PMR*n* to immediately overwrite the value in the PIT counter. 0 Value in PMR*n* replaces value in PIT counter when count reaches 0x0000. 1 Writing PMR*n* immediately replaces value in PIT counter.
3 PIE	PIT interrupt enable. This read/write bit enables PIF flag to generate interrupt requests. 0 PIF interrupt requests disabled 1 PIF interrupt requests enabled
2 PIF	PIT interrupt flag. This read/write bit is set when PIT counter reaches 0x0000. Clear PIF by writing a 1 to it or by writing to PMR. Writing 0 has no effect. Reset clears PIF. 0 PIT count has not reached 0x0000. 1 PIT count has reached 0x0000.
1 RLD	Reload bit. The read/write reload bit enables loading the value of PMR*n* into PIT counter when the count reaches 0x0000. 0 Counter rolls over to 0xFFFF on count of 0x0000 1 Counter reloaded from PMR*n* on count of 0x0000
0 EN	PIT enable bit. Enables PIT operation. When PIT is disabled, counter and prescaler are held in a stopped state. This bit is read anytime, write anytime. 0 PIT disabled 1 PIT enabled

22.2.2 PIT Modulus Register (PMR*n*)

The 16-bit read/write PMR*n* contains the timer modulus value loaded into the PIT counter when the count reaches 0x0000 and the PCSR*n*[RLD] bit is set.

When the PCSR*n*[OVW] bit is set, PMR*n* is transparent, and the value written to PMR*n* is immediately loaded into the PIT counter. The prescaler counter is reset (0xFFFF) anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR*n* returns the value written in the modulus latch. Reset initializes PMR*n* to 0xFFFF.

IPSBAR 0x15_0002 (PMR0)
Offset: 0x16_0002 (PMR1)

Access: Supervisor read/write

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

Figure 22-3. PIT Modulus Register (PMR*n*)

Table 22-4. PMR*n* Field Descriptions

Field	Description
15–0 PM	Timer modulus. The value of this register is loaded into the PIT counter when the count reaches zero and the PCSR*n*[RLD] bit is set. However, if PCSR*n*[OVW] is set, the value written to this field is immediately loaded into the counter. Reading this field returns the value written.

22.2.3 PIT Count Register (PCNTR*n*)

The 16-bit, read-only PCNTR*n* contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed coherent. Writing to PCNTR*n* has no effect, and write cycles are terminated normally.

IPSBAR 0x15_0004 (PCNTR0)
Offset: 0x16_0004 (PCNTR1)

Access: User read only

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

Figure 22-4. PIT Count Register (PCNTR*n*)

Table 22-5. PCNTR*n* Field Descriptions

Field	Description
15–0 PC	Counter value. Reading this field with two 8-bit reads is not guaranteed coherent. Writing to PCNTR*n* has no effect, and write cycles are terminated normally.

22.3 Functional Description

This section describes the PIT functional operation.

22.3.1 Set-and-Forget Timer Operation

This mode of operation is selected when the RLD bit in the PCSR register is set.

When PIT counter reaches a count of 0x0000, PIF flag is set in PCSR*n*. The value in the modulus register loads into the counter, and the counter begins decrementing toward 0x0000. If the PCSR*n*[PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR*n*[OVW] bit is set, the counter can be directly initialized by writing to PMR*n* without having to wait for the count to reach 0x0000.

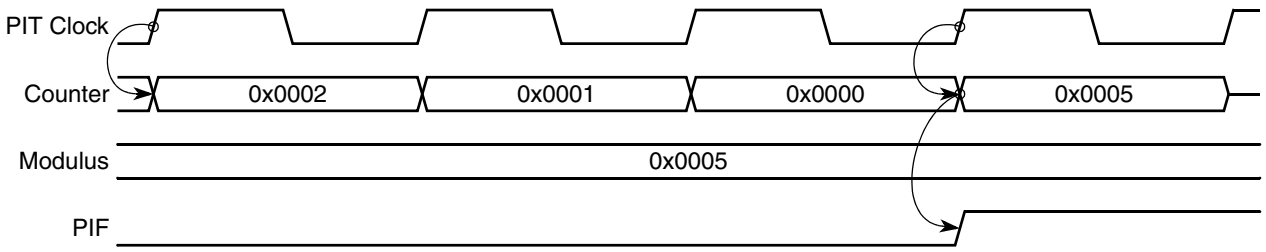


Figure 22-5. Counter Reloading from the Modulus Latch

22.3.2 Free-Running Timer Operation

This mode of operation is selected when the PCSR*n*[RLD] bit is clear. In this mode, the counter rolls over from 0x0000 to 0xFFFF without reloading from the modulus latch and continues to decrement.

When the counter reaches a count of 0x0000, PCSR*n*[PIF] flag is set. If the PCSR*n*[PIE] bit is set, PIF flag issues an interrupt request to the CPU.

When the PCSR*n*[OVW] bit is set, counter can be directly initialized by writing to PMR*n* without having to wait for the count to reach 0x0000.

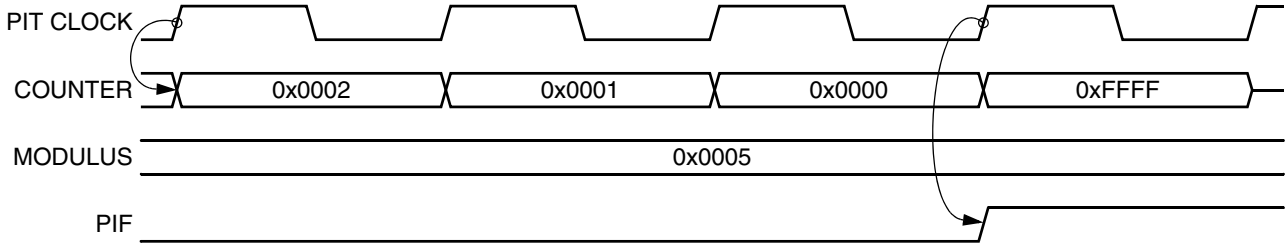


Figure 22-6. Counter in Free-Running Mode

22.3.3 Timeout Specifications

The 16-bit PIT counter and prescaler supports different timeout periods. The prescaler divides the internal bus clock period as selected by the PCSR*n*[PRE] bits. The PMR*n*[PM] bits select the timeout period.

$$\text{Timeout period} = \frac{2^{\text{PCSRn[PRE]}} \times (\text{PMRn[PM]} + 1)}{f_{sys/2}} \qquad \textit{Eqn. 22-1}$$

22.3.4 Interrupt Operation

Table 22-6 shows the interrupt request generated by the PIT.

Table 22-6. PIT Interrupt Requests

Interrupt Request	Flag	Enable Bit
Timeout	PIF	PIE

The PIF flag is set when the PIT counter reaches 0x0000. The PIE bit enables the PIF flag to generate interrupt requests. Clear PIF by writing a 1 to it or by writing to the PMR.

Chapter 23
General Purpose Timer Module (GPT)

23.1 Introduction

This device has one 4-channel general purpose timer module (GPT). It consists of a 16-bit counter driven by a 7-stage programmable prescaler.

A timer overflow function allows software to extend the timing capability of the system beyond the 16-bit range of the counter. Each of the four timer channels can be configured for input capture, which can capture the time of a selected transition edge, or for output compare, which can generate output waveforms and timer software delays. These functions allow simultaneous input waveform measurements and output waveform generation.

Additionally, channel 3 can be configured as a 16-bit pulse accumulator that can operate as a simple event counter or as a gated time accumulator. The pulse accumulator uses the GPT channel 3 input/output pin in event mode or gated time accumulation mode.

23.2 Features

Features of the general-purpose timer include the following:

- Four 16-bit input capture/output compare channels
- 16-bit architecture
- Programmable prescaler
- Pulse widths variable from microseconds to seconds
- Single 16-bit pulse accumulator
- Toggle-on-overflow feature for pulse-width modulator (PWM) generation

23.3 Block Diagram

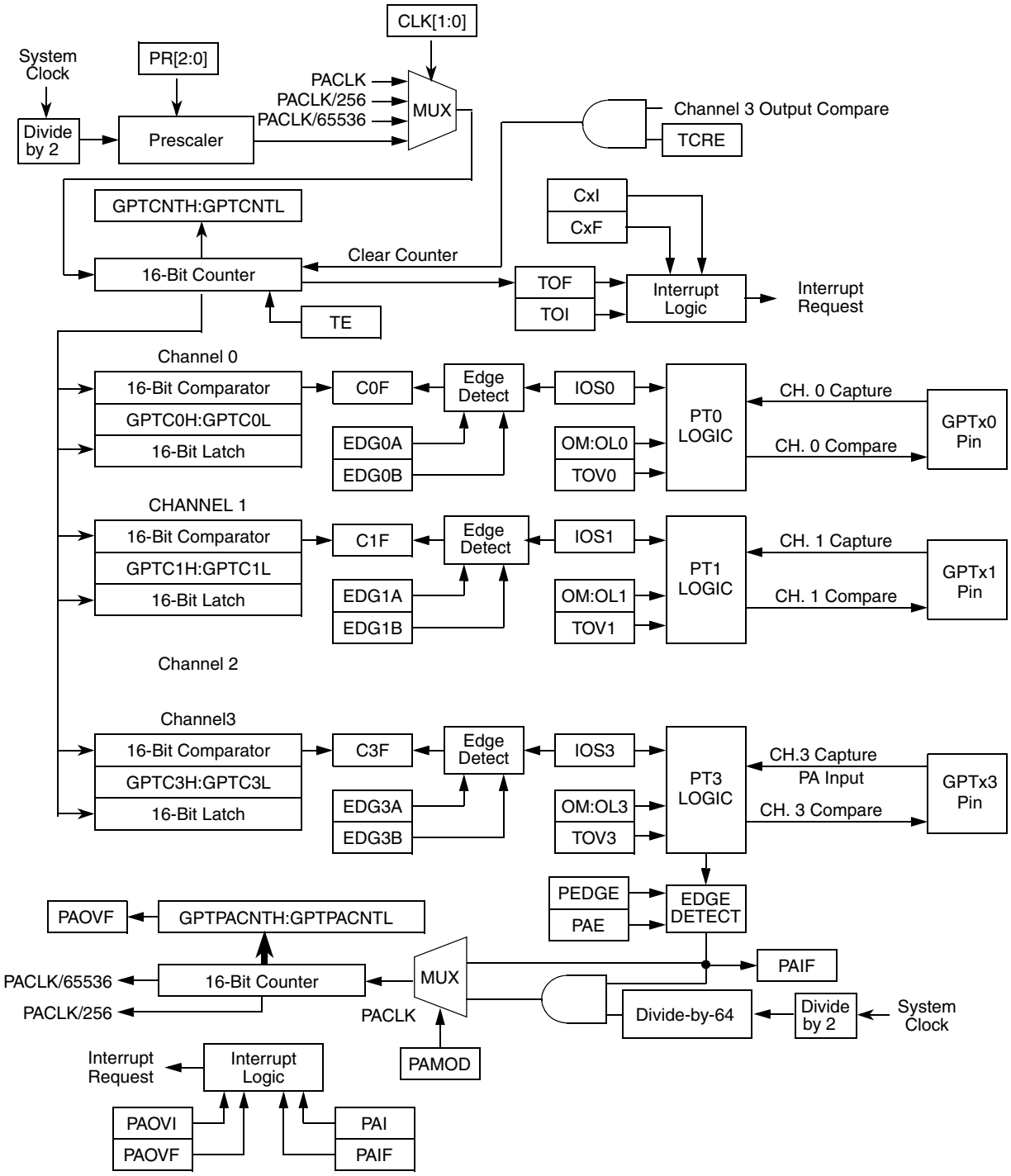


Figure 23-1. GPT Block Diagram

23.4 Low-Power Mode Operation

This subsection describes the operation of the general purpose time module in low-power modes and halted mode of operation. Low-power modes are described in Chapter 9, "Power Management." Table 23-1 shows the general purpose timer module operation in the low-power modes, and shows how this module may facilitate exit from each mode.

Table 23-1. Watchdog Module Operation in Low-power Modes

Low-power Mode	Watchdog Operation	Mode Exit
Wait	Normal	No
Doze	Normal	No
Stop	Stopped	No
Halted	Normal	No

General purpose timer operation stops in stop mode. When stop mode is exited, the general purpose timer continues to operate in its pre-stop mode state.

23.5 Signal Description

Table 23-2 provides an overview of the signal properties.

Table 23-2. Signal Properties

Pin Name	GPTPORT Register Bit	Function	Reset State	Pull-up
GPT0	PORTT*n*0	GPT channel 0 IC/OC pin	Input	Active
GPT1	PORTT*n*1	GPT channel 1 IC/OC pin	Input	Active
GPT2	PORTT*n*2	GPT channel 2 IC/OC pin	Input	Active
GPT3	PORTT*n*3	GPT channel 3 IC/OC or PA pin	Input	Active

23.5.1 GPT[2:0]

The GPT[2:0] pins are for channel 2–0 input capture and output compare functions. These pins are available for general-purpose input/output (I/O) when not configured for timer functions.

23.5.2 GPT3

The GPT3 pin is for channel 3 input capture and output compare functions or for the pulse accumulator input. This pin is available for general-purpose I/O when not configured for timer functions.

23.6 Memory Map and Registers

Table 23-3 shows the memory map of the GPT module. The base address for GPT is IPSBAR + 0x1A_0000.

NOTE

Reading reserved or unimplemented locations returns zeros. Writing to reserved or unimplemented locations has no effect.

Table 23-3. QSPI Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Mode Access Only					
0x1A_0000	GPT IC/OC Select Register (GPTIOS)	8	R/W	0x00	23.6.1/23-5
0x1A_0001	GPT Compare Force Register (GPTCFORC)	8	R/W	0x00	23.6.2/23-5
0x1A_0002	GPT Output Compare 3 Mask Register (GPTOC3M)	8	R/W	0x00	23.6.3/23-6
0x1A_0003	GPT Output Compare 3 Data Register (GPTOC3D)	8	R/W	0x00	23.6.4/23-7
0x1A_0004	GPT Counter Register High (GPTCNTH)[2]	8	R	0x00	23.6.5/23-7
0x1A_0005	GPT Counter Register Low (GPTCNTL)[2]	8	R	0x00	23.6.5/23-7
0x1A_0006	GPT System Control Register 1 (GPTSCR1)	8	R/W	0x00	23.6.6/23-8
0x1A_0008	GPT Toggle-on-Overflow Register (GPTTOV)	8	R/W	0x00	23.6.7/23-9
0x1A_0009	GPT Control Register 1 (GPTCTL1)	8	R/W	0x00	23.6.8/23-9
0x1A_000B	GPT Control Register 2 (GPTCTL2)	8	R/W	0x00	23.6.9/23-10
0x1A_000C	GPT Interrupt Enable Register (GPTIE)	8	R/W	0x00	23.6.10/23-10
0x1A_000D	GPT System Control Register 2 (GPTSCR2)	8	R/W	0x00	23.6.11/23-11
0x1A_000E	GPT Flag Register 1 (GPTFLG1)	8	R/W	0x00	23.6.12/23-12
0x1A_000F	GPT Flag Register 2 (GPTFLG2)	8	R/W	0x00	23.6.13/23-12
0x1A_0010	GPT Channel 0 Register High (GPTC0H)[2]	8			23.6.14/23-13
0x1A_0011	GPT Channel 0 Register Low (GPTC0L)[2]	8			23.6.14/23-13
0x1A_0012	GPT Channel 1 Register High (GPTC1H)[2]	8			23.6.14/23-13
0x1A_0013	GPT Channel 1 Register Low (GPTC1L)[2]	8			23.6.14/23-13
0x1A_0014	GPT Channel 2 Register High (GPTC2H)[2]	8			23.6.14/23-13
0x1A_0015	GPT Channel 2 Register Low (GPTC2L)[2]	8			23.6.14/23-13
0x1A_0016	GPT Channel 3 Register High (GPTC3H)[2]	8			23.6.14/23-13
0x1A_0017	GPT Channel 3 Register Low (GPTC3L)[2]	8			23.6.14/23-13
0x1A_0018	Pulse Accumulator Control Register (GPTPACTL)	8	R/W	0x00	23.6.15/23-13
0x1A_0019	Pulse Accumulator Flag Register (GPTPAFLG)	8	R/W	0x00	23.6.16/23-14
0x1A_001A	Pulse Accumulator Counter Register High (GPTPACNTH)[2]	8	R/W		23.6.17/23-15
0x1A_001B	Pulse Accumulator Counter Register Low (GPTPACNTL)[2]	8	R/W		23.6.17/23-15
0x1A_001D	GPT Port Data Register (GPTPORT)	8	R/W	0x00	23.6.18/23-16

Table 23-3. QSPI Memory Map (continued)

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x1A_001E	GPT Port Data Direction Register (GPTDDR)	8	R/W	0x00	23.6.19/23-16
0x1A_001F	GPT Test Register (GPTTST)	8			

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 This register is 16 bits wide, and should be read using only word accesses.

23.6.1 GPT Input Capture/Output Compare Select Register (GPTIOS)

IPSBAR Access: Supervisor read/write

Offset: 0x1A_0000 (GPTIOS)

	7	6	5	4	3	2	1	0
R	0	0	0	0	IOS			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-2. GPT Input Capture/Output Compare Select Register (GPTIOS)

Table 23-4. GPTIOS Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 IOS	I/O select. The IOS[3:0] bits enable input capture or output compare operation for the corresponding timer channels. These bits are read anytime (always read 0x00), write anytime. 1 Output compare enabled 0 Input capture enabled

23.6.2 GPT Compare Force Register (GPCFORC)

IPSBAR Access: Supervisor read/write

Offset: 0x1A_0001 (GPCFORC)

	7	6	5	4	3	2	1	0
R	0	0	0	0	FOC			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-3. GPT Input Compare Force Register (GPCFORC)

Table 23-5. GPTCFORC Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 FOC	Force output compare.Setting an FOC bit causes an immediate output compare on the corresponding channel. Forcing an output compare does not set the output compare flag. These bits are read anytime, write anytime. 1 Force output compare 0 No effect

NOTE

A successful channel 3 output compare overrides any compare on channels 2:0. For each OC3M bit that is set, the output compare action reflects the corresponding OC3D bit.

23.6.3 GPT Output Compare 3 Mask Register (GPTOC3M)

IPSBAR Access: Supervisor read/write

Offset: 0x1A_0002 (GPTOC3M)

	7	6	5	4	3	2	1	0
R	0	0	0	0	OC3M			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-4. GPT Output Compare 3 Mask Register (GPTOC3M)

Table 23-6. GPTOC3M Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 OC3M	Output compare 3 mask. Setting an OC3M bit configures the corresponding PORTT*n* pin to be an output. OC3M*n* makes the GPT port pin an output regardless of the data direction bit when the pin is configured for output compare (IOSx = 1). The OC3M*n* bits do not change the state of the PORTT*n*DDR bits. These bits are read anytime, write anytime. 1 Corresponding PORTT*n* pin configured as output 0 No effect

23.6.4 GPT Output Compare 3 Data Register (GPTOC3D)

IPSBAR
Offset: 0x1A_0003 (GPTOC3D)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	OC3D			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-5. GPT Output Compare 3 Data Register (GPTOC3D)

Table 23-7. GPTOC3D Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 OC3D	Output compare 3 data. When a successful channel 3 output compare occurs, these bits transfer to the PORTT*n* data register if the corresponding OC3M*n* bits are set. These bits are read anytime, write anytime.

NOTE

A successful channel 3 output compare overrides any channel 2:0 compares. For each OC3M bit that is set, the output compare action reflects the corresponding OC3D bit.

23.6.5 GPT Counter Register (GPTCNT)

IPSBAR
Offset: 0x1A_0004 (GPTCNT)

Access: Supervisor read-only

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

Figure 23-6. GPT Counter Register (GPTCNT)

Table 23-8. GPTCNT Field Descriptions

Field	Description
15–0 CNTR	Read-only field that provides the current count of the timer counter. To ensure coherent reading of the timer counter, such that a timer rollover does not occur between two back-to-back 8-bit reads, it is recommended that only word (16-bit) accesses be used. A write to GPTCNT may have an extra cycle on the first count because the write is not synchronized with the prescaler clock. The write occurs at least one cycle before the synchronization of the prescaler clock. These bits are read anytime. They should be written to only in test (special) mode; writing to them has no effect in normal modes.

23.6.6 GPT System Control Register 1 (GPTSCR1)

IPSBAR
Offset: 0x1A_0006 (GPTSCR1)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	GPTEN	0		TFFCA	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-7. GPT System Control Register 1 (GPTSCR1)

Table 23-9. GPTSCR1 Field Descriptions

Field	Description
7 GPTEN	Enables the general purpose timer. When the timer is disabled, only the registers are accessible. Clearing GPTEN reduces power consumption. These bits are read anytime, write anytime. 1 GPT enabled 0 GPT and GPT counter disabled
6–5	Reserved, should be cleared.
4 TFFCA	Timer fast flag clear all. Enables fast clearing of the main timer interrupt flag registers (GPTFLG1 and GPTFLG2) and the PA flag register (GPTPAFLG). TFFCA eliminates the software overhead of a separate clear sequence. See Figure 23-8. When TFFCA is set: • An input capture read or a write to an output compare channel clears the corresponding channel flag, CxF. • Any access of the GPT count registers (GPTCNTH/L) clears the TOF flag. • Any access of the PA counter registers (GPTPACNT) clears the PAOVF and PAIF flags in GPTPAFLG. Writing logic 1s to the flags clears them only when TFFCA is clear. 1 Fast flag clearing 0 Normal flag clearing
3–0	Reserved, should be cleared.

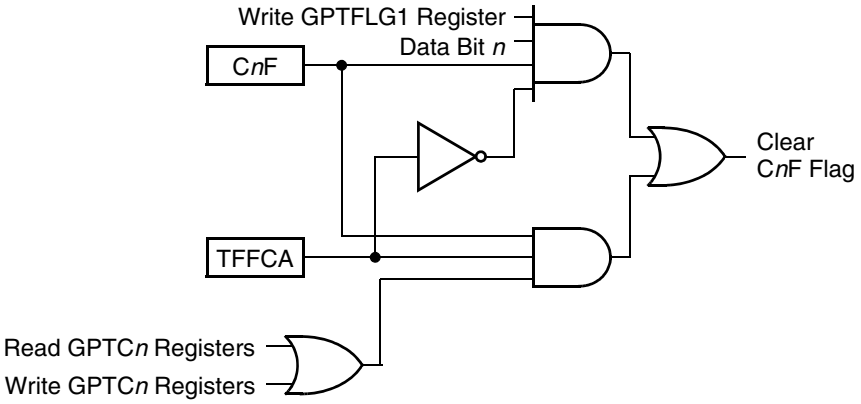


Figure 23-8. Fast Clear Flag Logic

23.6.7 GPT Toggle-On-Overflow Register (GPTTOV)

IPSBAR Access: Supervisor read/write
Offset: 0x1A_0008 (GPTTOV)

	7	6	5	4	3	2	1	0
R	0	0	0	0	TOV			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-9. GPT Toggle-On-Overflow Register (GPTTOV)

Table 23-10. GPTTOV Field Description

Field	Description
7–4	Reserved, should be cleared.
3–0 TOV	Toggles the output compare pin on overflow for each channel. This feature only takes effect when in output compare mode. When set, it takes precedence over forced output compare but not channel 3 override events. These bits are read anytime, write anytime. 1 Toggle output compare pin on overflow feature enabled 0 Toggle output compare pin on overflow feature disabled

23.6.8 GPT Control Register 1 (GPTCTL1)

IPSBAR Access: Supervisor read/write
Offset: 0x1A_0009 (GPTCTL1)

	7	6	5	4	3	2	1	0
R W	OM3	OL3	OM2	OL2	OM1	OL1	OM0	OL0
Reset:	0	0	0	0	0	0	0	0

Figure 23-10. GPT Control Register 1 (GPTCTL1)

Table 23-11. GPTCL1 Field Descriptions

Field	Description
7–0 OMx/OLx	Output mode/output level. Selects the output action to be taken as a result of a successful output compare on each channel. When OM*n* or OL*n* is set and the IOS*n* bit is set, the pin is an output regardless of the state of the corresponding DDR bit. These bits are read anytime, write anytime. 00 GPT disconnected from output pin logic 01 Toggle OC*n* output line 10 Clear OC*n* output line 11 Set OC*n* line **Note:** Channel 3 shares a pin with the pulse accumulator input pin. To use the PAI input, clear the OM3 and OL3 bits and clear the OC3M3 bit in the output compare 3 mask register.

23.6.9 GPT Control Register 2 (GPTCTL2)

IPSBAR Access: Supervisor read/write
Offset: 0x1A_000B (GPTCTL2)

	7	6	5	4	3	2	1	0
R W	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	EDG0B	EDG0A
Reset:	0	0	0	0	0	0	0	0

Figure 23-11. GPT Control Register 2(GPTCTL2)

Table 23-12. GPTLCTL2 Field Descriptions

Field	Description
7–0 EDGn[B:A]	Input capture edge control. Configures the input capture edge detector circuits for each channel. These bits are read anytime, write anytime. 00 Input capture disabled 01 Input capture on rising edges only 10 Input capture on falling edges only 11 Input capture on any edge (rising or falling)

23.6.10 GPT Interrupt Enable Register (GPTIE)

IPSBAR Access: Supervisor read/write
Offset: 0x1A_000C (GPTIE)

	7	6	5	4	3	2	1	0
R	0	0	0	0	CI			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-12. GPT Interrupt Enable Register (GPTIE)

Table 23-13. GPTIE Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 C*n*I	Channel interrupt enable. Enables the C[3:0]F flags in GPT flag register 1 to generate interrupt requests for each channel. These bits are read anytime, write anytime. 1 Corresponding channel interrupt requests enabled 0 Corresponding channel interrupt requests disabled

23.6.11 GPT System Control Register 2 (GPTSCR2)

IPSBAR
Offset: 0x1A_000D (GPTSCR2)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	TOI	0	PUPT	RDPT	TCRE	PR		
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-13. GPT System Control Register 2 (GPTSCR2)

Table 23-14. GPTSCR2 Field Descriptions

Field	Description
7 TOI	Enables timer overflow interrupt requests. 1 Overflow interrupt requests enabled 0 Overflow interrupt requests disabled
6	Reserved, should be cleared.
5 PUPT	Enables pull-up resistors on the GPT ports when the ports are configured as inputs. 1 Pull-up resistors enabled 0 Pull-up resistors disabled
4 RDPT	GPT drive reduction. Reduces the output driver size. 1 Output drive reduction enabled 0 Output drive reduction disabled
3 TCRE	Enables a counter reset after a channel 3 compare. 1 Counter reset enabled 0 Counter reset disabled **Note:** When the GPT channel 3 registers contain 0x0000 and TCRE is set, the GPT counter registers remain at 0x0000 all the time. When the GPT channel 3 registers contain 0xFFFF and TCRE is set, TOF does not get set even though the GPT counter registers go from 0xFFFF to 0x0000.
2–0 PR	Prescaler bits. Select the prescaler divisor for the GPT counter. 000 Prescaler divisor 1 001 Prescaler divisor 2 010 Prescaler divisor 4 011 Prescaler divisor 8 100 Prescaler divisor 16 101 Prescaler divisor 32 110 Prescaler divisor 64 111 Prescaler divisor 128 **Note:** The newly selected prescaled clock does not take effect until the next synchronized edge of the prescaled clock when the clock count transitions to 0x0000.)

23.6.12 GPT Flag Register 1 (GPTFLG1)

IPSBAR
Offset: 0x1A_000E (GPTFLG1)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	CF			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-14. GPT Flag Register 1 (GPTFLG1)

Table 23-15. GPTFLG1 Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 C*n*F	Channel flags. A channel flag is set when an input capture or output compare event occurs. These bits are read anytime, write anytime (writing 1 clears the flag, writing 0 has no effect). **Note:** When the fast flag clear all bit, GPTSCR1[TFFCA], is set, an input capture read or an output compare write clears the corresponding channel flag. When a channel flag is set, it does not inhibit subsequent output compares or input captures.

23.6.13 GPT Flag Register 2 (GPTFLG2)

IPSBAR
Offset: 0x1A_000F (GPTFLG2)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	TOF	0	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-15. GPT Flag Register 2 (GPTFLG2)

Table 23-16. GPTFLG2 Field Descriptions

Field	Description
7 TOF	Timer overflow flag. Set when the GPT counter rolls over from 0xFFFF to 0x0000. If the TOI bit in GPTSCR2 is also set, TOF generates an interrupt request. This bit is read anytime, write anytime (writing 1 clears the flag, and writing 0 has no effect). 1 Timer overflow 0 No timer overflow **Note:** When the GPT channel 3 registers contain 0xFFFF and TCRE is set, TOF does not get set even though the GPT counter registers go from 0xFFFF to 0x0000. When TOF is set, it does not inhibit subsequent overflow events.
6–0	Reserved, should be cleared.

Note: When the fast flag clear all bit, GPTSCR1[TFFCA], is set, any access to the GPT counter registers clears GPT flag register 2.

23.6.14 GPT Channel Registers (GPTC*n*)

IPSBAR Offsets: 0x1A_0010, 0x1A_0012, 0x1A_0014, 0x1A_0016, (GPTC*n*)

Access: Supervisor read/write

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

Figure 23-16. GPT Channel[0:3] Register (GPTC*n*)

Table 23-17. GPTC*n* Field Descriptions

Field	Description
15–0 CCNT	When a channel is configured for input capture (IOS*n* = 0), the GPT channel registers latch the value of the free-running counter when a defined transition occurs on the corresponding input capture pin. When a channel is configured for output compare (IOS*n* = 1), the GPT channel registers contain the output compare value. To ensure coherent reading of the GPT counter, such that a timer rollover does not occur between back-to-back 8-bit reads, it is recommended that only word (16-bit) accesses be used. These bits are read anytime, write anytime (for the output compare channel); writing to the input capture channel has no effect.

23.6.15 Pulse Accumulator Control Register (GPTPACTL)

IPSBAR Offsets: 0x1A_0018 (GPTPACTL)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	PAE	PAMOD	PEDGE	CLK		PAOVI	PAI
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-17. Pulse Accumulator Control Register (GPTPACTL)

Table 23-18. GPTPACTL Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 PAE	Enables the pulse accumulator. 1 Pulse accumulator enabled 0 Pulse accumulator disabled **Note:** The pulse accumulator can operate in event mode even when the GPT enable bit, GPTEN, is clear.
5 PAMOD	Pulse accumulator mode. Selects event counter mode or gated time accumulation mode. 1 Gated time accumulation mode 0 Event counter mode

Table 23-18. GPTPACTL Field Descriptions (continued)

Field	Description
4 PEDGE	Pulse accumulator edge. Selects falling or rising edges on the PAI pin to increment the counter. In event counter mode (PAMOD = 0): 1 Rising PAI edge increments counter 0 Falling PAI edge increments counter In gated time accumulation mode (PAMOD = 1): 1 Low PAI input enables divide-by-64 clock to pulse accumulator and trailing rising edge on PAI sets PAIF flag. 0 High PAI input enables divide-by-64 clock to pulse accumulator and trailing falling edge on PAI sets PAIF flag. **Note:** The timer prescaler generates the divide-by-64 clock. If the timer is not active, there is no divide-by-64 clock. To operate in gated time accumulation mode: 1. Apply logic 0 to $\overline{\text{RSTI}}$ pin. 2. Initialize registers for pulse accumulator mode test. 3. Apply appropriate level to PAI pin. 4. Enable GPT.
3–2 CLK	Select the GPT counter input clock. Changing the CLK bits causes an immediate change in the GPT counter clock input. 00 GPT prescaler clock (When PAE = 0, the GPT prescaler clock is always the GPT counter clock.) 01 PACLK 10 PACLK/256 11 PACLK/65536
1 PAOVI	Pulse accumulator overflow interrupt enable. Enables the PAOVF flag to generate interrupt requests. 1 PAOVF interrupt requests enabled 0 PAOVF interrupt requests disabled
0 PAI	Pulse accumulator input interrupt enable. Enables the PAIF flag to generate interrupt requests. 1 PAIF interrupt requests enabled 0 PAIF interrupt requests disabled

23.6.16 Pulse Accumulator Flag Register (GPTPAFLG)

IPSBAR Access: Supervisor read/write

Offset: 0x1A_0019 (GPTPAFLG)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PAOVF	PAIF
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-18. Pulse Accumulator Flag Register (GPTPAFLG)

Table 23-19. GPTPAFLG Field Descriptions

Field	Description
7–2	Reserved, should be cleared.

Table 23-19. GPTPAFLG Field Descriptions (continued)

Field	Description
1 PAOVF	Pulse accumulator overflow flag. Set when the 16-bit pulse accumulator rolls over from 0xFFFF to 0x0000. If the GPTPACTL[PAOVI] bit is also set, PAOVF generates an interrupt request. Clear PAOVF by writing a 1 to it. This bit is read anytime, write anytime. (Writing 1 clears the flag; writing 0 has no effect.) 1 Pulse accumulator overflow 0 No pulse accumulator overflow
0 PAIF	Pulse accumulator input flag. Set when the selected edge is detected at the PAI pin. In event counter mode, the event edge sets PAIF. In gated time accumulation mode, the trailing edge of the gate signal at the PAI pin sets PAIF. If the PAI bit in GPTPACTL is also set, PAIF generates an interrupt request. Clear PAIF by writing a 1 to it. 1 Active PAI input 0 No active PAI input

NOTE

When the fast flag clear all enable bit (GPTSCR1[TFFCA]) is set, any access to the pulse accumulator counter registers clears all the flags in GPTPAFLG.

23.6.17 Pulse Accumulator Counter Register (GPTPACNT)

IPSBAR Access: Supervisor read/write
Offset: 0x1A_001A (GPTPACNT)

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

Figure 23-19. Pulse Accumulator Counter Register (GPTPACNT)

Table 23-20. GPTPACR Field Descriptions

Field	Description
15–0 PACNT	Contains the number of active input edges on the PAI pin since the last reset. **Note:** Reading the pulse accumulator counter registers immediately after an active edge on the PAI pin may miss the last count because the input first has to be synchronized with the bus clock. To ensure coherent reading of the PA counter so that the counter does not increment between back-to-back 8-bit reads, it is recommended that only word (16-bit) accesses be used. These bits are read anytime, write anytime.

23.6.18 GPT Port Data Register (GPTPORT)

IPSBAR
Offset: 0x1A_001D (GPTPORT)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PORTT			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 23-20. GPT Port Data Register (GPTPORT)

Table 23-21. GPTPORT Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 PORTT	GPT port input capture/output compare data. Data written to GPTPORT is buffered and drives the pins only when they are configured as general-purpose outputs. Reading an input (DDR bit = 0) reads the pin state; reading an output (DDR bit = 1) reads the latched value. Writing to a pin configured as a GPT output does not change the pin state. These bits are read anytime (read pin state when corresponding PORTT*n* bit is 0, read pin driver state when corresponding GPTDDR bit is 1), write anytime.

23.6.19 GPT Port Data Direction Register (GPTDDR)

	7	6	5	4	3	2	1	0
Field	—				DDRT			
GPT Function	—				IC/OC			
Pulse Accumulator Function	—				PAI	—		
Reset	0000_0000							
R/W	R/W							
Address	IPSBAR + 0x1A_001E							

Figure 23-21. GPT Port Data Direction Register (GPTDDR)

Table 23-22. GPTDDR Field Descriptions

Bit(s)	Name	Description
7–4	—	Reserved, should be cleared.
3–0	DDRT	Control the port logic of PORTT*n*. Reset clears the PORTT*n* data direction register, configuring all GPT port pins as inputs. These bits are read anytime, write anytime. 1 Corresponding pin configured as output 0 Corresponding pin configured as input

23.7 Functional Description

The general purpose timer (GPT) module is a 16-bit, 4-channel timer with input capture and output compare functions and a pulse accumulator.

23.7.1 Prescaler

The prescaler divides the module clock by 1 or 16. The PR[2:0] bits in GPTSCR2 select the prescaler divisor.

23.7.2 Input Capture

Clearing an I/O select bit (IOS*n*) configures channel *n* as an input capture channel. The input capture function captures the time at which an external event occurs. When an active edge occurs on the pin of an input capture channel, the timer transfers the value in the GPT counter into the GPT channel registers (GPTC*n*).

The minimum pulse width for the input capture input is greater than two module clocks.

The input capture function does not force data direction. The GPT port data direction register controls the data direction of an input capture pin. Pin conditions such as rising or falling edges can trigger an input capture only on a pin configured as an input.

An input capture on channel *n* sets the C*n*F flag. The C*n*I bit enables the C*n*F flag to generate interrupt requests.

23.7.3 Output Compare

Setting an I/O select bit (IOS*n*) configures channel *n* as an output compare channel. The output compare function can generate a periodic pulse with a programmable polarity, duration, and frequency. When the GPT counter reaches the value in the channel registers of an output compare channel, the timer can set, clear, or toggle the channel pin. An output compare on channel *n* sets the C*n*F flag. The C*n*I bit enables the C*n*F flag to generate interrupt requests.

The output mode (OM*n)* and level bits (OL*n)* select, set, clear, or toggle on output compare. Clearing OM*n* and OL*n* disconnects the pin from the output logic.

Setting a force output compare bit (FOC*n*) causes an output compare on channel *n*. A forced output compare does not set the channel flag.

A successful output compare on channel 3 overrides output compares on all other output compare channels. A channel 3 output compare can cause bits in the output compare 3 data register to transfer to the GPT port data register, depending on the output compare 3 mask register. The output compare 3 mask register masks the bits in the output compare 3 data register. The GPT counter reset enable bit, TCRE, enables channel 3 output compares to reset the GPT counter. A channel 3 output compare can reset the GPT counter even if the OC3/PAI pin is being used as the pulse accumulator input.

An output compare overrides the data direction bit of the output compare pin but does not change the state of the data direction bit.

Writing to the PORTT*n* bit of an output compare pin does not affect the pin state. The value written is stored in an internal latch. When the pin becomes available for general-purpose output, the last value written to the bit appears at the pin.

23.7.4 Pulse Accumulator

The pulse accumulator (PA) is a 16-bit counter that can operate in two modes:

1. Event counter mode: counts edges of selected polarity on the pulse accumulator input pin, PAI
2. Gated time accumulation mode: counts pulses from a divide-by-64 clock

The PA mode bit (PAMOD) selects the mode of operation.

The minimum pulse width for the PAI input is greater than two module clocks.

23.7.5 Event Counter Mode

Clearing the PAMOD bit configures the PA for event counter operation. An active edge on the PAI pin increments the PA. The PA edge bit (PEDGE) selects falling edges or rising edges to increment the PA.

An active edge on the PAI pin sets the PA input flag (PAIF). The PA input interrupt enable bit (PAI) enables the PAIF flag to generate interrupt requests.

NOTE

The PAI input and GPT channel 3 use the same pin. To use the PAI input, disconnect it from the output logic by clearing the channel 3 output mode and output level bits, OM3 and OL3. Also clear the channel 3 output compare 3 mask bit (OC3M3).

The PA counter register (GPTPACNT) reflects the number of active input edges on the PAI pin since the last reset.

The PA overflow flag (PAOVF) is set when the PA rolls over from 0xFFFF to 0x0000. The PA overflow interrupt enable bit (PAOVI) enables the PAOVF flag to generate interrupt requests.

NOTE

The PA can operate in event counter mode even when the GPT enable bit (GPTEN) is clear.

23.7.6 Gated Time Accumulation Mode

Setting the PAMOD bit configures the PA for gated time accumulation operation. An active level on the PAI pin enables a divide-by-64 clock to drive the PA. The PA edge bit (PEDGE) selects low levels or high levels to enable the divide-by-64 clock.

The trailing edge of the active level at the PAI pin sets the PA input flag (PAIF). The PA input interrupt enable bit (PAI) enables the PAIF flag to generate interrupt requests.

NOTE

The PAI input and GPT channel 3 use the same pin. To use the PAI input, disconnect it from the output logic by clearing the channel 3 output mode (OM3) and output level (OL3) bits. Also clear the channel 3 output compare mask bit (OC3M3).

The PA counter register (GPTPACNT) reflects the number of pulses from the divide-by-64 clock since the last reset.

NOTE

The GPT prescaler generates the divide-by-64 clock. If the timer is not active, there is no divide-by-64 clock.

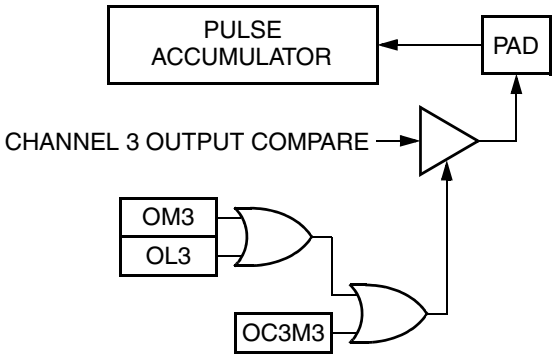


Figure 23-22. Channel 3 Output Compare/Pulse Accumulator Logic

23.7.7 General-Purpose I/O Ports

An I/O pin used by the timer defaults to general-purpose I/O unless an internal function that uses that pin is enabled.

The PORTT*n* pins can be configured for an input capture function or an output compare function. The IOS*n* bits in the GPT IC/OC select register configure the PORTT*n* pins as input capture or output compare pins.

The PORTT*n* data direction register controls the data direction of an input capture pin. External pin conditions trigger input captures on input capture pins configured as inputs.

To configure a pin for input capture:

1. Clear the pin's IOS bit in GPTIOS.
2. Clear the pin's DDR bit in PORTT*n*DDR.
3. Write to GPTCTL2 to select the input edge to detect.

PORTT*n*DDR does not affect the data direction of an output compare pin. The output compare function overrides the data direction register but does not affect the state of the data direction register.

To configure a pin for output compare:

1. Set the pin's IOS bit in GPTIOS.
2. Write the output compare value to GPTC*n*.

3. Clear the pin's DDR bit in PORTT*n*DDR.
4. Write to the OM*n*/OL*n* bits in GPTCTL1 to select the output action.

Table 23-23 shows how various timer settings affect pin functionality.

Table 23-23. GPT Settings and Pin Functions

GPTEN	DDR[1]	GPTIOS	EDGx [B:A]	OMx/ OLx[2]	OC3Mx[3]	Pin Data Dir.	Pin Driven by	Pin Function	Comments
0	0	X[4]	X	X	X	In	Ext.	Digital input	GPT disabled by GPTEN = 0
0	1	X	X	X	X	Out	Data reg.	Digital output	GPT disabled by GPTEN = 0
1	0	0 (IC)	0 (IC disabled)	X	0	In	Ext.	Digital input	Input capture disabled by EDG*n* setting
1	1	0	0	X	0	Out	Data reg.	Digital output	Input capture disabled by EDG*n* setting
1	0	0	<> 0	X	0	In	Ext.	IC and digital input	Normal settings for input capture
1	1	0	<> 0	X	0	Out	Data reg.	Digital output	Input capture of data driven to output pin by CPU
1	0	0	<> 0	X	1	In	Ext.	IC and digital input	OC3M setting has no effect because IOS = 0
1	1	0	<> 0	X	1	Out	Data reg.	Digital output	OC3M setting has no effect because IOS = 0; input capture of data driven to output pin by CPU
1	0	1 (OC)	X(3)	0[5]	0	In	Ext.	Digital input	Output compare takes place but does not affect the pin because of the OM*n*/OL*n* setting
1	1	1	X	0	0	Out	Data reg.	Digital output	Output compare takes place but does not affect the pin because of the OM*n*/OL*n* setting
1	0	1	X	<> 0	0	Out	OC action	Output compare	Pin readable only if DDR = 0(5)
1	1	1	X	<> 0	0	Out	OC action	Output compare	Pin driven by OC action(5)
1	0	1	X	X	1	Out	OC action/ OC3D*n*	Output compare (ch 3)	Pin readable only if DDR = 0[6]
1	1	1	X	X	1	Out	OC action/ OC3D*n*	Output compare/ OC3D*n* (ch 3)	Pin driven by channel OC action and OC3D*n* via channel 3 OC(6)

1 When DDR set the pin as input (0), reading the data register returns the state of the pin. When DDR set the pin as output (1), reading the data register returns the content of the data latch. Pin conditions such as rising or falling edges can trigger an input capture on a pin configured as an input.

2 OM*n*/OL*n* bit pairs select the output action to be taken as a result of a successful output compare. When OM*n* or OL*n* is set and the IOS*n* bit is set, the pin is an output regardless of the state of the corresponding DDR bit.

3 Setting an OC3M bit configures the corresponding PORTT*n* pin to be output. OC3M*n* makes the PORTT*n* pin an output regardless of the data direction bit when the pin is configured for output compare (IOS*n* = 1). The OC3M*n* bits do not change the state of the PORTT*n*DDR bits.

4 X = Don't care

5 An output compare overrides the data direction bit of the output compare pin but does not change the state of the data direction bit. Enabling output compare disables data register drive of the pin.

6 A successful output compare on channel 3 causes an output value determined by OC3D*n* value to temporarily override the output compare pin state of any other output compare channel.The next OC action for the specific channel remains output to the pin. A channel 3 output compare can cause bits in the output compare 3 data register to transfer to the GPT port data register, depending on the output compare 3 mask register.

23.8 Reset

Reset initializes the GPT registers to a known startup state as described in Section 23.6, "Memory Map and Registers."

23.9 Interrupts

Table 23-24 lists the interrupt requests generated by the timer.

Table 23-24. GPT Interrupt Requests

Interrupt Request	Flag	Enable Bit
Channel 3 IC/OC	C3F	C3I
Channel 2 IC/OC	C2F	C2I
Channel 1 IC/OC	C1F	C1I
Channel 0 IC/OC	C0F	C0I
PA overflow	PAOVF	PAOVI
PA input	PAIF	PAI
Timer overflow	TOF	TOI

23.9.1 GPT Channel Interrupts (C*n*F)

A channel flag is set when an input capture or output compare event occurs. Clear a channel flag by writing a 1 to it.

NOTE

When the fast flag clear all bit (GPTSCR1[TFFCA]) is set, an input capture read or an output compare write clears the corresponding channel flag.

When a channel flag is set, it does not inhibit subsequent output compares or input captures

23.9.2 Pulse Accumulator Overflow (PAOVF)

PAOVF is set when the 16-bit pulse accumulator rolls over from 0xFFFF to 0x0000. If the PAOVI bit in GPTPACTL is also set, PAOVF generates an interrupt request. Clear PAOVF by writing a 1 to this flag.

NOTE

When the fast flag clear all enable bit (GPTSCR1[TFFCA]) is set, any access to the pulse accumulator counter registers clears all the flags in GPTPAFLG.

23.9.3 Pulse Accumulator Input (PAIF)

PAIF is set when the selected edge is detected at the PAI pin. In event counter mode, the event edge sets PAIF. In gated time accumulation mode, the trailing edge of the gate signal at the PAI pin sets PAIF. If the PAI bit in GPTPACTL is also set, PAIF generates an interrupt request. Clear PAIF by writing a 1 to this flag.

NOTE

When the fast flag clear all enable bit (GPTSCR1[TFFCA]) is set, any access to the pulse accumulator counter registers clears all the flags in GPTPAFLG.

23.9.4 Timer Overflow (TOF)

TOF is set when the GPT counter rolls over from 0xFFFF to 0x0000. If the GPTSCR2[TOI] bit is also set, TOF generates an interrupt request. Clear TOF by writing a 1 to this flag.

NOTE

When the GPT channel 3 registers contain 0xFFFF and TCRE is set, TOF does not get set even though the GPT counter registers go from 0xFFFF to 0x0000.

When the fast flag clear all bit (GPTSCR1[TFFCA]) is set, any access to the GPT counter registers clears GPT flag register 2.

When TOF is set, it does not inhibit future overflow events.

Chapter 24
DMA Timers (DTIM0–DTIM3)

24.1 Introduction

This chapter describes the configuration and operation of the four direct memory access (DMA) timer modules (DTIM0, DTIM1, DTIM2, and DTIM3). These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or DMA triggers. Additionally, programming examples are included.

NOTE

The designation *n* appears throughout this section to refer to registers or signals associated with one of the four identical timer modules: DTIM0, DTIM1, DTIM2, or DTIM3.

24.1.1 Overview

Each DMA timer module has a separate register set for configuration and control. The timers can be configured to operate from the internal bus clock (f_{sys}) or from an external clocking source using the DTIN*n* signal. If the internal bus clock is selected, it can be divided by 16 or 1. The selected clock source is routed to an 8-bit programmable prescaler that clocks the actual DMA timer counter register (DTCN*n*). Using the DTMR*n*, DTXMR*n*, DTCR*n*, and DTRR*n* registers, the DMA timer may be configured to assert an output signal, generate an interrupt, or request a DMA transfer on a particular event.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “General Purpose I/O Module”) prior to configuring the DMA Timers.

Figure 24-1 is a block diagram of one of the four identical timer modules.

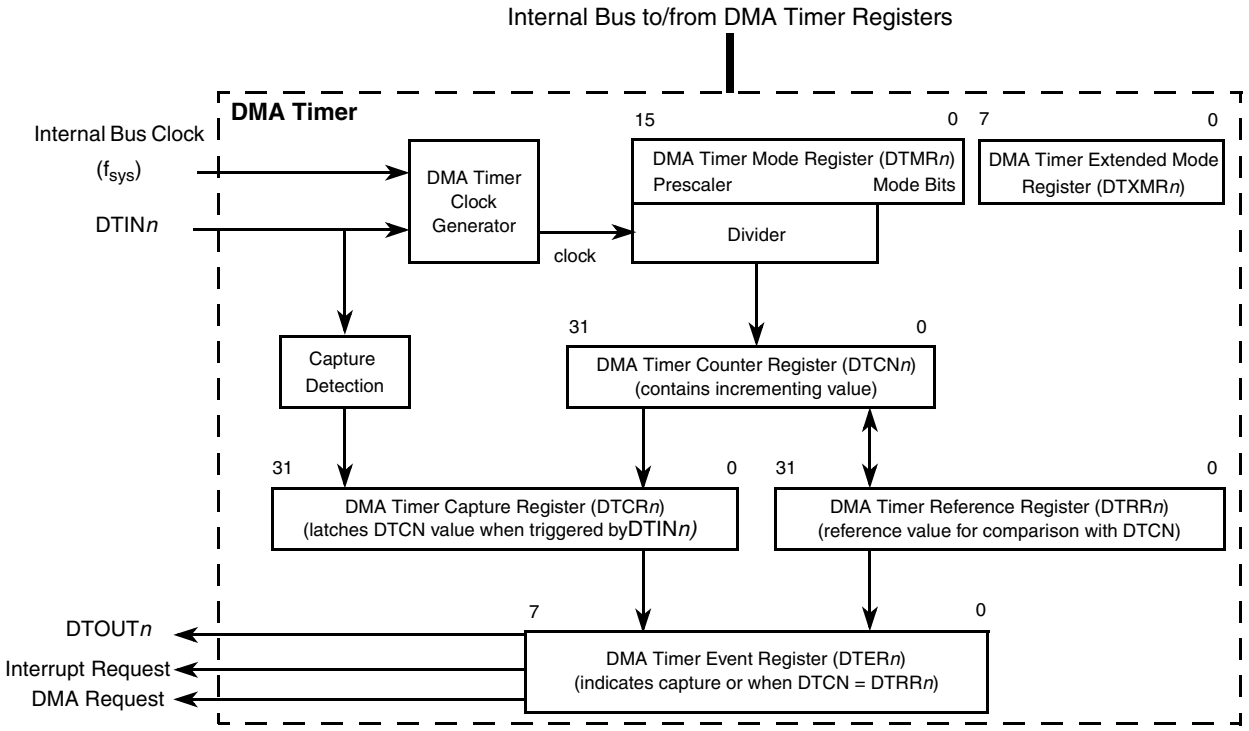


Figure 24-1. DMA Timer Block Diagram

24.1.2 Features

Each DMA timer module has:

- Maximum timeout period of 293,203 seconds at 60 MHz (~81 hours)
- 17-ns resolution at 60 MHz
- Programmable sources for the clock input, including external clock
- Programmable prescaler
- Input-capture capability with programmable trigger edge on input pin
- Programmable mode for the output pin on reference compare
- Free run and restart modes
- Programmable interrupt or DMA request on input capture or reference-compare
- Ability to stop the timer from counting when the ColdFire core is halted

24.2 Memory Map/Register Definition

The timer module registers, shown in Table 24-1, can be modified at any time.

Table 24-1. DMA Timer Module Memory Map

IPSBAR Offset DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3	Register	Width (bits)	Access	Reset Value	Section/Page
0x00_0400 0x00_0440 0x00_0480 0x00_04C0	DMA Timer *n* Mode Register (DTMR*n*)	16	R/W	0x0000	24.2.1/24-3
0x00_0402 0x00_0442 0x00_0482 0x00_04C2	DMA Timer *n* Extended Mode Register (DTXMR*n*)	8	R/W	0x00	24.2.2/24-5
0x00_0403 0x00_0443 0x00_0483 0x00_04C3	DMA Timer *n* Event Register (DTER*n*)	8	R/W	0x00	24.2.3/24-5
0x00_0404 0x00_0444 0x00_0484 0x00_04C4	DMA Timer *n* Reference Register (DTRR*n*)	32	R/W	0xFFFF_FFFF	24.2.4/24-7
0x00_0408 0x00_0448 0x00_0488 0x00_04C8	DMA Timer *n* Capture Register (DTCR*n*)	32	R/W	0x0000_0000	24.2.5/24-7
0x00_040C 0x00_044C 0x00_048C 0x00_04CC	DMA Timer *n* Counter Register (DTCN*n*)	32	R	0x0000_0000	24.2.6/24-8

24.2.1 DMA Timer Mode Registers (DTMR*n*)

The DTMR*n* registers program the prescaler and various timer modes.

IPSBAR 0x00_0400 (DTMR0)
Offset: 0x00_0440 (DTMR1)
0x00_0480 (DTMR2)
0x00_04C0 (DTMR3)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PS								CE		OM	ORRI	FRR	CLK		RST
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-2. DTMR*n* Registers

Table 24-2. DTMR*n* Field Descriptions

Field	Description
15–8 PS	Prescaler value. Divides the clock input (internal bus clock/(16 or 1) or clock on DTIN*n*) 0x00 1 ... 0xFF 256
7–6 CE	Capture edge. 00 Disable capture event output. Timer in reference mode. 01 Capture on rising edge only 10 Capture on falling edge only 11 Capture on any edge
5 OM	Output mode. 0 Active-low pulse for one internal bus clock cycle (17-ns resolution at 60 MHz) 1 Toggle output.
4 ORRI	Output reference request, interrupt enable. If ORRI is set when DTER*n*[REF] is set, a DMA request or an interrupt occurs, depending on the value of DTXMR*n*[DMAEN] (DMA request if set, interrupt if cleared). 0 Disable DMA request or interrupt for reference reached (does not affect DMA request or interrupt on capture function). 1 Enable DMA request or interrupt upon reaching the reference value.
3 FRR	Free run/restart 0 Free run. Timer count continues incrementing after reaching the reference value. 1 Restart. Timer count is reset immediately after reaching the reference value.
2–1 CLK	Input clock source for the timer. Avoid setting CLK when RST is already set. Doing so causes CLK to zero (stop counting). 00 Stop count 01 Internal bus clock divided by 1 10 Internal bus clock divided by 16. This clock source is not synchronized with the timer; therefore, successive time-outs may vary slightly. 11 DTIN*n* pin (falling edge)
0 RST	Reset timer. Performs a software timer reset similar to an external reset, although other register values can be written while RST is cleared. A transition of RST from 1 to 0 resets register values. The timer counter is not clocked unless the timer is enabled. 0 Reset timer (software reset) 1 Enable timer

24.2.2 DMA Timer Extended Mode Registers (DTXMR*n*)

The DTXMR*n* registers program DMA request and increment modes for the timers.

IPSBAR 0x00_0402 (DTXMR0) Access: User read/write
Offset: 0x00_0442 (DTXMR1)
0x00_0482 (DTXMR2)
0x00_04C2 (DTXMR3)

	7	6	5	4	3	2	1	0
R	DMAEN	HALTED	0	0	0	0	0	MODE16
W								
Reset:	0	0	0	0	0	0	0	0

Figure 24-3. DTXMR*n* Registers

Table 24-3. DTXMR*n* Field Descriptions

Field	Description
7 DMAEN	DMA request. Enables DMA request output on counter reference match or capture edge event. 0 DMA request disabled 1 DMA request enabled
6 HALTED	Controls the counter when the core is halted. This allows debug mode to be entered without timer interrupts affecting the debug flow. 0 Timer function is not affected by core halt. 1 Timer stops counting while the core is halted. **Note:** This bit is only applicable in reference compare mode, see Section 24.3.3, “Reference Compare.”
5–1	Reserved, must be cleared.
0 MODE16	Selects the increment mode for the timer. Setting MODE16 is intended to exercise the upper bits of the 32-bit timer in diagnostic software without requiring the timer to count through its entire dynamic range. When set, the counter’s upper 16 bits mirror its lower 16 bits. All 32 bits of the counter remain compared to the reference value. 0 Increment timer by 1 1 Increment timer by 65,537

24.2.3 DMA Timer Event Registers (DTER*n*)

DTER*n*, shown in Figure 24-4, reports capture or reference events by setting DTER*n*[CAP] or DTER*n*[REF]. This reporting happens regardless of the corresponding DMA request or interrupt enable values, DTXMR*n*[DMAEN] and DTMR*n*[ORRI,CE].

Writing a 1 to DTER*n*[REF] or DTER*n*[CAP] clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. If configured to generate an interrupt request, clear REF and CAP early in the interrupt service routine so the timer module can negate the interrupt request signal to the interrupt controller. If configured to generate a DMA request, processing of the DMA data transfer automatically clears the REF and CAP flags via the internal DMA ACK signal.

IPSBAR 0x00_0403 (DTER0)
Offset: 0x00_0443 (DTER1)
0x00_0483 (DTER2)
0x00_04C3 (DTER3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	REF	CAP
W							w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 24-4. DTER*n* Registers

Table 24-4. DTER*n* Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 REF	Output reference event. The counter value (DTCN*n*) equals DTRR*n*. Writing a 1 to REF clears the event condition. Writing a 0 has no effect.
0 CAP	Capture event. The counter value has been latched into DTCR*n*. Writing a 1 to CAP clears the event condition. Writing a 0 has no effect.

REF	DTMR*n*[ORRI]	DTXMR*n*[DMAEN]	
0	X	X	No event
1	0	0	No request asserted
1	0	1	No request asserted
1	1	0	Interrupt request asserted
1	1	1	DMA request asserted

CAP	DTMR*n*[CE]	DTXMR*n* [DMAEN]	
0	XX	X	No event
1	00	0	Disable capture event output
1	00	1	Disable capture event output
1	01	0	Capture on rising edge and trigger interrupt
1	01	1	Capture on rising edge and trigger DMA
1	10	0	Capture on falling edge and trigger interrupt
1	10	1	Capture on falling edge and trigger DMA
1	11	0	Capture on any edge and trigger interrupt
1	11	1	Capture on any edge and trigger DMA

24.2.4 DMA Timer Reference Registers (DTRR*n*)

As part of the output-compare function, each DTRR*n* contains the reference value compared with the respective free-running timer counter (DTCN*n*).

The reference value is matched when DTCN*n* equals DTRR*n*. The prescaler indicates that DTCN*n* should be incremented again. Therefore, the reference register is matched after DTRR*n* + 1 time intervals.

IPSBAR 0x00_0404 (DTRR0) Access: User read/write
Offset: 0x00_0444 (DTRR1)
0x00_0484 (DTRR2)
0x00_04C4 (DTRR3)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	REF (32-bit reference value)																															
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 24-5. DTRR*n* Registers

Table 24-5. DTRR*n* Field Descriptions

Field	Description
31–0 REF	Reference value compared with the respective free-running timer counter (DTCN*n*) as part of the output-compare function.

24.2.5 DMA Timer Capture Registers (DTCR*n*)

Each DTCR*n* latches the corresponding DTCN*n* value during a capture operation when an edge occurs on DTIN*n*, as programmed in DTMR*n*. The internal bus clock is assumed to be the clock source. DTIN*n* cannot simultaneously function as a clocking source and as an input capture pin. Indeterminate operation results if DTIN*n* is set as the clock source when the input capture mode is used.

IPSBAR 0x00_0408 (DTCR0) Access: User read-only
Offset: 0x00_0448 (DTCR1)
0x00_0488 (DTCR2)
0x00_04C8 (DTCR3)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CAP (32-bit capture counter value)																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-6. DTCR*n* Registers

Table 24-6. DTCR*n* Field Descriptions

Field	Description
31–0 CAP	Captures the corresponding DTCN*n* value during a capture operation when an edge occurs on DTIN*n*, as programmed in DTMR*n*.

24.2.6 DMA Timer Counters (DTCN*n*)

The current value of the 32-bit timer counter can be read at anytime without affecting counting. Writes to DTCN*n* clear the timer counter. The timer counter increments on the clock source rising edge (internal bus clock divided by 1, internal bus clock divided by 16, or DTIN*n*).

IPSBAR 0x00_040C (DTCN0) Access: User read/write
Offset: 0x00_044C (DTCN1)
0x00_048C (DTCN2)
0x00_04CC (DTCN3)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CNT (32-bit timer counter value count)																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-7. DMA Timer Counters (DTCN*n*)

Table 24-7. DTCN*n* Field Descriptions

Field	Description
31–0 CNT	Timer counter. Can be read at anytime without affecting counting and any write to this field clears it.

24.3 Functional Description

24.3.1 Prescaler

The prescaler clock input is selected from the internal bus clock (f_{sys} divided by 1 or 16) or from the corresponding timer input, DTIN*n*. DTIN*n* is synchronized to the internal bus clock, and the synchronization delay is between two and three internal bus clocks. The corresponding DTMR*n*[CLK] selects the clock input source. A programmable prescaler divides the clock input by values from 1 to 256. The prescaler output is an input to the 32-bit counter, DTCN*n*.

24.3.2 Capture Mode

Each DMA timer has a 32-bit timer capture register (DTCR*n*) that latches the counter value when the corresponding input capture edge detector senses a defined DTIN*n* transition. The capture edge bits (DTMR*n*[CE]) select the type of transition that triggers the capture and sets the timer event register capture event bit, DTER*n*[CAP]. If DTER*n*[CAP] and DTXMR*n*[DMAEN] are set, a DMA request is asserted. If DTER*n*[CAP] is set and DTXMR*n*[DMAEN] is cleared, an interrupt is asserted.

24.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value. If the reference value is met, DTER*n*[REF] is set.

- If DTMR*n*[ORRI] is set and DTXMR*n*[DMAEN] is cleared, an interrupt is asserted.
- If DTMR*n*[ORRI] and DTXMR*n*[DMAEN] are set, a DMA request is asserted.

If the free run/restart bit (DTMR*n*[FRR]) is set, a new count starts. If it is clear, the timer keeps running.

24.3.4 Output Mode

When a timer reaches the reference value selected by DTRR, it can send an output signal on DTOUT*n*. DTOUT*n* can be an active-low pulse or a toggle of the current output, as selected by the DTMR*n*[OM] bit.

24.4 Initialization/Application Information

The general-purpose timer modules typically, but not necessarily, follow this program order:

- The DTMR*n* and DTXMR*n* registers are configured for the desired function and behavior.
  - Count and compare to a reference value stored in the DTRR*n* register
  - Capture the timer value on an edge detected on DTIN*n*
  - Configure DTOUT*n* output mode
  - Increment counter by 1 or by 65,537 (16-bit mode)
  - Enable/disable interrupt or DMA request on counter reference match or capture edge
- The DTMR*n*[CLK] register is configured to select the clock source to be routed to the prescaler.
  - Internal bus clock (can be divided by 1 or 16)
  - DTIN*n*, the maximum value of DTIN*n* is 1/5 of the internal bus clock, as described in the device's electrical characteristics

NOTE

DTIN*n* may not be configured as a clock source when the timer capture mode is selected or indeterminate operation results.

- The 8-bit DTMR*n*[PS] prescaler value is set.
- Using DTMR*n*[RST], counter is cleared and started.
- Timer events are managed with an interrupt service routine, a DMA request, or by a software polling mechanism.

24.4.1 Code Example

The following code provides an example of how to initialize and use DMA Timer0 for counting time-out periods.

```
DTMR0 EQU IPSBARx+0x400 ;Timer0 mode register
DTMR1 EQU IPSBARx+0x440 ;Timer1 mode register
DTRR0 EQU IPSBARx+0x404 ;Timer0 reference register
DTRR1 EQU IPSBARx+0x444 ;Timer1 reference register
DTCR0 EQU IPSBARx+0x408 ;Timer0 capture register
DTCR1 EQU IPSBARx+0x448 ;Timer1 capture register
DTCN0 EQU IPSBARx+0x40C ;Timer0 counter register
DTCN1 EQU IPSBARx+0x44C ;Timer1 counter register
DTER0 EQU IPSBARx+0x403 ;Timer0 event register
DTER1 EQU IPSBARx+0x443 ;Timer1 event register
* TMR0 is defined as: *
*[PS] = 0xFF,     divide clock by 256
```

```
*[CE] = 00       disable capture event output
*[OM] = 0        output=active-low pulse
*[ORRI] = 0,     disable ref. match output
*[FRR] = 1,      restart mode enabled
*[CLK] = 10,     internal bus clock/16
*[RST] = 0,      timer0 disabled
        move.w #0xFF0C,D0
        move.w D0,TMR0
        move.l #0x0000,D0;writing to the timer counter with any
        move.l DO,TCN0 ;value resets it to zero
        move.l #0xAFAF,DO ;set the timer0 reference to be
        move.l #D0,TRR0 ;defined as 0xAFAF
```

The simple example below uses Timer0 to count time-out loops. A time-out occurs when the reference value, 0xAFAF, is reached.

```
timer0_ex
        clr.l DO
        clr.l D1
        clr.l D2
        move.l #0x0000,D0
        move.l D0,TCN0          ;reset the counter to 0x0000
        move.b #0x03,D0         ;writing ones to TER0[REF,CAP]
        move.b D0,TER0          ;clears the event flags
        move.w TMR0,D0          ;save the contents of TMR0 while setting
        bset #0,D0              ;the 0 bit. This enables timer 0 and starts counting
        move.w D0,TMR0          ;load the value back into the register, setting TMR0[RST]
T0_LOOP
        move.b TER0,D1          ;load TER0 and see if
        btst #1,D1              ;TER0[REF] has been set
        beq T0_LOOP
        addi.l #1,D2            ;Increment D2
        cmp.l #5,D2             ;Did D2 reach 5? (i.e. timer ref has timed)
        beq T0_FINISH           ;If so, end timer0 example. Otherwise jump back.
        move.b #0x02,D0         ;writing one to TER0[REF] clears the event flag
        move.b D0,TER0
        jmp T0_LOOP
T0_FINISH
        HALT                    ;End processing. Example is finished
```

24.4.2 Calculating Time-Out Values

Equation 24-1 determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}n[\text{PS}] + 1) \times (\text{DTRR}n[\text{REF}] + 1) \qquad \textit{Eqn. 24-1}$$

When calculating time-out periods, add one to the prescaler to simplify calculating, because DTMR*n*[PS] equal to 0x00 yields a prescaler of one, and DTMR*n*[PS] equal to 0xFF yields a prescaler of 256.

For example, if a 60-MHz timer clock is divided by 16, DTMR*n*[PS] equals 0x7F, and the timer is referenced at 0xFBC5 (64,453 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{60 \times 10^6 \text{ Hz}} \times 16 \times (127 + 1) \times (64453 + 1) = 2.20 \text{ seconds} \qquad \textit{Eqn. 24-2}$$

Chapter 25
Queued Serial Peripheral Interface (QSPI)

25.1 Introduction

This chapter describes the queued serial peripheral interface (QSPI) module.

25.1.1 Block Diagram

Figure 25-1 illustrates the QSPI module.

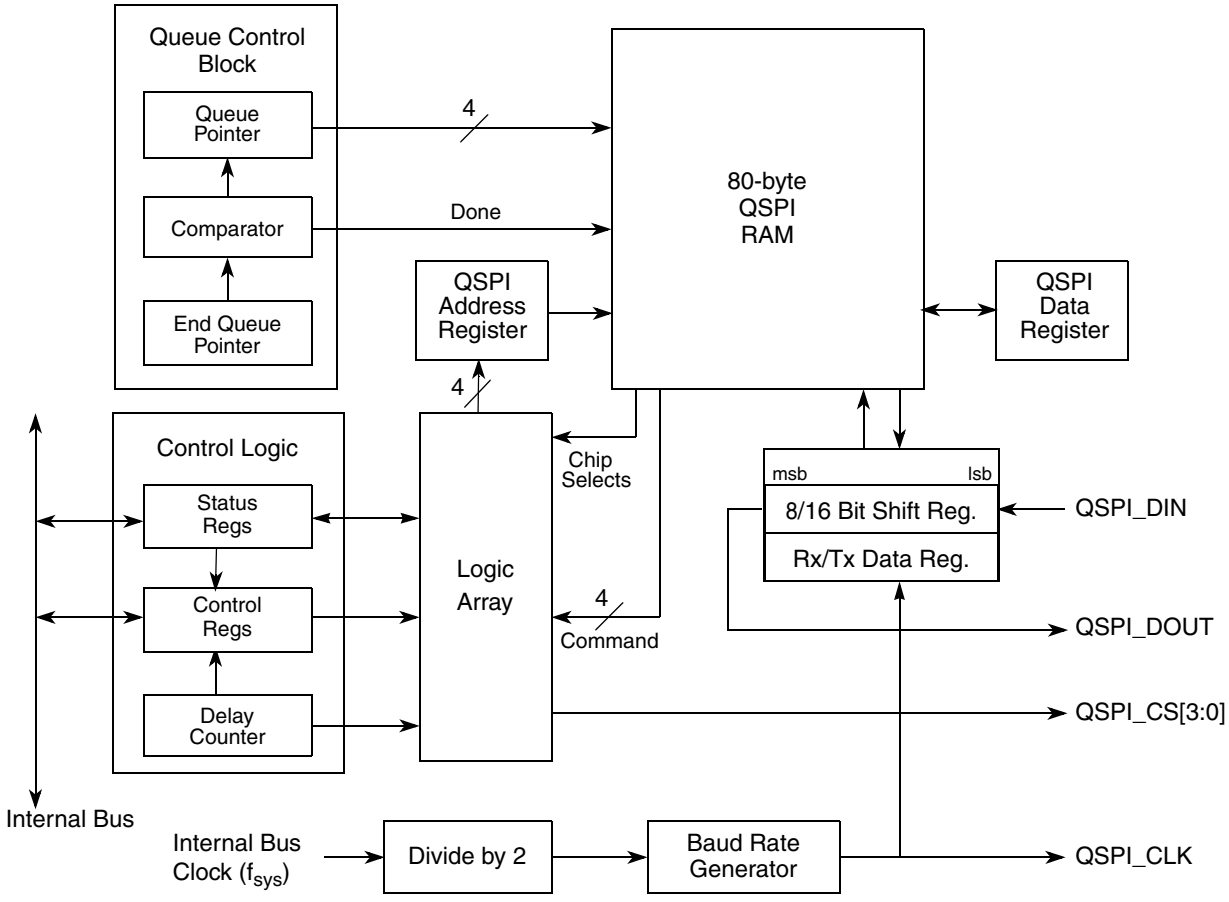


Figure 25-1. QSPI Block Diagram

25.1.2 Overview

The queued serial peripheral interface module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers. Transfer RAM in the QSPI is indirectly accessible using address and data registers.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “General Purpose I/O Module”) prior to configuring the QSPI module.

25.1.3 Features

Features include:

- Programmable queue to support up to 16 transfers without user intervention
  - 80 bytes of data storage provided
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices (All chip selects may not be available on all devices. See Chapter 2, “Signal Descriptions,” for details on which chip-selects are pinned-out.)
- Baud rates from 117.6 Kbps to 15 Mbps at 60 MHz internal bus frequency
- Programmable delays before and after transfers
- Programmable QSPI clock phase and polarity
- Supports wraparound mode for continuous transfers

25.1.4 Modes of Operation

Because the QSPI module only operates in master mode, the master bit in the QSPI mode register (QMR[MSTR]) must be set for the QSPI to function properly. If the master bit is not set, QSPI activity is indeterminate. The QSPI can initiate serial transfers but cannot respond to transfers initiated by other QSPI masters.

25.2 External Signal Description

The module provides access to as many as 15 devices with a total of seven signals: QSPI_DOUT, QSPI_DIN, QSPI_CLK, QSPI_CS[3:0].

Peripheral chip-select signals, QSPI_CS*n*, are used to select an external device as the source or destination for serial data transfer. Signals are asserted when a command in the queue is executed. More than one chip-select signal can be asserted simultaneously.

Although QSPI_CS*n* signals function as simple chip selects in most applications, up to 15 devices can be selected by decoding them with an external 4-to-16 decoder.

Table 25-1. QSPI Input and Output Signals and Functions

Signal Name	Hi-Z or Actively Driven	Function
Data output (QSPI_DOUT)	Configurable	Serial data output from QSPI
Data input (QSPI_DIN)	N/A	Serial data input to QSPI
Serial clock (QSPI_CLK)	Actively driven	Clock output from QSPI
Peripheral chip selects (QSPI_CS*n*)	Actively driven	Peripheral selects from QSPI

25.3 Memory Map/Register Definition

Table 25-2 is the QSPI register memory map. Reading reserved locations returns zeros.

Table 25-2. QSPI Memory Map

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x00_0340	QSPI Mode Register (QMR)	16	R/W	0x0104	25.3.1/25-3
0x00_0344	QSPI Delay Register (QDLYR)	16	R/W	0x0404	25.3.2/25-5
0x00_0348	QSPI Wrap Register (QWR)	16	R/W[2]	0x0000	25.3.3/25-6
0x00_034C	QSPI Interrupt Register (QIR)	16	R/W[2]	0x0000	25.3.4/25-6
0x00_0350	QSPI Address Register (QAR)	16	R/W[2]	0x0000	25.3.5/25-7
0x00_0354	QSPI Data Register (QDR)	16	R/W	0x0000	25.3.6/25-8

1 Addresses not assigned to a register and undefined register bits are reserved for expansion.

2 See the register description for special cases. Some bits may be read- or write-only.

25.3.1 QSPI Mode Register (QMR)

The QMR, shown in Figure 25-2, determines the basic operating modes of the QSPI module. Parameters such as QSPI_CLK polarity and phase, baud rate, master mode operation, and transfer size are determined by this register.

NOTE

Because the QSPI does not operate in slave mode, the master mode enable bit (QMR[MSTR]) must be set for the QSPI module to operate correctly.

NOTE

Because of the implementation of the QSPI module on this device, CPOL and CPHA may be modified only once, typically during software initialization. Changing CPOL and CPHA during operation is not supported.

IPSBAR Offset: 0x00_0340 (QMR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MSTR	0	BITS				CPOL	CPHA	BAUD							
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0

Figure 25-2. QSPI Mode Register (QMR)

Table 25-3. QMR Field Descriptions

Field	Description
15 MSTR	Master mode enable. 0 Reserved, do not use. 1 The QSPI is in master mode. Must be set for the QSPI module to operate correctly.
14	Reserved, must be cleared.
13–10 BITS	Transfer size. Determines the number of bits to be transferred for each entry in the queue. **BITS** / **Bits per Transfer** 0000 / 16 0001–0111 / Reserved 1000 / 8 1001 / 9 1010 / 10 1011 / 11 1100 / 12 1101 / 13 1110 / 14 1111 / 15
9 CPOL	Clock polarity. Defines the clock polarity of QSPI_CLK. 0 The inactive state value of QSPI_CLK is logic level 0. 1 The inactive state value of QSPI_CLK is logic level 1.
8 CPHA	Clock phase. Defines the QSPI_CLK clock-phase. 0 Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. 1 Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.
7–0 BAUD	Baud rate divider. The baud rate is selected by writing a value in the range 2–255. A value of zero disables the QSPI. A value of 1 is an invalid setting. The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression: QMR[BAUD] = $f_{sys/}$ / (2 × [desired QSPI_CLK baud rate])

Figure 25-3 shows an example of a QSPI clocking and data transfer.

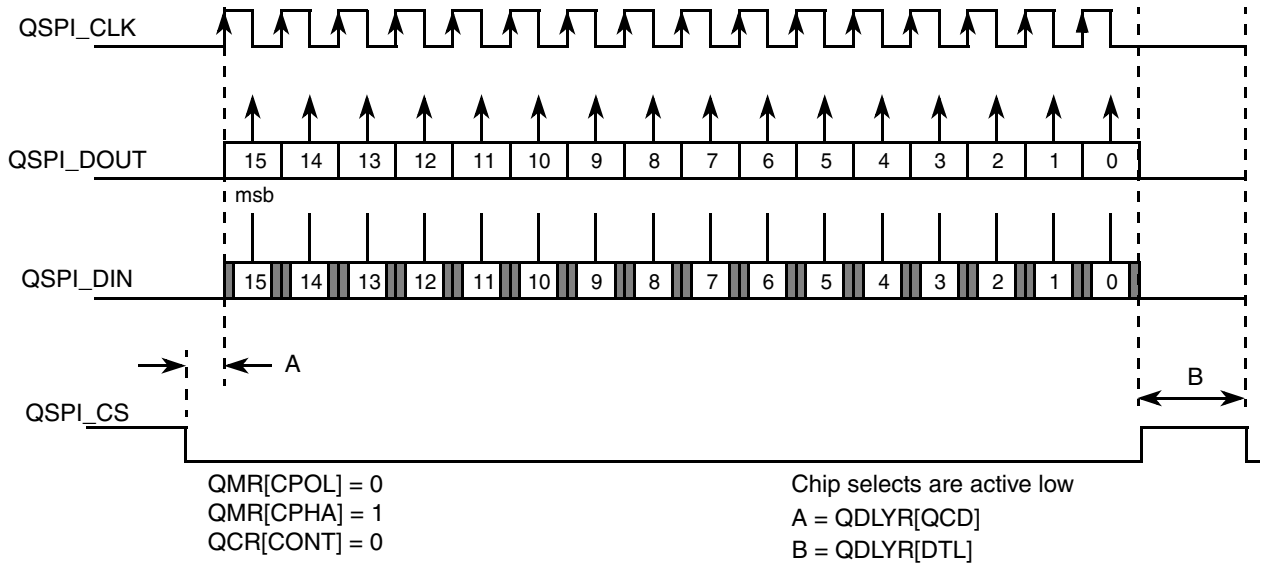


Figure 25-3. QSPI Clocking and Data Transfer Example

25.3.2 QSPI Delay Register (QDLYR)

The QDLYR is used to initiate master mode transfers and to set various delay parameters.

IPSBAR Offset: 0x00_0344 (QDLYR) Access: User read/write

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

Figure 25-4. QSPI Delay Register (QDLYR)

Table 25-4. QDLYR Field Descriptions

Field	Description
15 SPE	QSPI enable. When set, the QSPI initiates transfers in master mode by executing commands in the command RAM. The QSPI clears this bit automatically when a transfer completes. The user can also clear this bit to abort transfer unless QIR[ABRTL] is set. The recommended method for aborting transfers is to set QWR[HALT].
14–8 QCD	QSPI_CLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition. See Section 25.4.3, “Transfer Delays” for information on setting this bit field.
7–0 DTL	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

25.3.3 QSPI Wrap Register (QWR)

The QSPI wrap register provides halt transfer control, wraparound settings, and queue pointer locations.

IPSBAR Offset: 0x00_0348 (QWR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HALT	WREN	WRTO	CSIV	ENDQP				CPTQP				NEWQP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-5. QSPI Wrap Register (QWR)

Table 25-5. QWR Field Descriptions

Field	Description
15 HALT	Halt transfers. Assertion of this bit causes the QSPI to stop execution of commands after it has completed execution of the current command.
14 WREN	Wraparound enable. Enables wraparound mode. 0 Execution stops after executing the command pointed to by QWR[ENDQP]. 1 After executing command pointed to by QWR[ENDQP], wrap back to entry zero, or the entry pointed to by QWR[NEWQP] and continue execution.
13 WRTO	Wraparound location. Determines where the QSPI wraps to in wraparound mode. 0 Wrap to RAM entry zero. 1 Wrap to RAM entry pointed to by QWR[NEWQP].
12 CSIV	QSPI_CS inactive level. 0 QSPI chip select outputs return to zero when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 0, chip selects are active high). 1 QSPI chip select outputs return to one when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 1, chip selects are active low).
11–8 ENDQP	End of queue pointer. Points to the RAM entry that contains the last transfer description in the queue.
7–4 CPTQP	Completed queue entry pointer. Points to the RAM entry that contains the last command to have been completed. This field is read only.
3–0 NEWQP	Start of queue pointer. This 4-bit field points to the first entry in the RAM to be executed on initiating a transfer.

25.3.4 QSPI Interrupt Register (QIR)

The QIR contains QSPI interrupt enables and status flags.

IPSBAR Offset: 0x00_034C (QIR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WCEFB	ABRTB	0	ABRTL	WCEFE	ABRTE	0	SPIFE	0	0	0	0	WCEF	ABRT	0	SPIF
W													w1c	w1c		w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-6. QSPI Interrupt Register (QIR)

Table 25-6. QIR Field Descriptions

Field	Description
15 WCEFB	Write collision access error enable. A write collision occurs during a data transfer when the RAM entry containing the current command is written to by the CPU with the QDR. When this bit is asserted, the write access to QDR results in an access error.
14 ABRTB	Abort access error enable. An abort occurs when QDLYR[SPE] is cleared during a transfer. When set, an attempt to clear QDLYR[SPE] during a transfer results in an access error.
13	Reserved, must be cleared.
12 ABRTL	Abort lock-out. When set, QDLYR[SPE] cannot be cleared by writing to the QDLYR. QDLYR[SPE] is only cleared by the QSPI when a transfer completes.
11 WCEFE	Write collision (WCEF) interrupt enable. 0 Write collision interrupt disabled 1 Write collision interrupt enabled
10 ABRTE	Abort (ABRT) interrupt enable. 0 Abort interrupt disabled 1 Abort interrupt enabled
9	Reserved, must be cleared.
8 SPIFE	QSPI finished (SPIF) interrupt enable. 0 SPIF interrupt disabled 1 SPIF interrupt enabled
7–4	Reserved, must be cleared.
3 WCEF	Write collision error flag. Indicates that an attempt has been made to write to the RAM entry that is currently being executed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
2 ABRT	Abort flag. Indicates that QDLYR[SPE] has been cleared by the user writing to the QDLYR rather than by completion of the command queue by the QSPI. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
1	Reserved, must be cleared.
0 SPIF	QSPI finished flag. Asserted when the QSPI has completed all the commands in the queue. Set on completion of the command pointed to by QWR[ENDQP], and on completion of the current command after assertion of QWR[HALT]. In wraparound mode, this bit is set every time the command pointed to by QWR[ENDQP] is completed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.

25.3.5 QSPI Address Register (QAR)

The QAR is used to specify the location in the QSPI RAM that read and write operations affect. As shown in Section 25.4.1, “QSPI RAM”, the transmit RAM is located at addresses 0x0 to 0xF, the receive RAM is located at 0x10 to 0x1F, and the command RAM is located at 0x20 to 0x2F. (These addresses refer to the QSPI RAM space, not the device memory map.)

NOTE

A read or write to the QSPI RAM causes QAR to increment. However, the QAR does not wrap after the last queue entry within each section of the RAM. The application software must manage address range errors.

IPSBAR Offset: 0x00_0350 (QAR) Access: User read/write

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

Figure 25-7. QSPI Address Register (QAR)

Table 25-7. QAR Field Descriptions

Field	Description
15–6	Reserved, must be cleared.
5–0 ADDR	Address used to read/write the QSPI RAM. Ranges are as follows: 0x00–0x0F Transmit RAM 0x10–0x1F Receive RAM 0x20–0x2F Command RAM 0x30–0x3F Reserved

25.3.6 QSPI Data Register (QDR)

The QDR is used to access QSPI RAM indirectly. The CPU reads and writes all data from and to the QSPI RAM through this register.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR]. This also causes the value in QAR to increment. Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

IPSBAR Offset: 0x00_0354 (QDR) Access: User read/write

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

Figure 25-8. QSPI Data Register (QDR)

Table 25-8. QDR Field Descriptions

Field	Description
15–0 DATA	A write to this field causes data to be written to the QSPI RAM entry specified by QAR[ADDR]. Similarly, a read of this field returns the data in the QSPI RAM at the address specified by QAR[ADDR]. During command RAM accesses (QAR[ADDR] = 0x20–0x2F), only the most significant byte of this field is used.

25.3.7 Command RAM Registers (QCR0–QCR15)

The command RAM is accessed using the upper byte of the QDR; the QSPI cannot modify information in command RAM. There are 16 bytes in the command RAM. Each byte is divided into two fields. The chip select field enables external peripherals for transfer. The command field provides transfer operations.

NOTE

The command RAM is accessed only using the most significant byte of QDR and indirect addressing based on QAR[ADDR].

Address: QAR[ADDR] Access: CPU write-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	CONT	BITSE	DT	DSCK	QSPI_CS				0	0	0	0	0	0	0	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 25-9. Command RAM Registers (QCR0–QCR15)

Table 25-9. QCR0–QCR15 Field Descriptions

Field	Description
15 CONT	Continuous. 0 Chip selects return to inactive level defined by QWR[CSIV] when a single word transfer is complete. 1 Chip selects return to inactive level defined by QWR[CSIV] only after the transfer of the queue entries (max of 16 words). **Note:** To keep the chip selects asserted for transfers beyond 16 words, the QWR[CSIV] bit must be set to control the level that the chip selects return to after the first transfer.
14 BITSE	Bits per transfer enable. 0 Eight bits 1 Number of bits set in QMR[BITS]
13 DT	Delay after transfer enable. 0 Default reset value. 1 The QSPI provides a variable delay at the end of serial transfer to facilitate interfacing with peripherals that have a latency requirement. The delay between transfers is determined by QDLYR[DTL].
12 DSCK	Chip select to QSPI_CLK delay enable. 0 Chip select valid to QSPI_CLK transition is one-half QSPI_CLK period. 1 QDLYR[QCD] specifies the delay from QSPI_CS valid to QSPI_CLK.
11–8 QSPI_CS	Peripheral chip selects. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 11-8 map directly to the corresponding QSPI_CS*n* pins. If more than four chip selects are needed, then an external demultiplexor can be used with the QSPI_CS*n* pins. 0 Enable chip select. 1 Mask chip select. **Note:** Not all chip selects may be available on all device packages. See Chapter 2, “Signal Descriptions,” for details on which chip selects are pinned-out.
7–0	Reserved, must be cleared.

25.4 Functional Description

The QSPI uses a dedicated 80-byte block of static RAM accessible to the module and CPU to perform queued operations. The RAM is divided into three segments:

- 16 command control bytes (command RAM)
- 32 transmit data bytes (transmit data RAM)

- 32 receive data bytes (receive data RAM)

The RAM is organized so that 1 byte of command control data, 1 word of transmit data, and 1 word of receive data comprise 1 of the 16 queue entries (0x0–0xF).

NOTE

Throughout ColdFire documentation, the term word is used to designate a 16-bit data unit. The only exceptions to this appear in discussions of serial communication modules such as QSPI that support variable-length data units. To simplify these discussions, the functional unit is referred to as a word regardless of length.

The user initiates QSPI operation by loading a queue of commands in command RAM, writing transmit data into transmit RAM, and then enabling the QSPI data transfer. The QSPI executes the queued commands and sets the completion flag in the QSPI interrupt register (QIR[SPIF]) to signal their completion. As another option, QIR[SPIFE] can be enabled to generate an interrupt.

The QSPI uses four queue pointers. The user can access three of them through fields in QSPI wrap register (QWR):

- New queue pointer (QWR[NEWQP])—points to the first command in the queue
- Internal queue pointer—points to the command currently being executed
- Completed queue pointer (QWR[CPTQP])—points to the last command executed
- End queue pointer (QWR[ENDQP]) —points to the final command in the queue

The internal pointer is initialized to the same value as QWR[NEWQP]. During normal operation, the following sequence repeats:

1. The command pointed to by the internal pointer is executed.
2. The value in the internal pointer is copied into QWR[CPTQP].
3. The internal pointer is incremented.

Execution continues at the internal pointer address unless the QWR[NEWQP] value is changed. After each command is executed, QWR[ENDQP] and QWR[CPTQP] are compared. When a match occurs, QIR[SPIF] is set and the QSPI stops unless wraparound mode is enabled. Setting QWR[WREN] enables wraparound mode.

QWR[NEWQP] is cleared at reset. When the QSPI is enabled, execution begins at address 0x0 unless another value has been written into QWR[NEWQP]. QWR[ENDQP] is cleared at reset but is changed to show the last queue entry before the QSPI is enabled. QWR[NEWQP] and QWR[ENDQP] can be written at any time. When the QWR[NEWQP] value changes, the internal pointer value also changes unless a transfer is in progress, in which case the transfer completes normally. Leaving QWR[NEWQP] and QWR[ENDQP] set to 0x0 causes a single transfer to occur when the QSPI is enabled.

Data is transferred relative to QSPI_CLK, which can be generated in any one of four combinations of phase and polarity using QMR[CPHA,CPOL]. Data is transferred with the most significant bit (msb) first. The number of bits transferred defaults to 8, but can be set to any value between 8 and 16 by writing a value into the BITSE field of the command RAM (QCR[BITSE]).

25.4.1 QSPI RAM

The QSPI contains an 80-byte block of static RAM that can be accessed by the user and the QSPI. This RAM does not appear in the device memory map, because it can only be accessed by the user indirectly through the QSPI address register (QAR) and the QSPI data register (QDR). The RAM is divided into three segments with 16 addresses each:

- Receive data RAM—the initial destination for all incoming data
- Transmit data RAM—a buffer for all out-bound data
- Command RAM—where commands are loaded

The transmit data and command RAM are user write-only. The receive RAM is user read-only. Figure 25-10 shows the RAM configuration. The RAM contents are undefined immediately after a reset.

The command and data RAM in the QSPI are indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data, and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR] and causes the value in QAR to increment. Correspondingly, a read from QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Relative Address	Register	Function
0x00	QTR0	Transmit RAM
0x01	QTR1	
...	...	16 bits wide
0x0F	QTR15	
0x10	QRR0	Receive RAM
0x11	QRR1	
...	...	16 bits wide
0x1F	QRR15	
0x20	QCR0	Command RAM
0x21	QCR1	
...	...	8 bits wide
0x2F	QCR15	

Figure 25-10. QSPI RAM Model

25.4.1.1 Receive RAM

Data received by the QSPI is stored in the receive RAM segment located at 0x10 to 0x1F in the QSPI RAM space. Read this segment to retrieve data from the QSPI. Data words with less than 16 bits are stored in

the least significant bits of the RAM. Unused bits in a receive queue entry are set to zero upon completion of the individual queue entry. Receive RAM is not writeable.

QWR[CPTQP] shows which queue entries have been executed. The user can query this field to determine which locations in receive RAM contain valid data.

25.4.1.2 Transmit RAM

Data to be transmitted by the QSPI is stored in the transmit RAM segment located at addresses 0x0 to 0xF. The user normally writes 1 word into this segment for each queue command to be executed. The user cannot read data in the transmit RAM.

Outbound data must be written to transmit RAM in a right-justified format. The unused bits are ignored. The QSPI copies the data to its data serializer (shift register) for transmission. The data is transmitted most significant bit first and remains in transmit RAM until overwritten by the user.

25.4.1.3 Command RAM

The CPU writes one byte of control information to this segment for each QSPI command to be executed. Command RAM, referred to as QCR0–15, is write-only memory from a user's perspective.

Command RAM consists of 16 bytes, each divided into two fields. The peripheral chip select field controls the QSPI_CS signal levels for the transfer. The command control field provides transfer options.

A maximum of 16 commands can be in the queue. Queue execution proceeds from the address in QWR[NEWQP] through the address in QWR[ENDQP].

The QSPI executes a queue of commands defined by the control bits in each command RAM entry that sequence the following actions:

- Chip-select pins are activated.
- Data is transmitted from the transmit RAM and received into the receive RAM.
- The synchronous transfer clock QSPI_CLK is generated.

Before any data transfers begin, control data must be written to the command RAM, and any out-bound data must be written to the transmit RAM. Also, the queue pointers must be initialized to the first and last entries in the command queue.

Data transfer is synchronized with the internally generated QSPI_CLK, whose phase and polarity are controlled by QMR[CPHA] and QMR[CPOL]. These control bits determine which QSPI_CLK edge is used to drive outgoing data and to latch incoming data.

25.4.2 Baud Rate Selection

The maximum QSPI clock frequency is one-fourth the clock frequency of the internal bus clock (f_{sys}). Baud rate is selected by writing a value from 2–255 into QMR[BAUD]. The QSPI uses a prescaler to derive the QSPI_CLK rate from the internal bus clock divided by two. Table 25-10 shows the QSPI_CLK frequency as a function of internal bus clock and baud rate.

A baud rate value of zero turns off the QSPI_CLK.

The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression:

$$\text{QMR[BAUD]} = \frac{f_{sys}}{2 \times [\text{desired QSPI\_CLK baud rate}]} \qquad \textit{Eqn. 25-1}$$

Table 25-10. QSPI_CLK Frequency as Function of Internal Bus Clock and Baud Rate

Internal Bus Clock = 60 MHz	
QMR [BAUD]	**QSPI_CLK**
2	15 MHz
4	7.5 MHz
8	3.75 MHz
16	1.88 MHz
32	937.5 kHz
255	117.6 kHz

25.4.3 Transfer Delays

The QSPI supports programmable delays for the QSPI_CS signals before and after a transfer. The time between QSPI_CS assertion and the leading QSPI_CLK edge, and the time between the end of one transfer and the beginning of the next, are both independently programmable.

The chip select to clock delay enable bit in the command RAM, QCR[DSCK], enables the programmable delay period from QSPI_CS assertion until the leading edge of QSPI_CLK. QDLYR[QCD] determines the period of delay before the leading edge of QSPI_CLK. The following expression determines the actual delay before the QSPI_CLK leading edge:

$$\text{QSPI\_CS-to-QSPI\_CLK delay} = \frac{\text{QDLYR[QCD]}}{f_{sys}} \qquad \textit{Eqn. 25-2}$$

QDLYR[QCD] has a range of 1–127.

When QDLYR[QCD] or QCR[DSCK] equals zero, the standard delay of one-half the QSPI_CLK period is used.

The command RAM delay after transmit enable bit, QCR[DT], enables the programmable delay period from the negation of the QSPI_CS signals until the start of the next transfer. The delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. There are two transfer delay options: the user can choose to delay a standard period after serial transfer is complete or can specify a delay period. Writing a value to QDLYR[DTL] specifies a delay period. QCR[DT] determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay when DT equals 1:

$$\text{Delay after transfer} = \frac{32 \times \text{QDLYR[DTL]}}{f_{sys}} \quad (\text{DT} = 1) \qquad \textit{Eqn. 25-3}$$

where QDLYR[DTL] has a range of 1–255. A zero value for DTL causes a delay-after-transfer value of $8192/f_{sys}$. Standard delay period (DT = 0) is calculated by the following:

$$\text{Standard delay after transfer} = \frac{17}{f_{sys}} \quad (DT = 0) \qquad \textit{Eqn. 25-4}$$

Adequate delay between transfers must be specified for long data streams because the QSPI module requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If the internal bus clock is operating at a slower rate, the delay between transfers must be increased proportionately.

25.4.4 Transfer Length

There are two transfer length options. The user can choose a default value of 8 bits or a programmed value of 8 to 16 bits. The programmed value must be written into QMR[BITS]. The command RAM bits per transfer enable field, QCR[BITSE], determines whether the default value (BITSE = 0) or the BITS[3–0] value (BITSE = 1) is used. QMR[BITS] indicates the required number of bits to be transferred, with the default value of 16 bits.

25.4.5 Data Transfer

The transfer operation is initiated by setting QDLYR[SPE]. Shortly after QDLYR[SPE] is set, the QSPI executes the command at the command RAM address pointed to by QWR[NEWQP]. Data at the pointer address in transmit RAM is loaded into the data serializer and transmitted. Data that is simultaneously received is stored at the pointer address in receive RAM.

When the proper number of bits has been transferred, the QSPI stores the working queue pointer value in QWR[CPTQP], increments the working queue pointer, and loads the next data for transfer from the transmit RAM. The command pointed to by the incremented working queue pointer is executed next unless a new value has been written to QWR[NEWQP]. If a new queue pointer value is written while a transfer is in progress, the current transfer is completed normally.

When the CONT bit in the command RAM is set, the QSPI_CS*n* signals are asserted between transfers. When CONT is cleared, QSPI_CS*n* are negated between transfers. The QSPI_CS*n* signals are not high impedance.

When the QSPI reaches the end of the queue, it asserts the SPIF flag, QIR[SPIF]. If QIR[SPIFE] is set, an interrupt request is generated when QIR[SPIF] is asserted. Then the QSPI clears QDLYR[SPE] and stops, unless wraparound mode is enabled.

Wraparound mode is enabled by setting QWR[WREN]. The queue can wrap to pointer address 0x0, or to the address specified by QWR[NEWQP], depending on the state of QWR[WRTO].

In wraparound mode, the QSPI cycles through the queue continuously, even while requesting interrupt service. QDLYR[SPE] is not cleared when the last command in the queue is executed. New receive data overwrites previously received data in the receive RAM. Each time the end of the queue is reached,

QIR[SPIFE] is set. QIR[SPIF] is not automatically reset. If interrupt driven QSPI service is used, the service routine must clear QIR[SPIF] to abort the current request. Additional interrupt requests during servicing can be prevented by clearing QIR[SPIFE].

There are two recommended methods of exiting wraparound mode: clearing QWR[WREN] or setting QWR[HALT]. Exiting wraparound mode by clearing QDLYR[SPE] is not recommended because this may abort a serial transfer in progress. The QSPI sets SPIF, clears QDLYR[SPE], and stops the first time it reaches the end of the queue after QWR[WREN] is cleared. After QWR[HALT] is set, the QSPI finishes the current transfer, then stops executing commands. After the QSPI stops, QDLYR[SPE] can be cleared.

25.5 Initialization/Application Information

The following steps are necessary to set up the QSPI 12-bit data transfers and a QSPI_CLK of MHz. The QSPI RAM is set up for a queue of 16 transfers. All four QSPI_CS signals are used in this example.

1. Write the QMR with 0xB308 to set up 12-bit data words with the data shifted on the falling clock edge, and a QSPI_CLK frequency of 3.75 MHz (assuming a 60-MHz internal bus clock).
2. Write QDLYR with the desired delays.
3. Write QIR with 0xD00F to enable write collision, abort bus errors, and clear any interrupts.
4. Write QAR with 0x0020 to select the first command RAM entry.
5. Write QDR with 0x7E00, 0x7E00, 0x7E00, 0x7E00, 0x7D00, 0x7D00, 0x7D00, 0x7D00, 0x7B00, 0x7B00, 0x7B00, 0x7B00, 0x7700, 0x7700, 0x7700, and 0x7700 to set up four transfers for each chip select. The chip selects are active low in this example.
6. Write QAR with 0x0000 to select the first transmit RAM entry.
7. Write QDR with sixteen 12-bit words of data.
8. Write QWR with 0x0F00 to set up a queue beginning at entry 0 and ending at entry 15.
9. Set QDLYR[SPE] to enable the transfers.
10. Wait until the transfers are complete. QIR[SPIF] is set when the transfers are complete.
11. Write QAR with 0x0010 to select the first receive RAM entry.
12. Read QDR to get the received data for each transfer.
13. Repeat steps 5 through 13 to do another transfer.

Chapter 26
UART Modules

26.1 Introduction

This chapter describes the use of the three universal asynchronous receiver/transmitters (UARTs) and includes programming examples.

NOTE

The designation *n* appears throughout this section to refer to registers or signals associated with one of the three identical UART modules: UART0, UART1, or UART2.

26.1.1 Overview

The internal bus clock can clock each of the three independent UARTs, eliminating the need for an external UART clock. As Figure 26-1 shows, each UART module interfaces directly to the CPU and consists of:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic

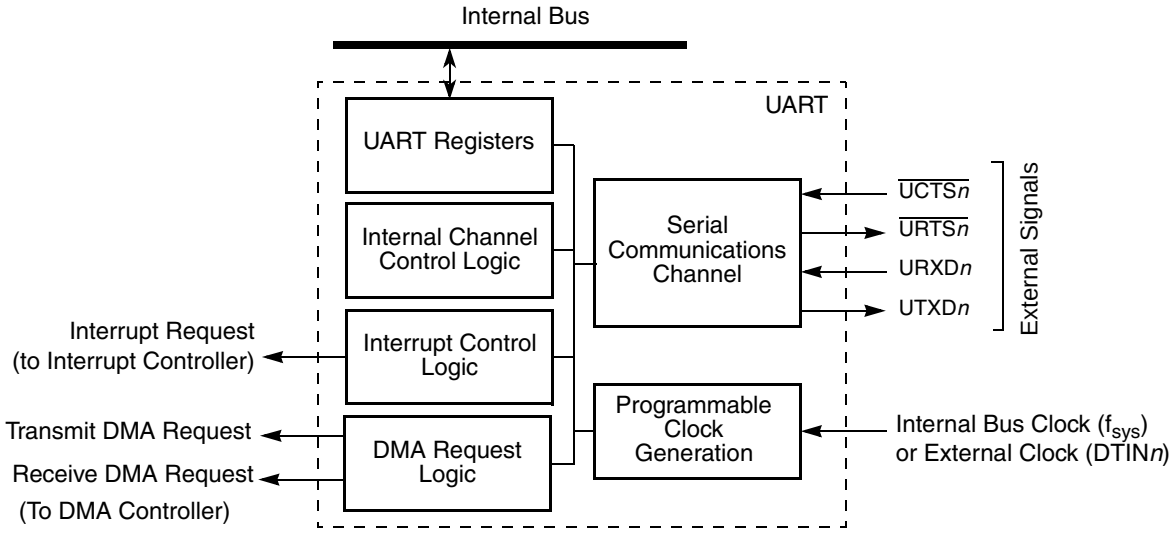


Figure 26-1. UART Block Diagram

NOTE

The DTIN*n* pin can clock UART*n*. However, if the timers are operating and the UART uses DTIN*n* as a clock source, input capture mode is not available for that timer.

The serial communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter deriving an operating frequency from the internal bus clock or an external clock using the timer pin. The transmitter converts parallel data from the CPU to a serial bit stream, inserting appropriate start, stop, and parity bits. It outputs the resulting stream on the transmitter serial data output (UTXD*n*). See Section 26.4.2.1, "Transmitter."

The receiver converts serial data from the receiver serial data input (URXD*n*) to parallel format, checks for a start, stop, and parity bits, or break conditions, and transfers the assembled character onto the bus during read operations. The receiver may be polled, interrupt driven, or use DMA requests for servicing. See Section 26.4.2.2, "Receiver."

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, "General Purpose I/O Module") prior to configuring the UART module.

26.1.2 Features

The device contains three independent UART modules with:

- Each clocked by external clock or internal bus clock (eliminates need for an external UART clock)
- Full-duplex asynchronous/synchronous receiver/transmitter
- Quadruple-buffered receiver
- Double-buffered transmitter
- Independently programmable receiver and transmitter clock sources
- Programmable data format:
  - 5–8 data bits plus parity
  - Odd, even, no parity, or force parity
  - One, one-and-a-half, or two stop bits
- Each serial channel programmable to normal (full-duplex), automatic echo, local loopback, or remote loopback mode
- Automatic wake-up mode for multidrop applications
- Four maskable interrupt conditions
- All three UARTs have DMA request capability
- Parity, framing, and overrun error detection
- False-start bit detection
- Line-break detection and generation
- Detection of breaks originating in the middle of a character

- Start/end break interrupt/status

26.2 External Signal Description

Table 26-1 briefly describes the UART module signals.

Table 26-1. UART Module External Signals

Signal	Description
UTXD*n*	Transmitter Serial Data Output. UTXD*n* is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on UTXD*n* on the falling edge of the clock source, with the least significant bit (lsb) sent first.
URXD*n*	Receiver Serial Data Input. Data received on URXD*n* is sampled on the rising edge of the clock source, with the lsb received first.
$\overline{\text{UCTS}n}$	Clear-to- Send. This input can generate an interrupt on a change of state.
$\overline{\text{URTS}n}$	Request-to-Send. This output can be programmed to be negated or asserted automatically by the receiver or the transmitter. When connected to a transmitter's $\overline{\text{UCTS}n}$, $\overline{\text{URTS}n}$ can control serial data flow.

Figure 26-2 shows a signal configuration for a UART/RS-232 interface.

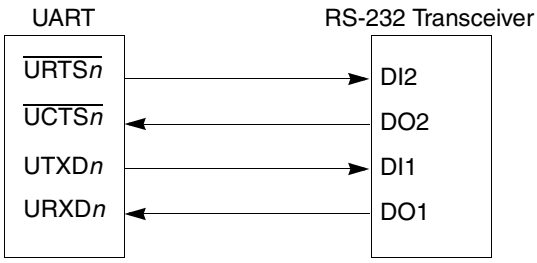


Figure 26-2. UART/RS-232 Interface

26.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in Section 26.5, "Initialization/Application Information," describe basic UART module programming. Writing control bytes into the appropriate registers controls the operation of the UART module.

NOTE

UART registers are accessible only as bytes.

NOTE

Interrupt can mean an interrupt request asserted to the CPU or a DMA request.

Table 26-2. UART Module Memory Map

UART0 UART1 UART2	Register	Width (bit)	Access	Reset Value	Section/Page
0x00 0x0 0x0	UART Mode Registers[1] (UMR1*n*), (UMR2*n*)	8	R/W	0x00	26.3.1/26-5 26.3.2/26-6
0x04 0x4 0x4	UART Status Register (USR*n*)	8	R	0x00	26.3.3/26-8
	UART Clock Select Register[1](UCSR*n*)	8	W	See Section	26.3.4/26-9
0x08 0x8 0x8	UART Command Registers (UCR*n*)	8	W	0x00	26.3.5/26-9
0x0C 0xC 0xC	UART Receive Buffers (URB*n*)	8	R	0xFF	26.3.6/26-11
	UART Transmit Buffers (UTB*n*)	8	W	0x00	26.3.7/26-12
0x10 0x0 0x0	UART Input Port Change Register (UIPCR*n*)	8	R	See Section	26.3.8/26-12
	UART Auxiliary Control Register (UACR*n*)	8	W	0x00	26.3.9/26-13
0x14 0x4 0x4	UART Interrupt Status Register (UISR*n*)	8	R	0x00	26.3.10/26-13
	UART Interrupt Mask Register (UIMR*n*)	8	W	0x00	
0x18 0x8 0x8	UART Baud Rate Generator Register (UBG1*n*)	8	W[2]	0x00	26.3.11/26-15
0x1C 0xC 0xC	UART Baud Rate Generator Register (UBG2*n*)	8	W[2]	0x00	26.3.11/26-15
0x34 0x4 0x4	UART Input Port Register (UIP*n*)	8	R	0xFF	26.3.12/26-15
0x38 0x8 0x8	UART Output Port Bit Set Command Register (UOP1*n*)	8	W[2]	0x00	26.3.13/26-16
0x3C 0xC 0xC	UART Output Port Bit Reset Command Register (UOP0*n*)	8	W[2]	0x00	26.3.13/26-16

1 UMR1*n*, UMR2*n*, and UCSR*n* must be changed only after the receiver/transmitter is issued a software reset command. If operation is not disabled, undesirable results may occur.

2 Reading this register results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

26.3.1 UART Mode Registers 1 (UMR1*n*)

The UMR1*n* registers control UART module configuration. UMR1*n* can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCR*n*[MISC]. After UMR1*n* is read or written, the pointer points to UMR2*n*.

IPSBAR 0x00_0200 (UMR10)
Offset: 0x00_0240 (UMR11)
0x00_0280 (UMR12)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	RXRTS	RXIRQ/ FFULL	ERR	PM		PT	B/C	
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 26-3. UART Mode Registers 1 (UMR1*n*)

Table 26-3. UMR1*n* Field Descriptions

Field	Description
7 RXRTS	Receiver request-to-send. Allows the $\overline{\text{URTS}}n$ output to control the $\overline{\text{UCTS}}n$ input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for $\overline{\text{URTS}}n$ control, $\overline{\text{URTS}}n$ control is disabled for both. Transmitter RTS control is configured in UMR2*n*[TXRTS]. 0 The receiver has no effect on $\overline{\text{URTS}}n$. 1 When a valid start bit is received, $\overline{\text{URTS}}n$ is negated if the UART's FIFO is full. $\overline{\text{URTS}}n$ is reasserted when the FIFO has an empty position available.
6 RXIRQ/ FFULL	Receiver interrupt select. 0 RXRDY is the source generating interrupt or DMA requests. 1 FFULL is the source generating interrupt or DMA requests.
5 ERR	Error mode. Configures the FIFO status bits, USR*n*[RB,FE,PE]. 0 Character mode. The USR*n* values reflect the status of the character at the top of the FIFO. ERR must be 0 for correct A/D flag information when in multidrop mode. 1 Block mode. The USR*n* values are the logical OR of the status for all characters reaching the top of the FIFO since the last RESET ERROR STATUS command for the UART was issued. See Section 26.3.5, "UART Command Registers (UCRn)."
4–3 PM	Parity mode. Selects the parity or multidrop mode for the UART. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. The value of PM affects PT, as shown below.

Table 26-3. UMR1*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PT</td><td>Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11).
<table>
<tr><th>PM</th><th>Parity Mode</th><th>Parity Type (PT= 0)</th><th>Parity Type (PT= 1)</th></tr>
<tr><td>00</td><td>With parity</td><td>Even parity</td><td>Odd parity</td></tr>
<tr><td>01</td><td>Force parity</td><td>Low parity</td><td>High parity</td></tr>
<tr><td>10</td><td>No parity</td><td colspan="2">N/A</td></tr>
<tr><td>11</td><td>Multidrop mode</td><td>Data character</td><td>Address character</td></tr>
</table></td></tr>
<tr><td>1–0
B/C</td><td>Bits per character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits.
00 5 bits
01 6 bits
10 7 bits
11 8 bits</td></tr>
</table>

26.3.2 UART Mode Register 2 (UMR2*n*)

The UMR2*n* registers control UART module configuration. UMR2*n* can be read or written when the mode register pointer points to it, which occurs after any access to UMR1*n*. UMR2*n* accesses do not update the pointer.

IPSBAR 0x00_0200 (UMR20)
Offset: 0x00_0240 (UMR21)
0x00_0280 (UMR22)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	CM		TXRTS	TXCTS	SB			
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 26-4. UART Mode Registers 2 (UMR2*n*)

Table 26-4. UMR2*n* Field Descriptions

Field	Description
7–6 CM	Channel mode. Selects a channel mode. Section 26.4.3, "Looping Modes," describes individual modes. 00 Normal 01 Automatic echo 10 Local loopback 11 Remote loopback
5 TXRTS	Transmitter ready-to-send. Controls negation of $\overline{\text{URTS}}n$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same UART for $\overline{\text{URTS}}n$ control is not permitted and disables $\overline{\text{URTS}}n$ control for both. 0 The transmitter has no effect on $\overline{\text{URTS}}n$. 1 In applications where the transmitter is disabled after transmission completes, setting this bit automatically clears UOP[RTS] one bit time after any characters in the transmitter shift and holding registers are completely sent, including the programmed number of stop bits.
4 TXCTS	Transmitter clear-to-send. If TXCTS and TXRTS are set, TXCTS controls the operation of the transmitter. 0 $\overline{\text{UCTS}}n$ has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\text{UCTS}}n$ each time it is ready to send a character. If $\overline{\text{UCTS}}n$ is asserted, the character is sent; if it is deasserted, the signal UTXD*n* remains in the high state and transmission is delayed until $\overline{\text{UCTS}}n$ is asserted. Changes in $\overline{\text{UCTS}}n$ as a character is being sent do not affect its transmission.
3–0 SB	Stop-bit length control. Selects length of stop bit appended to the transmitted character. Stop-bit lengths of 9/16 to 2 bits are programmable for 6–8 bit characters. Lengths of 1-1/16 to 2 bits are programmable for 5-bit characters. In all cases, the receiver checks only for a high condition at the center of the first stop-bit position, one bit time after the last data bit or after the parity bit, if parity is enabled. If an external 1x clock is used for the transmitter, clearing bit 3 selects one stop bit and setting bit 3 selects two stop bits for transmission.

SB	5 Bits	6–8 Bits	SB	5–8 Bits
0000	1.063	0.563	1000	1.563
0001	1.125	0.625	1001	1.625
0010	1.188	0.688	1010	1.688
0011	1.250	0.750	1011	1.750
0100	1.313	0.813	1100	1.813
0101	1.375	0.875	1101	1.875
0110	1.438	0.938	1110	1.938
0111	1.500	1.000	1111	2.000

26.3.3 UART Status Registers (USR*n*)

The USR*n* registers show the status of the transmitter, the receiver, and the FIFO.

IPSBAR 0x00_0204 (USR0) Access: User read-only
Offset: 0x00_0244 (USR1)
0x00_0284 (USR2)

	7	6	5	4	3	2	1	0
R	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
W								
Reset:	0	0	0	0	0	0	0	0

Figure 26-5. UART Status Registers (USR*n*)

Table 26-5. USR*n* Field Descriptions

Field	Description
7 RB	Received break. The received break circuit detects breaks originating in the middle of a received character. However, a break in the middle of a character must persist until the end of the next detected character time. 0 No break was received. 1 An all-zero character of the programmed length was received without a stop bit. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until URXD*n* returns to the high state for at least one-half bit time, which equals two successive edges of the UART clock. RB is valid only when RXRDY is set.
6 FE	Framing error. 0 No framing error occurred. 1 No stop bit was detected when the corresponding data character in the FIFO was received. The stop-bit check occurs in the middle of the first stop-bit position. FE is valid only when RXRDY is set.
5 PE	Parity error. Valid only if RXRDY is set. 0 No parity error occurred. 1 If UMR1*n*[PM] equals 0*x* (with parity or force parity), the corresponding character in the FIFO was received with incorrect parity. If UMR1*n*[PM] equals 11 (multidrop), PE stores the received address or data (A/D) bit. PE is valid only when RXRDY is set.
4 OE	Overrun error. Indicates whether an overrun occurs. 0 No overrun occurred. 1 One or more characters in the received data stream have been lost. OE is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. The RESET ERROR STATUS command in UCR*n* clears OE.
3 TEMP	Transmitter empty. 0 The transmit buffer is not empty. A character is shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming UCR*n*[TC]. 1 The transmitter has underrun (the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.
2 TXRDY	Transmitter ready. 0 The CPU loaded the transmitter holding register, or the transmitter is disabled. 1 The transmitter holding register is empty and ready for a character. TXRDY is set when a character is sent to the transmitter shift register or when the transmitter is first enabled. If the transmitter is disabled, characters loaded into the transmitter holding register are not sent.

Table 26-5. USR*n* Field Descriptions (continued)

Field	Description
1 FFULL	FIFO full. 0 The FIFO is not full but may hold up to two unread characters. 1 A character was received and the receiver FIFO is now full. Any characters received when the FIFO is full are lost.
0 RXRDY	Receiver ready. 0 The CPU has read the receive buffer and no characters remain in the FIFO after this read. 1 One or more characters were received and are waiting in the receive buffer FIFO.

26.3.4 UART Clock Select Registers (UCSR*n*)

The UCSRs select an external clock on the DTIN input (divided by 1 or 16) or a prescaled internal bus clock as the clocking source for the transmitter and receiver. See Section 26.4.1, "Transmitter/Receiver Clock Source." The transmitter and receiver can use different clock sources. To use the internal bus clock for both, set UCSR*n* to 0xDD.

IPSBAR 0x00_0204 (UCSR0) Access: User write-only
Offset: 0x00_0244 (UCSR1)
0x00_0284 (UCSR2)

	7	6	5	4	3	2	1	0
R								
W	RCS				TCS			
Reset:	See Note				See Note			

Note: The RCS and TCS reset values are set so the receiver and transmitter use the prescaled internal bus clock as their clock source.

Figure 26-6. UART Clock Select Registers (UCSR*n*)

Table 26-6. UCSR*n* Field Descriptions

Field	Description
7–4 RCS	Receiver clock select. Selects the clock source for the receiver. 1101 Prescaled internal bus clock (f_{sys}) 1110 DTIN*n* divided by 16 1111 DTIN*n*
3–0 TCS	Transmitter clock select. Selects the clock source for the transmitter. 1101 Prescaled internal bus clock (f_{sys}) 1110 DTIN*n* divided by 16 1111 DTIN*n*

26.3.5 UART Command Registers (UCR*n*)

The UCRs supply commands to the UART. Only multiple commands that do not conflict can be specified in a single write to a UCR*n*. For example, RESET TRANSMITTER and ENABLE TRANSMITTER cannot be specified in one command.

IPSBAR 0x00_0208 (UCR0)
Offset: 0x00_0248 (UCR1)
0x00_0288 (UCR2)

Access: User write-only

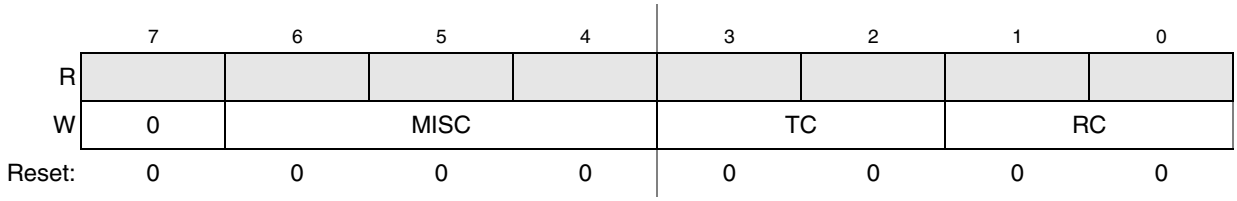


Figure 26-7. UART Command Registers (UCR*n*)

Table 26-7 describes UCR*n* fields and commands. Examples in Section 26.4.2, "Transmitter and Receiver Operating Modes," show how these commands are used.

Table 26-7. UCR*n* Field Descriptions

Field	Description
7	Reserved, must be cleared.
6–4 MISC	MISC Field (this field selects a single command)

	Command	Description
000	NO COMMAND	—
001	RESET MODE REGISTER POINTER	Causes the mode register pointer to point to UMR1*n*.
010	RESET RECEIVER	Immediately disables the receiver, clears USR*n*[FFULL,RXRDY], and reinitializes the receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfiguring the receiver.
011	RESET TRANSMITTER	Immediately disables the transmitter and clears USR*n*[TXEMP,TXRDY]. No other registers are altered. Because it places the transmitter in a known state, use this command instead of TRANSMITTER DISABLE when reconfiguring the transmitter.
100	RESET ERROR STATUS	Clears USR*n*[RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received.
101	RESET BREAK – CHANGE INTERRUPT	Clears the delta break bit, UISR*n*[DB].
110	START BREAK	Forces UTXD*n* low. If the transmitter is empty, break may be delayed up to one bit time. If the transmitter is active, break starts when character transmission completes. Break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break. Transmitter must be enabled for the command to be accepted. This command ignores the state of $\overline{\text{UCTS}}n$.
111	STOP BREAK	Causes UTXD*n* to go high (mark) within two bit times. Any characters in the transmit buffer are sent.

Table 26-7. UCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–2
TC</td><td>Transmit command field. Selects a single transmit command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the transmitter to stay in its current mode: if the transmitter is enabled, it remains enabled; if the transmitter is disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>TRANSMITTER ENABLE</td><td>Enables operation of the UART’s transmitter. USRn[TXEMP,TXRDY] are set. If the transmitter is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>TRANSMITTER DISABLE</td><td>Terminates transmitter operation and clears USRn[TXEMP,TXRDY]. If a character is being sent when the transmitter is disabled, transmission completes before the transmitter becomes inactive. If the transmitter is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
<tr><td>1–0
RC</td><td>Receive command field. Selects a single receive command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>RECEIVER ENABLE</td><td>If the UART module is not in multidrop mode (UMR1n[PM] ≠ 11), RECEIVER ENABLE enables the UART's receiver and forces it into search-for-start-bit state. If the receiver is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>RECEIVER DISABLE</td><td>Disables the receiver immediately. Any character being received is lost. The command does not affect receiver status bits or other control registers. If the UART module is programmed for local loopback or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
</table>

26.3.6 UART Receive Buffers (URB*n*)

The receive buffers contain one serial shift register and three receiver holding registers, which act as a FIFO. URXD*n* is connected to the serial shift register. The CPU reads from the top of the FIFO while the receiver shifts and updates from the bottom when the shift register is full (see Figure 26-18). RB contains the character in the receiver.

IPSBAR 0x00_020C (URB0) Access: User read-only
Offset: 0x00_024C (URB1)
0x00_028C (URB2)

	7	6	5	4	3	2	1	0
R	RB							
W								
Reset:	1	1	1	1	1	1	1	1

Figure 26-8. UART Receive Buffer (URB*n*)

26.3.7 UART Transmit Buffers (UTB*n*)

The transmit buffers consist of the transmitter holding register and the transmitter shift register. The holding register accepts characters from the bus master if UART's USR*n*[TXRDY] is set. A write to the transmit buffer clears USR*n*[TXRDY], inhibiting any more characters until the shift register can accept more data. When the shift register is empty, it checks if the holding register has a valid character to be sent (TXRDY = 0). If there is a valid character, the shift register loads it and sets USR*n*[TXRDY] again. Writes to the transmit buffer when the UART's TXRDY is cleared and the transmitter is disabled have no effect on the transmit buffer.

Figure 26-9 shows UTB*n*. TB contains the character in the transmit buffer.

IPSBAR 0x00_020C (UTB0) Access: User write-only
Offset: 0x00_024C (UTB1)
0x00_028C (UTB2)

	7	6	5	4	3	2	1	0
R								
W	TB							
Reset:	0	0	0	0	0	0	0	0

Figure 26-9. UART Transmit Buffer (UTB*n*)

26.3.8 UART Input Port Change Registers (UIPCR*n*)

The UIPCRs hold the current state and the change-of-state for $\overline{\text{UCTS}}n$.

IPSBAR 0x00_0210 (UIPCR0) Access: User read-only
Offset: 0x00_0250 (UIPCR1)
0x00_0290 (UIPCR2)

	7	6	5	4	3	2	1	0
R	0	0	0	COS	1	1	1	CTS
W								
Reset:	0	0	0	0	1	1	1	$\overline{\text{UCTS}}n$

Figure 26-10. UART Input Port Changed Registers (UIPCR*n*)

Table 26-8. UIPCR*n* Field Descriptions

Field	Description
7–5	Reserved
4 COS	Change of state (high-to-low or low-to-high transition). 0 No change-of-state since the CPU last read UIPCR*n*. Reading UIPCR*n* clears UISR*n*[COS]. 1 A change-of-state longer than 25–50 μs occurred on the $\overline{\text{UCTS}n}$ input. UACR*n* can be programmed to generate an interrupt to the CPU when a change of state is detected.
3–1	Reserved
0 CTS	Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of $\overline{\text{UCTS}n}$. If $\overline{\text{UCTS}n}$ is detected asserted at that time, COS is set, which initiates an interrupt if UACR*n*[IEC] is enabled. 0 The current state of the $\overline{\text{UCTS}n}$ input is asserted. 1 The current state of the $\overline{\text{UCTS}n}$ input is deasserted.

26.3.9 UART Auxiliary Control Register (UACR*n*)

The UACRs control the input enable.

IPSBAR 0x00_0210 (UACR0) Access: User write-only
Offset: 0x00_0250 (UACR1)
0x00_0290 (UACR2)

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	IEC
Reset:	0	0	0	0	0	0	0	0

Figure 26-11. UART Auxiliary Control Registers (UACR*n*)

Table 26-9. UACR*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 IEC	Input enable control. 0 Setting the corresponding UIPCR*n* bit has no effect on UISR*n*[COS]. 1 UISR*n*[COS] is set and an interrupt is generated when the UIPCR*n*[COS] is set by an external transition on the $\overline{\text{UCTS}n}$ input (if UIMR*n*[COS] = 1).

26.3.10 UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

The UISRs provide status for all potential interrupt sources. UISR*n* contents are masked by UIMR*n*. If corresponding UISR*n* and UIMR*n* bits are set, internal interrupt output is asserted. If a UIMR*n* bit is cleared, state of the corresponding UISR*n* bit has no effect on the output.

The UISR*n* and UIMR*n* registers share the same space in memory. Reading this register provides the user with interrupt status, while writing controls the mask bits.

NOTE

True status is provided in the UISR*n* regardless of UIMR*n* settings. UISR*n* is cleared when the UART module is reset.

IPSBAR 0x00_0214 (UISR0) Access: User read/write
Offset: 0x00_0254 (UISR1)
0x00_0294 (UISR2)

	7	6	5	4	3	2	1	0
R (UISR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
W (UIMR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset:	0	0	0	0	0	0	0	0

Figure 26-12. UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

Table 26-10. UISR*n*/UIMR*n* Field Descriptions

Field	Description
7 COS	Change-of-state. 0 UIPCR*n*[COS] is not selected. 1 Change-of-state occurred on $\overline{\text{UCTS}}$*n* and was programmed in UACR*n*[IEC] to cause an interrupt.
6–3	Reserved, must be cleared.
2 DB	Delta break. 0 No new break-change condition to report. Section 26.3.5, "UART Command Registers (UCRn)," describes the RESET BREAK-CHANGE INTERRUPT command. 1 The receiver detected the beginning or end of a received break.
1 FFULL/ RXRDY	Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USR*n*[FIFO] and USR*n*[RXRDY]
0 TXRDY	Transmitter ready. This bit is the duplication of USR*n*[TXRDY]. 0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TXRDY is cleared are not sent. 1 The transmitter holding register is empty and ready to be loaded with a character.

UIMR*n* [FFULL/RXRDY]	UISR*n* [FFULL/RXRDY]	UMR1*n*[FFULL/RXRDY]	
		0 (RXRDY)	1 (FIFO)
0	0	Receiver not ready	FIFO not full
1	0	Receiver not ready	FIFO not full
0	1	Receiver is ready, Do not interrupt	FIFO is full, Do not interrupt
1	1	Receiver is ready, interrupt	FIFO is full, interrupt

26.3.11 UART Baud Rate Generator Registers (UBG1*n*/UBG2*n*)

The UBG1*n* registers hold the MSB, and the UBG2*n* registers hold the LSB of the preload value. UBG1*n* and UBG2*n* concatenate to provide a divider to the internal bus clock for transmitter/receiver operation, as described in Section 26.4.1.2.1, "Internal Bus Clock Baud Rates."

IPSBAR 0x00_0218 (UBG10) Access: User write-only
Offset: 0x00_0258 (UBG11)
0x00_0298 (UBG12)

	7	6	5	4	3	2	1	0
R								
W	Divider MSB							
Reset:	0	0	0	0	0	0	0	0

Figure 26-13. UART Baud Rate Generator Registers (UBG1*n*)

IPSBAR 0x00_021C (UBG20) Access: User write-only
Offset: 0x00_025C (UBG21)
0x00_029C (UBG22)

	7	6	5	4	3	2	1	0
R								
W	Divider LSB							
Reset:	0	0	0	0	0	0	0	0

Figure 26-14. UART Baud Rate Generator Registers (UBG2*n*)

NOTE

The minimum value loaded on the concatenation of UBG1*n* with UBG2*n* is 0x0002. The UBG2*n* reset value of 0x00 is invalid and must be written to before the UART transmitter or receiver are enabled. UBG1*n* and UBG2*n* are write-only and cannot be read by the CPU.

26.3.12 UART Input Port Register (UIP*n*)

The UIP*n* registers show the current state of the $\overline{\text{UCTS}n}$ input.

IPSBAR 0x00_0234 (UIP0) Access: User read-only
Offset: 0x00_0274 (UIP1)
0x00_02B4 (UIP2)

	7	6	5	4	3	2	1	0
R	1	1	1	1	1	1	1	CTS
W								
Reset:	1	1	1	1	1	1	1	1

Figure 26-15. UART Input Port Registers (UIP*n*)

Table 26-11. UIP*n* Field Descriptions

Field	Description
7–1	Reserved
0 CTS	Current state of clear-to-send. The $\overline{\text{UCTS}n}$ value is latched and reflects the state of the input pin when UIP*n* is read. **Note:** This bit has the same function and value as UIPCR*n*[CTS]. 0 The current state of the $\overline{\text{UCTS}n}$ input is logic 0. 1 The current state of the $\overline{\text{UCTS}n}$ input is logic 1.

26.3.13 UART Output Port Command Registers (UOP1*n*/UOP0*n*)

The $\overline{\text{URTS}n}$ output can be asserted by writing a 1 to UOP1*n*[RTS] and negated by writing a 1 to UOP0*n*[RTS].

IPSBAR 0x00_0238 (UOP10) Access: User write-only
Offset: 0x00_023C (UOP00)
0x00_0278 (UOP11)
0x00_027C (UOP01)
0x00_02B8 (UOP12)
0x00_02BC (UOP02)

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	RTS
Reset:	0	0	0	0	0	0	0	0

Figure 26-16. UART Output Port Command Registers (UOP1*n*/UOP0*n*)

Table 26-12. UOP1*n*/UOP0*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{\text{URTS}n}$ output. 0 Not affected. 1 Asserts $\overline{\text{URTS}n}$ in UOP1. Negates $\overline{\text{URTS}n}$ in UOP0.

26.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

26.4.1 Transmitter/Receiver Clock Source

The internal bus clock serves as the basic timing reference for the clock source generator logic, which consists of a clock generator and a programmable 16-bit divider dedicated to each UART. The 16-bit divider is used to produce standard UART baud rates.

26.4.1.1 Programmable Divider

As Figure 26-17 shows, the UART*n* transmitter and receiver can use the following clock sources:

- An external clock signal on the DTIN*n* pin. When not divided, DTIN*n* provides a synchronous clock; when divided by 16, it is asynchronous.
- The internal bus clock supplies an asynchronous clock source divided by 32 and then divided by the 16-bit value programmed in UBG1*n* and UBG2*n*. See Section 26.3.11, "UART Baud Rate Generator Registers (UBG1n/UBG2n)."

The choice of DTIN or internal bus clock is programmed in the UCSR.

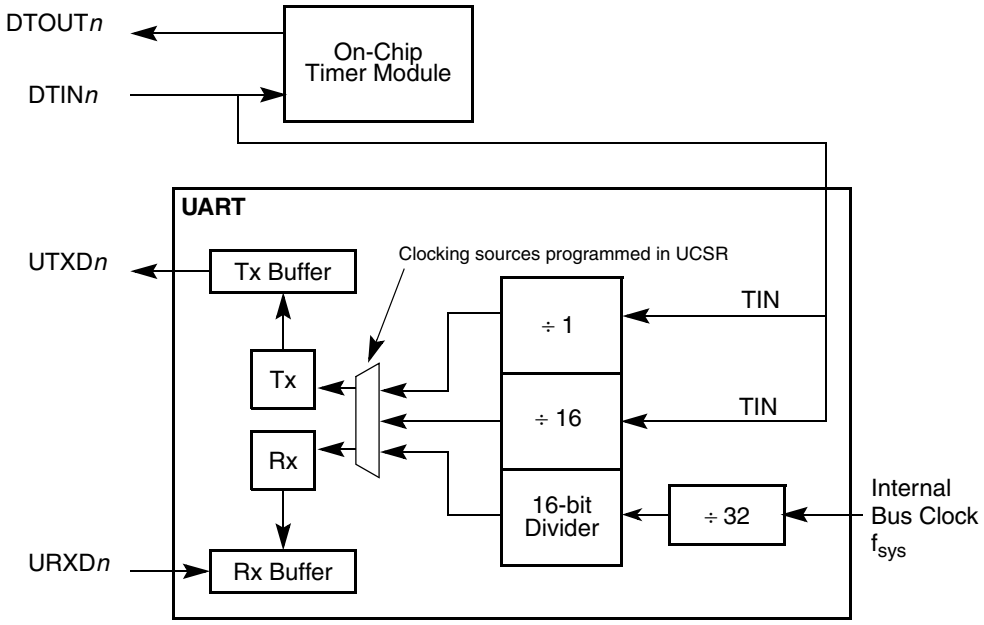


Figure 26-17. Clocking Source Diagram

NOTE

If DTIN*n* is a clocking source for the timer or UART, that timer module cannot use DTIN*n* for timer input capture.

26.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

26.4.1.2.1 Internal Bus Clock Baud Rates

When the internal bus clock is the UART clocking source, it goes through a divide-by-32 prescaler and then passes through the 16-bit divider of the concatenated UBG1*n* and UBG2*n* registers. The baud-rate calculation is:

$$\text{Baudrate} = \frac{f_{sys}}{[32 \times \text{Divider}]} \qquad \textit{Eqn. 26-1}$$

Using a 60-MHz internal bus clock and letting baud rate equal 9600, then

$$\text{Divider} = \frac{60 \times 10^6 \text{ Hz}}{[32 \times 9600 \text{ Hz}]} = 195 \text{ (decimal)} = 0\text{x}00\text{C}3 \text{ (hexadecimal)} \qquad \textit{Eqn. 26-2}$$

Therefore, UBG1*n* equals 0x00 and UBG2*n* equals 0xC3.

26.4.1.2.2 External Clock

An external source clock (DTIN*n*) passes through a divide-by-1 or 16 prescaler. If f_{extc} is the external clock frequency, baud rate can be described with this equation:

$$\text{Baudrate} = \frac{f_{extc}}{(16 \text{ or } 1)} \qquad \textit{Eqn. 26-3}$$

26.4.2 Transmitter and Receiver Operating Modes

Figure 26-18 is a functional block diagram of the transmitter and receiver showing the command and operating registers, which are described generally in the following sections. For detailed descriptions, refer to Section 26.3, "Memory Map/Register Definition."

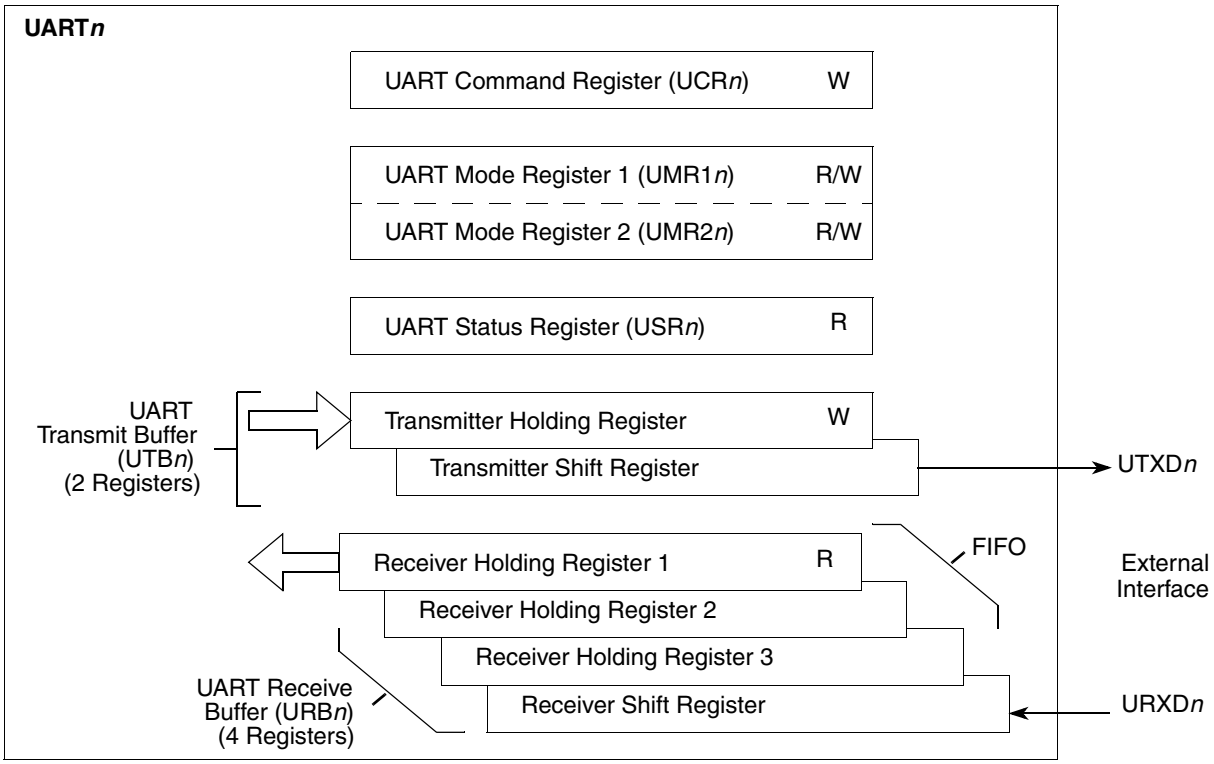


Figure 26-18. Transmitter and Receiver Functional Diagram

26.4.2.1 Transmitter

The transmitter is enabled through the UART command register (UCR*n*). When it is ready to accept a character, UART sets USR*n*[TXRDY]. The transmitter converts parallel data from the CPU to a serial bit stream on UTXD*n*. It automatically sends a start bit followed by the programmed number of data bits, an

optional parity bit, and the programmed number of stop bits. The lsb is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

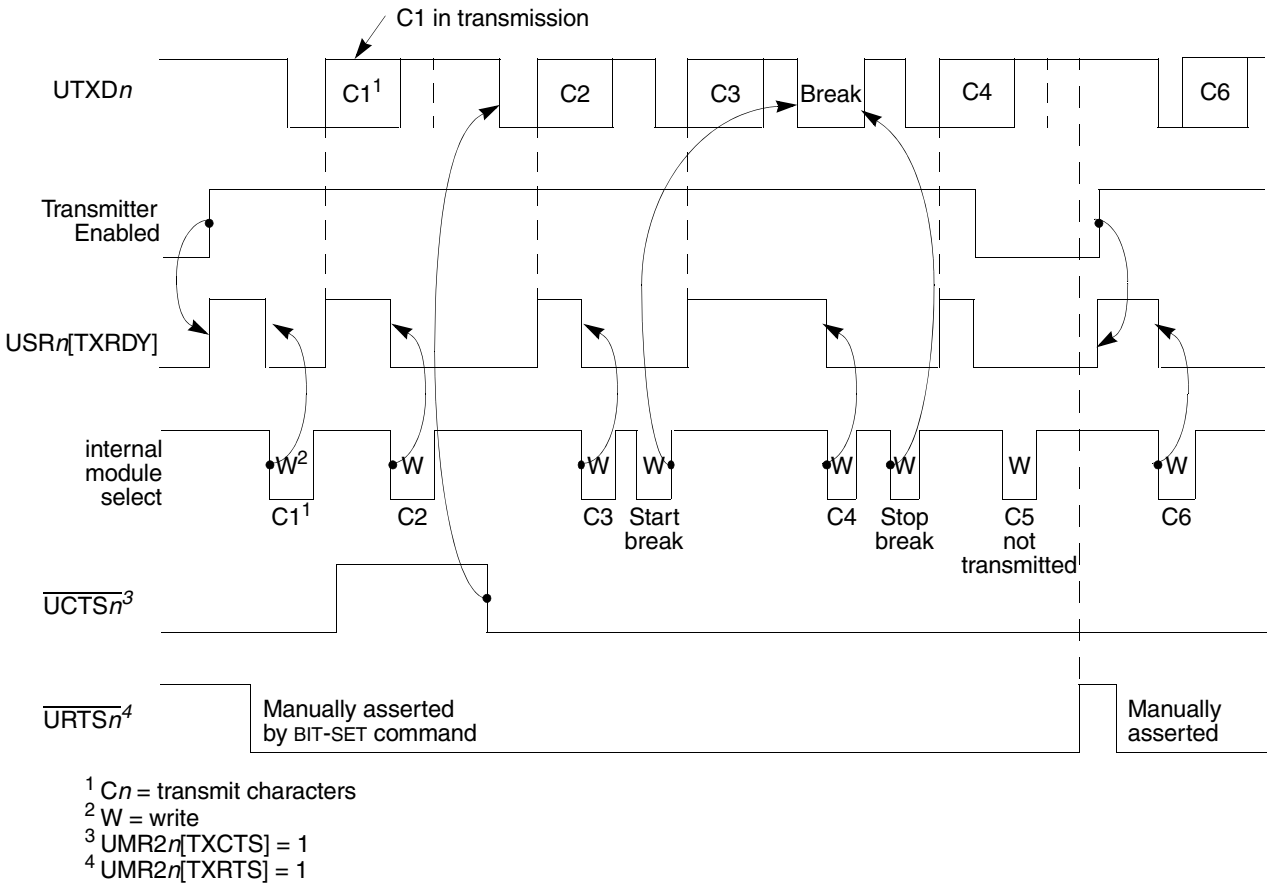
After the stop bits are sent, if no new character is in the transmitter holding register, the UTXD*n* output remains high (mark condition) and the transmitter empty bit (USR*n*[TXEMP]) is set. Transmission resumes and TXEMP is cleared when the CPU loads a new character into the UART transmit buffer (UTB*n*). If the transmitter receives a disable command, it continues until any character in the transmitter shift register is completely sent.

If the transmitter is reset through a software command, operation stops immediately (see Section 26.3.5, "UART Command Registers (UCRn)"). The transmitter is reenabled through the UCR*n* to resume operation after a disable or software reset.

If the clear-to-send operation is enabled, $\overline{\text{UCTS}}$*n* must be asserted for the character to be transmitted. If $\overline{\text{UCTS}}$*n* is negated in the middle of a transmission, the character in the shift register is sent and UTXD*n* remains in mark state until $\overline{\text{UCTS}}$*n* is reasserted. If transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, transmitter ignores the state of $\overline{\text{UCTS}}$*n*.

If the transmitter is programmed to automatically negate $\overline{\text{URTS}}$*n* when a message transmission completes, $\overline{\text{URTS}}$*n* must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{URTS}}$*n* is appropriately programmed, $\overline{\text{URTS}}$*n* is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting $\overline{\text{URTS}}$*n* before the next message is sent.

Figure 26-19 shows the functional timing information for the transmitter.



[1] C*n* = transmit characters
[2] W = write
[3] UMR2*n*[TXCTS] = 1
[4] UMR2*n*[TXRTS] = 1

Figure 26-19. Transmitter Timing Diagram

26.4.2.2 Receiver

The receiver is enabled through its UCR*n*, as described in Section 26.3.5, "UART Command Registers (UCRn)."

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on URXD*n*, the state of URXD*n* is sampled eight times on the edge of the bit time clock starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If URXD*n* is sampled high, start bit is invalid and the search for the valid start bit begins again.

If URXD*n* remains low, a valid start bit is assumed. The receiver continues sampling the input at one-bit time intervals at the theoretical center of the bit until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the URXD*n* input is sampled on the rising edge of the programmed clock source. The lsb is received first. The data then transfers to a receiver holding register and USR*n*[RXRDY] is set. If the character is less than 8 bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and URXD*n* remains low for one-half of the bit period after the stop bit is sampled, receiver operates as if a new start bit were detected. Parity error,

framing error, overrun error, and received break conditions set the respective PE, FE, OE, and RB error and break flags in the USR*n* at the received character boundary. They are valid only if USR*n*[RXRDY] is set.

If a break condition is detected (URXD*n* is low for the entire character including the stop bit), a character of all 0s loads into the receiver holding register and USR*n*[RB,RXRDY] are set. URXD*n* must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. The receiver places the damaged character in the Rx FIFO and sets the corresponding USR*n* error bits and USR*n*[RXRDY]. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets USR*n*[RB,RXRDY].

Figure 26-20 shows receiver functional timing.

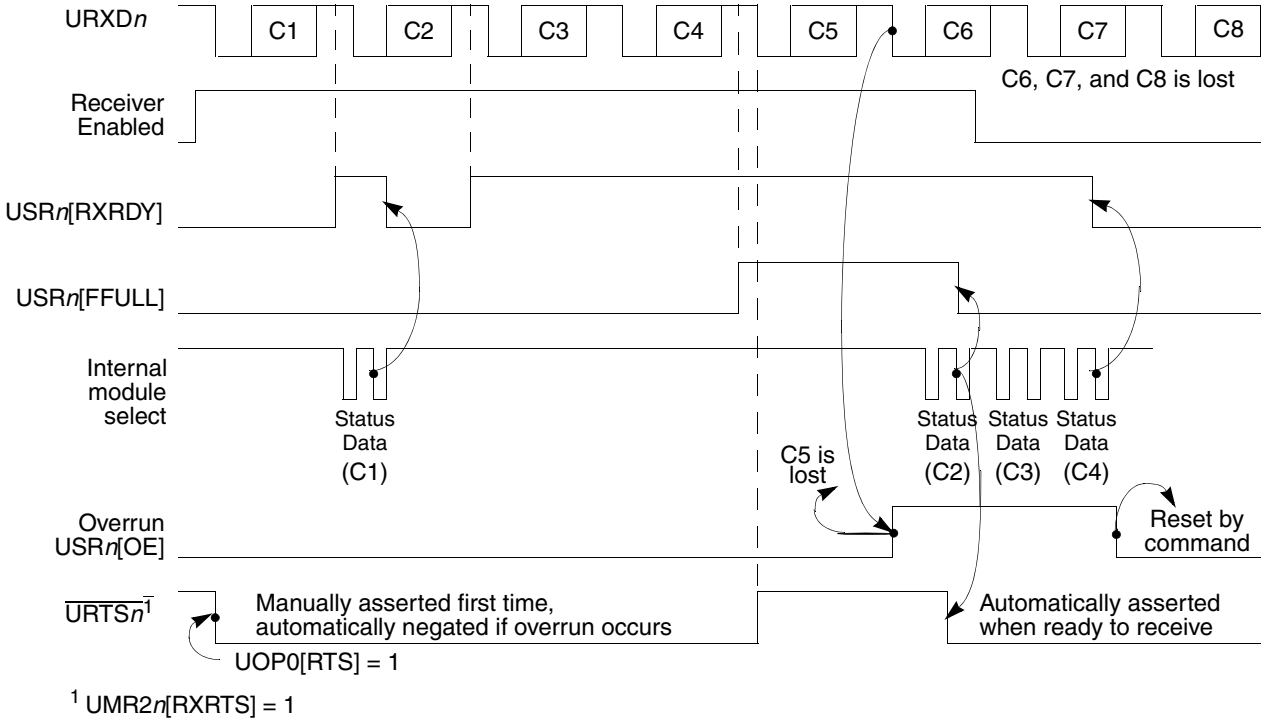


Figure 26-20. Receiver Timing Diagram

26.4.2.3 FIFO

The FIFO is used in the UART's receive buffer logic. The FIFO consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the URXD*n* (see Figure 26-18). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Therefore, data flowing from the receiver to the CPU is quadruple-buffered.

In addition to the data byte, three status bits—parity error (PE), framing error (FE), and received break (RB)—are appended to each data character in the FIFO; overrun error (OE) is not appended. By

programming the ERR bit in the UART's mode register (UMR1*n*), status is provided in character or block modes.

USR*n*[RXRDY] is set when at least one character is available to be read by the CPU. A read of the receive buffer produces an output of data from the top of the FIFO. After the read cycle, the data at the top of the FIFO and its associated status bits are popped and the receiver shift register can add new data at the bottom of the FIFO. The FIFO-full status bit (FFULL) is set if all three positions are filled with data. The RXRDY or FFULL bit can be selected to cause an interrupt and TXRDY or RXRDY can be used to generate a DMA request.

The two error modes are selected by UMR1*n*[ERR]:

- In character mode (UMR1*n*[ERR] = 0), status is given in the USR*n* for the character at the top of the FIFO.
- In block mode, the USR*n* shows a logical OR of all characters reaching the top of the FIFO since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the USR*n* does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USR*n* should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the character in the receiver shift register is lost, the FIFOs are unaffected, and USR*n*[OE] is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert $\overline{\text{URTS}}n$, in which case the receiver automatically negates $\overline{\text{URTS}}n$ when a valid start bit is detected and the FIFO is full. The receiver asserts $\overline{\text{URTS}}n$ when a FIFO position becomes available; therefore, connecting $\overline{\text{URTS}}n$ to the $\overline{\text{UCTS}}n$ input of the transmitting device can prevent overrun errors.

NOTE

The receiver continues reading characters in the FIFO if the receiver is disabled. If the receiver is reset, the FIFO, $\overline{\text{URTS}}n$ control, all receiver status bits, interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

26.4.3 Looping Modes

The UART can be configured to operate in various looping modes. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs and in Section 26.3, "Memory Map/Register Definition."

The UART's transmitter and receiver should be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

26.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in Figure 26-21, the UART automatically resends received data bit by bit. The local CPU-to-receiver communication continues normally, but the CPU-to-transmitter link is disabled. In this mode, received data is clocked on the receiver clock and re-sent on UTXD*n*. The receiver must be enabled, but the transmitter need not be.

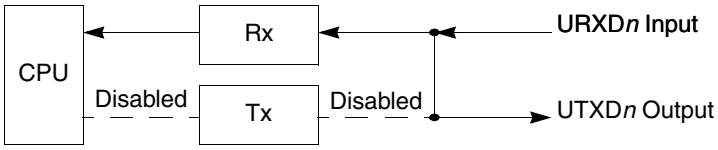


Figure 26-21. Automatic Echo

Because the transmitter is inactive, USR*n*[TXEMP,TXRDY] is inactive and data is sent as it is received. Received parity is checked but not recalculated for transmission. Character framing is also checked, but stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected.

26.4.3.2 Local Loopback Mode

Figure 26-22 shows how UTXD*n* and URXD*n* are internally connected in local loopback mode. This mode is for testing the operation of a UART by sending data to the transmitter and checking data assembled by the receiver to ensure proper operations.

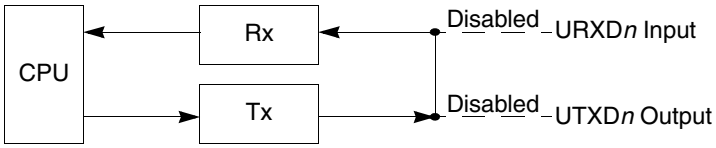


Figure 26-22. Local Loopback

Features of this local loopback mode are:

- Transmitter and CPU-to-receiver communications continue normally in this mode.
- URXD*n* input data is ignored.
- UTXD*n* is held marking.
- The receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be.

26.4.3.3 Remote Loopback Mode

In remote loopback mode, shown in Figure 26-23, the UART automatically transmits received data bit by bit on the UTXD*n* output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote UART. For this mode, transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and all status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are sent as they are received. A received break is echoed as received until next valid start bit is detected.

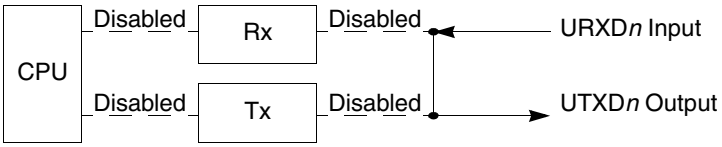


Figure 26-23. Remote Loopback

26.4.4 Multidrop Mode

Setting UMR1*n*[PM] programs the UART to operate in a wake-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver notifies its respective CPU by setting USR*n*[RXRDY] and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Unaddressed slave stations continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process. Functional timing information for multidrop mode is shown in Figure 26-24.

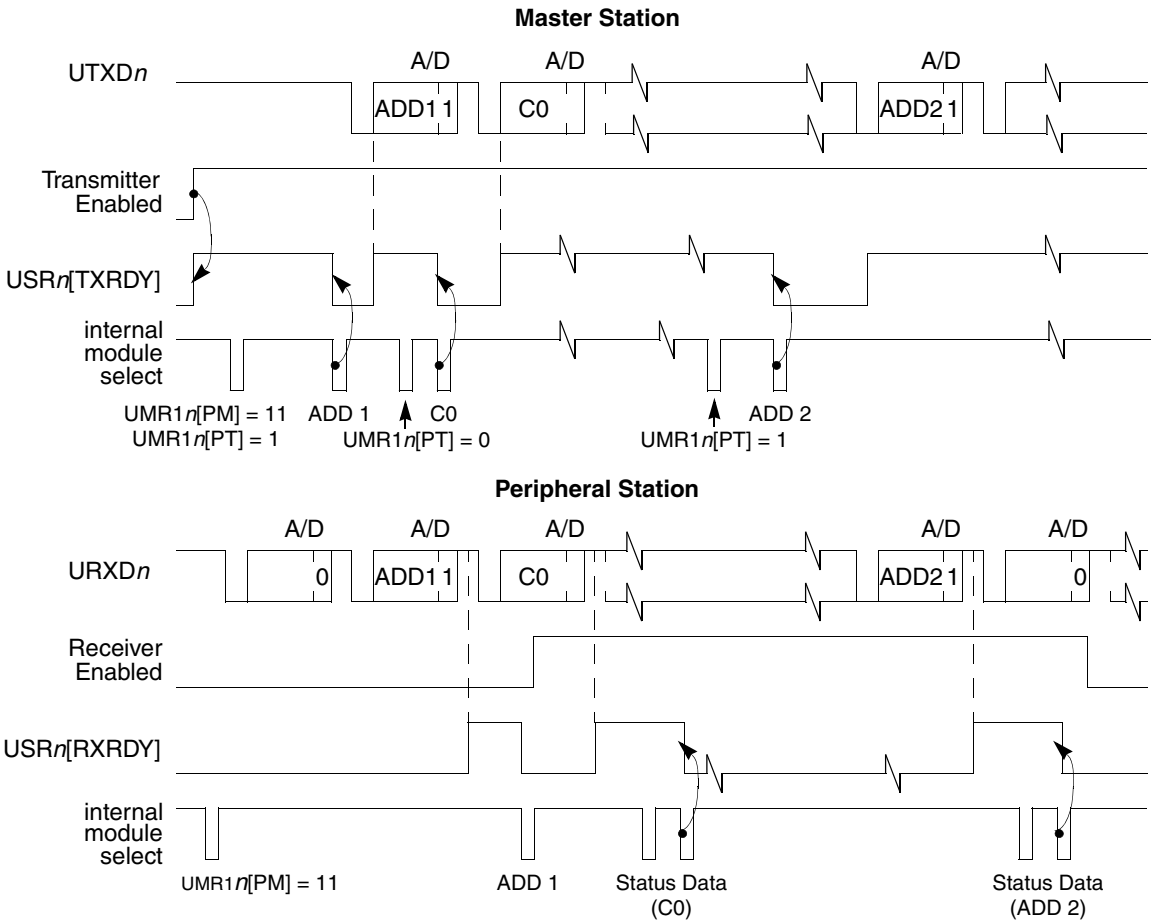


Figure 26-24. Multidrop Mode Timing Diagram

A character sent from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. A/D equals 1 indicates an address character; A/D equals 0 indicates a data character. The polarity of A/D is selected through UMR1*n*[PT]. UMR1*n* should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RXRDY bit and loads the character into the receiver holding register FIFO provided the received A/D bit is a 1 (address tag). The character is discarded if the received A/D bit is 0 (data tag). If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register during read operations.

In either case, data bits load into the data portion of the FIFO while the A/D bit loads into the status portion of the FIFO normally used for a parity error (USR*n*[PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may continues containing error detection and correction information. If 8-bit characters are not required, one way to provide error detection is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

26.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

26.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

26.4.5.2 Write Cycles

The UART module accepts write data as bytes only. Write cycles to read-only or reserved registers complete normally without an error termination, but data is ignored.

26.5 Initialization/Application Information

The software flowchart, Figure 26-25, consists of:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 26-30 and Sheet 2 p. 26-31). Before SINIT is called at system initialization, the calling routine allocates 2 words on the system FIFO. On return to the calling routine, SINIT passes UART status data on the FIFO. If SINIT finds no errors, the transmitter and receiver are enabled. SINIT calls CHCHK to perform the checks. When called, SINIT places the UART in local loopback mode and checks for the following errors:
  - Transmitter never ready
  - Receiver never ready
  - Parity error
  - Incorrect character received
- I/O driver routine—This routine (See Sheet 4 p. 26-33 and Sheet 5 p. 26-34) consists of INCH, the terminal input character routine which gets a character from the receiver, and OUTCH, which sends a character to the transmitter.
- Interrupt handling—This consists of SIRQ (See Sheet 4 p. 26-33), which is executed after the UART module generates an interrupt caused by a change-in-break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

26.5.1 Interrupt and DMA Request Initialization

26.5.1.1 Setting up the UART to Generate Core Interrupts

The list below provides steps to properly initialize the UART to generate an interrupt request to the processor's interrupt controller. See Section 15.3.8.1, "Interrupt Sources," for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICR*x* register in the interrupt controller.
2. Unmask appropriate bits in IMR in the interrupt controller.

3. Unmask appropriate bits in the core's status register (SR) to enable interrupts.
4. If TXRDY or RXRDY generates interrupt requests, verify that DMAREQC (in the SCM) does not also assign the UART's TXRDY and RXRDY into DMA channels.
5. Initialize interrupts in the UART, see Table 26-13.

Table 26-13. UART Interrupts

Register	Bit	Interrupt
UMR1*n*	6	RxIRQ
UIMR*n*	7	Change of State (COS)
UIMR*n*	2	Delta Break
UIMR*n*	1	RxFIFO Full
UIMR*n*	0	TXRDY

26.5.1.2 Setting up the UART to Request DMA Service

The UART is capable of generating two internal DMA request signals: transmit and receive.

The transmit DMA request signal is asserted when the TXRDY (transmitter ready) in the UART interrupt status register (UISR*n*[TXRDY]) is set. When the transmit DMA request signal is asserted, the DMA can initiate a data copy, reading the next character transmitted from memory and writing it into the UART transmit buffer (UTB*n*). This allows the DMA channel to stream data from memory to the UART for transmission without processor intervention. After the entire message has been moved into the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) could query the UART programming model to determine the end-of-transmission status.

Similarly, the receive DMA request signal is asserted when the FIFO full or receive ready (FFULL/RXRDY) flag in the interrupt status register (UISR*n*[FFULL/RXRDY]) is set. When the receive DMA request signal is asserted, the DMA can initiate a data move, reading the appropriate characters from the UART receive buffer (URB*n*) and storing them in memory. This allows the DMA channel to stream data from the UART receive buffer into memory without processor intervention. After the entire message has been moved from the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) should query the UART programming model to determine the end-of-transmission status. In typical applications, the receive DMA request should be configured to use RXRDY directly (and not FFULL) to remove any complications related to retrieving the final characters from the FIFO buffer.

The implementation described in this section allows independent DMA processing of transmit and receive data while continuing to support interrupt notification to the processor for $\overline{\text{CTS}}$ change-of-state and delta break error managing.

To configure the UART for DMA requests:

1. Initialize the DMAREQC in the SCM to map the desired UART DMA requests to the desired DMA channels. For example, setting DMAREQC[7:4] to 1000 maps UART0 receive DMA requests to DMA channel 1, setting DMAREQC[11:8] to 1101 maps UART1 transmit DMA requests to DMA channel 2, and so on. It is possible to independently map transmit-based and receive-based UART DMA requests in the DMAREQC.
2. Disable interrupts using the UIMR register. The appropriate UIMR bits must be cleared so that interrupt requests are disabled for those conditions for which a DMA request is desired. For example, to generate transmit DMA requests from UART1, UIMR1[TXRDY] should be cleared. This prevents TXRDY from generating an interrupt request while a transmit DMA request is generated.
3. Enable DMA access to the UART*n* registers by setting the corresponding PACR register in the SCM for read/write in supervisor and user modes.
4. Enable DMA access to SRAM by setting the SPV bit in the core RAMBAR, and the BDE bit in the SCM RAMBAR
5. Initialize the DMA channel. The DMA should be configured for cycle steal mode and a source and destination size of one byte. This causes a single byte to be transferred for each UART DMA request. Set the disable request bit (DCR*n*[D_REQ] to disable external requests when the BCR reaches zero.
6. For a transmit process:
   — Set the DMA SAR register to the address of the source data
   — Set DCR*n*[SINC] to increment the source pointer
   — Set DAR to the address if the UART transmit buffer (UTB)
   — Clear DCR*n*[DINC]
   — Set BCR to the number of bytes to transmit.
7. For a receive process:
   — Set the DMA SAR register to the address of the UART receive buffer (URB)
   — Clear DCR*n*[SINC]
   — Set DAR to the address of the source data
   — Set DCR*n*[DINC] to increment the destination pointer
   — Set BCR to the number of bytes to transmit.
8. Start the data transfer by setting DCR*n*[EEXT], which enables the UART channel to issue DMA requests.

Table 26-14 shows the DMA requests.

Table 26-14. UART DMA Requests

Register	Bit	DMA Request
UISR*n*	1	Receive DMA request
UISR*n*	0	Transmit DMA request

26.5.2 UART Module Initialization Sequence

The following shows the UART module initialization sequence.

1. UCR*n*:
   a) Reset the receiver and transmitter.
   b) Reset the mode pointer (MISC[2–0] = 0b001).
2. UIMR*n*: Enable the desired interrupt sources.
3. UACR*n*: Initialize the input enable control (IEC bit).
4. UCSR*n*: Select the receiver and transmitter clock. Use timer as source if required.
5. UMR1*n*:
   a) If preferred, program operation of receiver ready-to-send (RXRTS bit).
   a) Select receiver-ready or FIFO-full notification (RXRDY/FFULL bit).
   b) Select character or block error mode (ERR bit).
   c) Select parity mode and type (PM and PT bits).
   d) Select number of bits per character (B/Cx bits).
6. UMR2*n*:
   a) Select the mode of operation (CM bits).
   b) If preferred, program operation of transmitter ready-to-send (TXRTS).
   c) If preferred, program operation of clear-to-send (TXCTS bit).
   d) Select stop-bit length (SB bits).
7. UCR*n*: Enable transmitter and/or receiver.

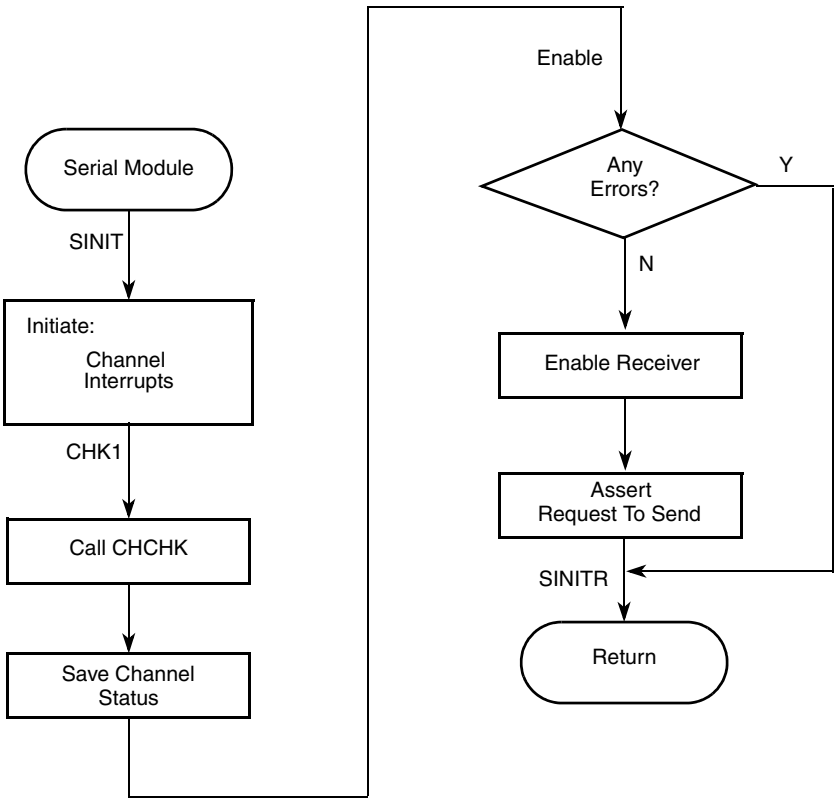


Figure 26-25. UART Mode Programming Flowchart (Sheet 1 of 5)

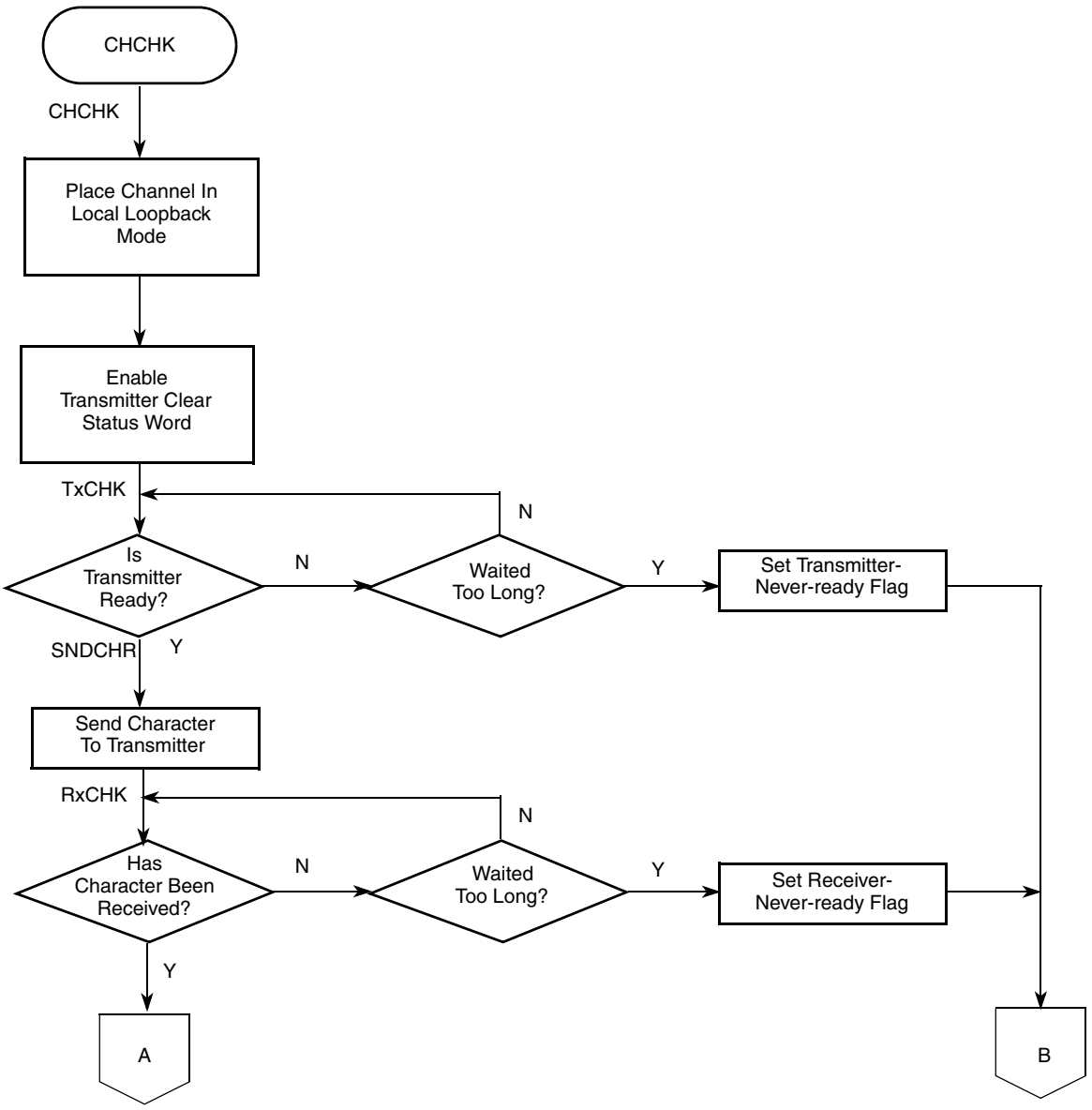


Figure 26-25. UART Mode Programming Flowchart (Sheet 2 of 5)

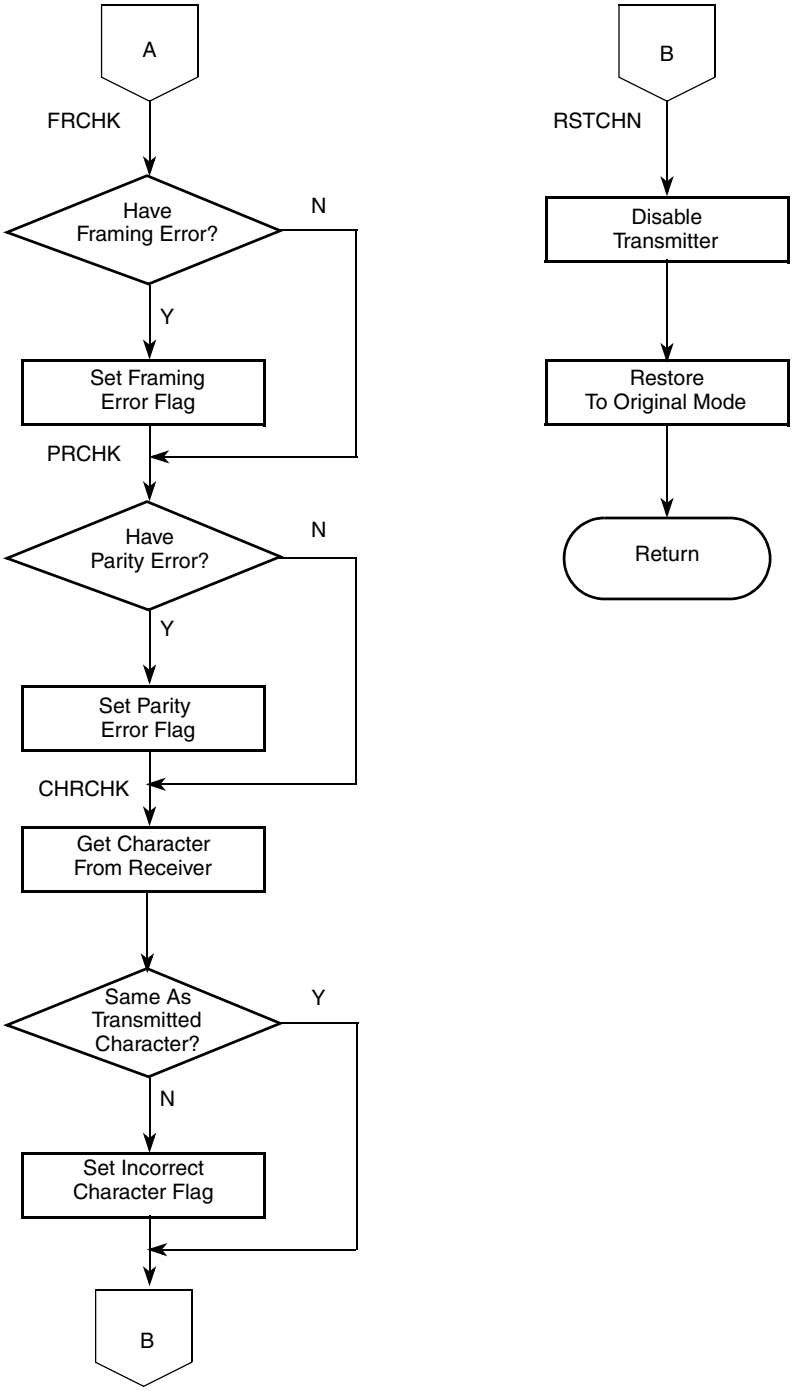


Figure 26-25. UART Mode Programming Flowchart (Sheet 3 of 5)

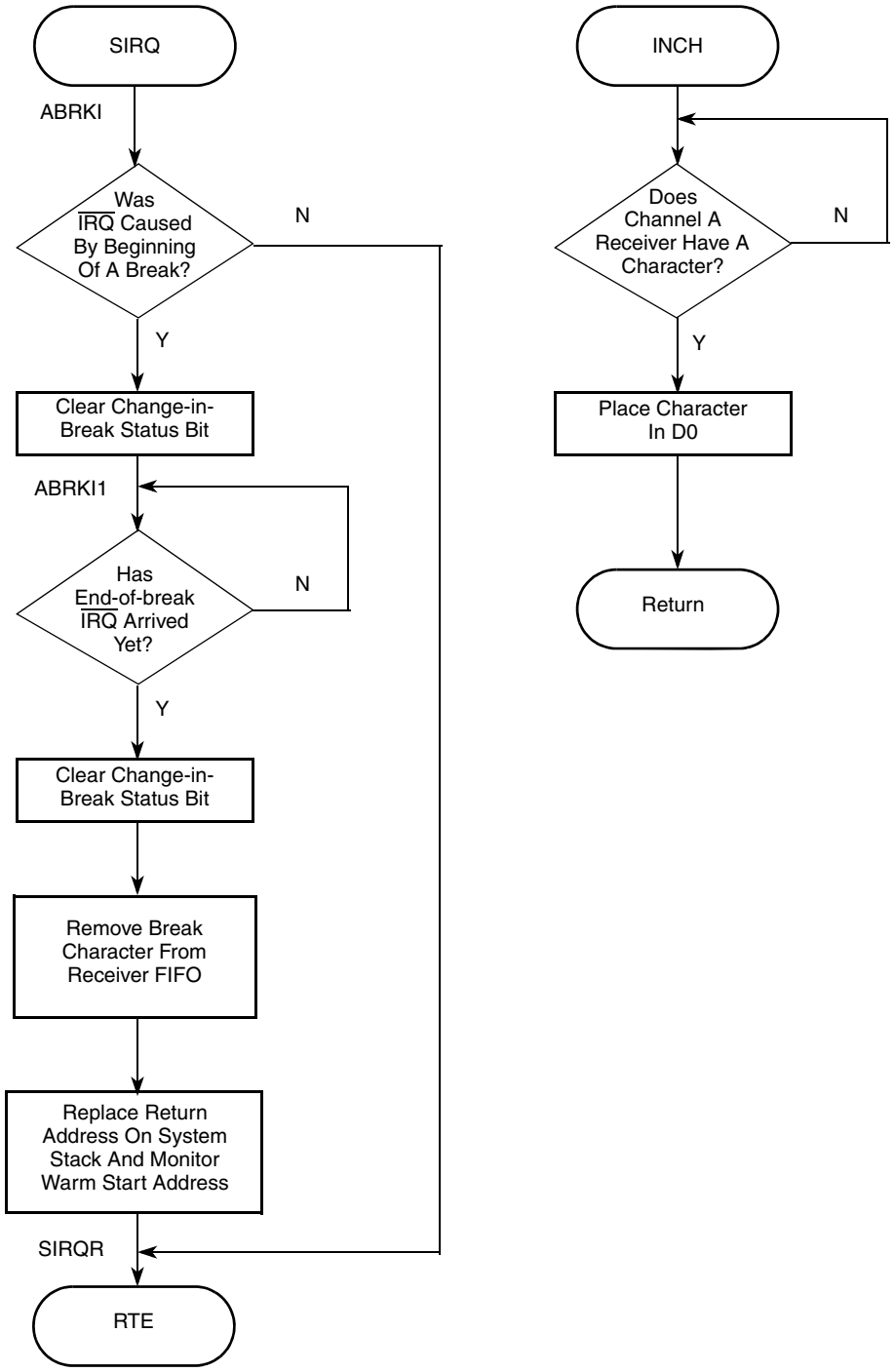


Figure 26-25. UART Mode Programming Flowchart (Sheet 4 of 5)

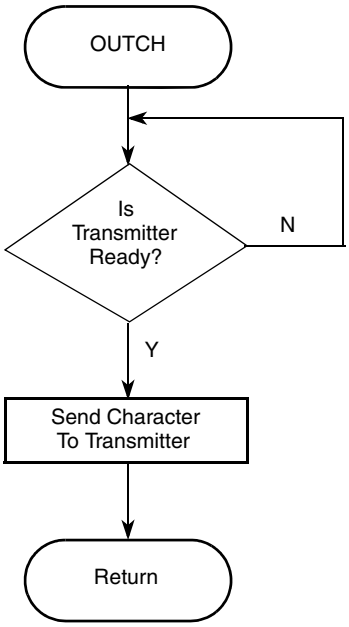


Figure 26-25. UART Mode Programming Flowchart (Sheet 5 of 5)

Chapter 27
I²C Interface

27.1 Introduction

This chapter describes the I²C module, clock synchronization, and I²C programming model registers. It also provides extensive programming examples.

27.1.1 Block Diagram

Figure 27-1 is a I²C module block diagram, illustrating the interaction of the registers described in Section 27.2, “Memory Map/Register Definition”.

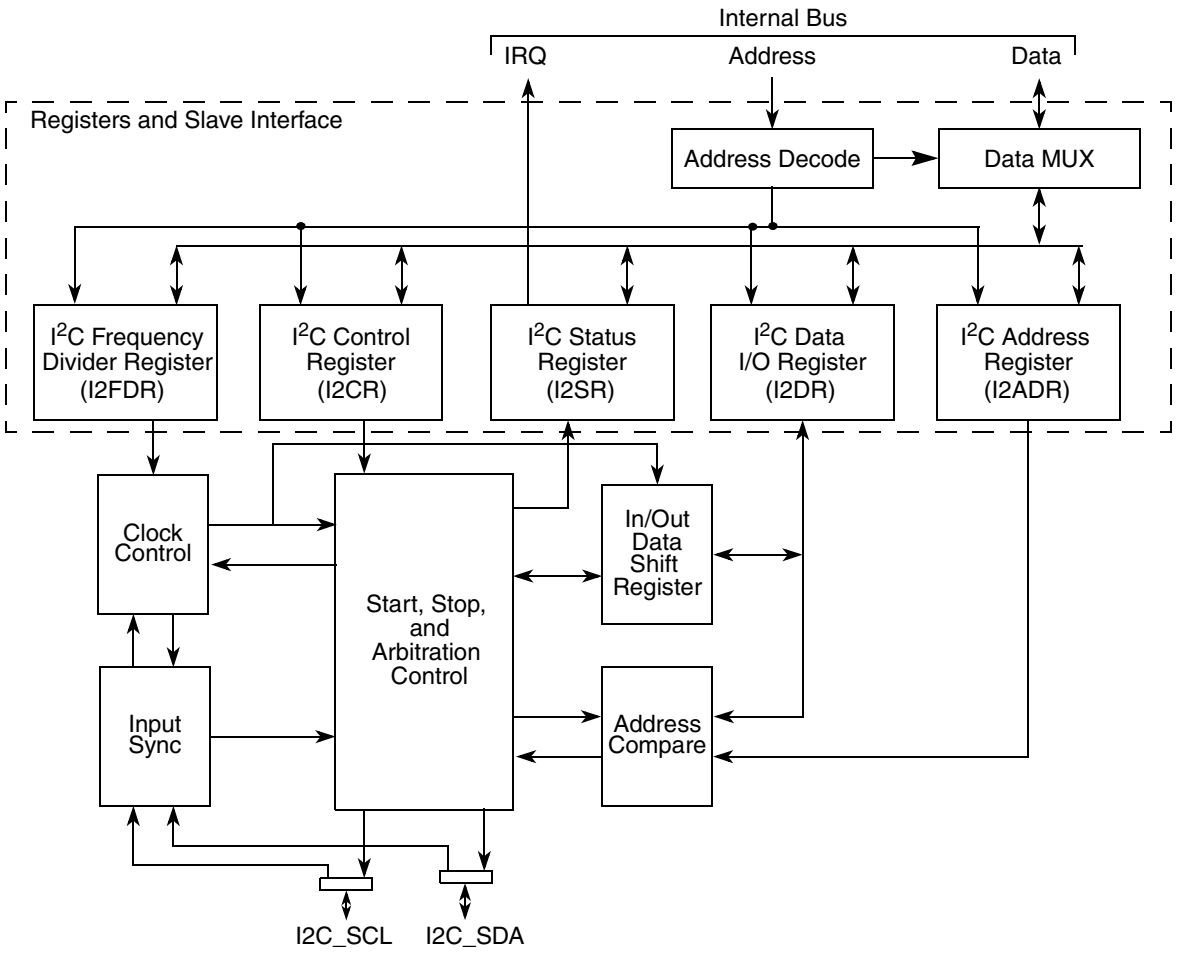


Figure 27-1. I²C Module Block Diagram

27.1.2 Overview

I²C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications that require occasional communication between many devices over a short distance. The flexible I²C bus allows additional devices to connect to the bus for expansion and system development.

The interface operates up to 100 Kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of the internal bus clock divided by 20, with reduced bus loading. The maximum communication length and the number of devices connected are limited by a maximum bus capacitance of 400 pF.

The I²C system is a true multiple-master bus; it uses arbitration and collision detection to prevent data corruption in the event that multiple devices attempt to control the bus simultaneously. This supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

NOTE

The I²C module is compatible with the Philips I²C bus protocol. For information on system configuration, protocol, and restrictions, see *The I²C Bus Specification, Version 2.1*.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “General Purpose I/O Module”) prior to configuring the I²C module.

27.1.3 Features

The I²C module has these key features:

- Compatibility with I²C bus standard version 2.1
- Multiple-master operation
- Software-programmable for one of 50 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

27.2 Memory Map/Register Definition

The below table lists the configuration registers used in the I²C interface.

Table 27-1. I²C Module Memory Map

IPSBAR Offset	Register	Access	Reset Value	Section/Page
0x00_0300	I²C Address Register (I2ADR)	R/W	0x00	27.2.1/27-3
0x00_0304	I²C Frequency Divider Register (I2FDR)	R/W	0x00	27.2.2/27-3
0x00_0308	I²C Control Register (I2CR)	R/W	0x00	27.2.3/27-4
0x00_030C	I²C Status Register (I2SR)	R/W	0x81	27.2.4/27-6
0x00_0310	I²C Data I/O Register (I2DR)	R/W	0x00	27.2.5/27-7

27.2.1 I²C Address Register (I2ADR)

I2ADR holds the address the I²C responds to when addressed as a slave. It is not the address sent on the bus during the address transfer when the module is performing a master transfer.

IPSBAR Offset: 0x00_0300 (I2ADR) Access: User read/write

	7	6	5	4	3	2	1	0
R	ADR							0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 27-2. I²C Address Register (I2ADR)

Table 27-2. I2ADR Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I²C module. Slave mode is the default I²C mode for an address match on the bus.
0	Reserved, must be cleared.

27.2.2 I²C Frequency Divider Register (I2FDR)

The I2FDR, shown in Figure 27-3, provides a programmable prescaler to configure the I²C clock for bit-rate selection.

IPSBAR Offset: 0x00_0304 (I2FDR) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	IC					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 27-3. I²C Frequency Divider Register (I2FDR)

Table 27-3. I2FDR Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 IC	I²C clock rate. Prescales the clock for bit-rate selection. The serial bit clock frequency is equal to the internal bus clock divided by the divider shown below. Due to potentially slow I2C_SCL and I2C_SDA rise and fall times, bus signals are sampled at the prescaler frequency.

IC	Divider	IC	Divider	IC	Divider	IC	Divider
0x00	28	0x10	288	0x20	20	0x30	160
0x01	30	0x11	320	0x21	22	0x31	192
0x02	34	0x12	384	0x22	24	0x32	224
0x03	40	0x13	480	0x23	26	0x33	256
0x04	44	0x14	576	0x24	28	0x34	320
0x05	48	0x15	640	0x25	32	0x35	384
0x06	56	0x16	768	0x26	36	0x36	448
0x07	68	0x17	960	0x27	40	0x37	512
0x08	80	0x18	1152	0x28	48	0x38	640
0x09	88	0x19	1280	0x29	56	0x39	768
0x0A	104	0x1A	1536	0x2A	64	0x3A	896
0x0B	128	0x1B	1920	0x2B	72	0x3B	1024
0x0C	144	0x1C	2304	0x2C	80	0x3C	1280
0x0D	160	0x1D	2560	0x2D	96	0x3D	1536
0x0E	192	0x1E	3072	0x2E	112	0x3E	1792
0x0F	240	0x1F	3840	0x2F	128	0x3F	2048

27.2.3 I²C Control Register (I2CR)

I2CR enables the I²C module and the I²C interrupt. It also contains bits that govern operation as a slave or a master.

IPSBAR Offset: 0x00_0308 (I2CR) Access: User read/write

	7	6	5	4	3	2	1	0
R	IEN	IIEN	MSTA	MTX	TXAK	RSTA	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 27-4. I²C Control Register (I2CR)

Table 27-4. I2CR Field Descriptions

Field	Description
7 IEN	I²C enable. Controls the software reset of the entire I²C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next START condition is detected. Master mode is not aware that the bus is busy; initiating a start cycle may corrupt the current bus cycle, ultimately causing the current master or the I²C module to lose arbitration, after which bus operation returns to normal. 0 The I²C module is disabled, but registers can be accessed. 1 The I²C module is enabled. This bit must be set before any other I2CR bits have any effect.
6 IIEN	I²C interrupt enable. 0 I²C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 I²C module interrupts are enabled. An I²C interrupt occurs if I2SR[IIF] is also set.
5 MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4 MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When the device is addressed as a slave, software must set MTX according to I2SR[SRW]. In master mode, MTX must be set according to the type of transfer required. Therefore, when the MCU addresses a slave device, MTX is always 1.
3 TXAK	Transmit acknowledge enable. Specifies the value driven onto I2C_SDA during acknowledge cycles for master and slave receivers. Writing TXAK applies only when the I²C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (acknowledge bit = 1).
2 RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1–0	Reserved, must be cleared.

27.2.4 I²C Status Register (I2SR)

I2SR contains bits that indicate transaction direction and status.

IPSBAR 0x00_030C (I2SR) Access: User read/write
Offset:

	7	6	5	4	3	2	1	0
R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK
W								
Reset:	1	0	0	0	0	0	0	1

Figure 27-5. I²C Status Register (I2SR)

Table 27-5. I2SR Field Descriptions

Field	Description
7 ICF	I²C Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by falling edge of ninth clock of a byte transfer.
6 IAAS	I²C addressed as a slave bit. The CPU is interrupted if I2CR[IIEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CR clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5 IBB	I²C bus busy bit. Indicates the status of the bus. 0 Bus is idle. If a STOP signal is detected, IBB is cleared. 1 Bus is busy. When START is detected, IBB is set.
4 IAL	I²C arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.) • I2C_SDA sampled low when the master drives high during an address or data-transmit cycle. • I2C_SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it.
3	Reserved, must be cleared.
2 SRW	Slave read/write. When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I²C module is a slave and has an address match. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1 IIF	I²C interrupt. Must be cleared by software by writing a 0 in the interrupt routine. 0 No I²C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1). Set when one of the following occurs: • Complete one byte transfer (set at the falling edge of the ninth clock) • Reception of a calling address that matches its own specific address in slave-receive mode • Arbitration lost
0 RXAK	Received acknowledge. The value of I2C_SDA during the acknowledge bit of a bus cycle. 0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus 1 No acknowledge signal was detected at the ninth clock.

27.2.5 I²C Data I/O Register (I2DR)

In master-receive mode, reading I2DR allows a read to occur and for the next data byte to be received. In slave mode, the same function is available after the I²C has received its slave address.

IPSBAR Offset: 0x00_0310 (I2DR) Access: User read/write

	7	6	5	4	3	2	1	0
R W	DATA							
Reset:	0	0	0	0	0	0	0	0

Figure 27-6. I²C Data I/O Register (I2DR)

Table 27-6. I2DR Field Description

Field	Description
7–0 DATA	I²C data. When data is written to this register in master transmit mode, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates the reception of the next byte of data. In slave mode, the same functions are available after an address match has occurred. **Note:** In master transmit mode, the first byte of data written to I2DR following assertion of I2CR[MSTA] is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/W bit (in position D0). This bit (D0) is not automatically appended by the hardware, software must provide the appropriate R/W bit. **Note:** I2CR[MSTA] generates a start when a master does not already own the bus. I2CR[RSTA] generates a start (restart) without the master first issuing a stop (i.e., the master already owns the bus). To start the read of data, a dummy read to this register starts the read process from the slave. The next read of the I2DR register contains the actual data.

27.3 Functional Description

The I²C module uses a serial data line (I2C_SDA) and a serial clock line (I2C_SCL) for data transfer. For I²C compliance, all devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I²C default state is as a slave receiver. Therefore, when not programmed to be a master or responding to a slave transmit address, the I²C module should return to the default slave receiver state. See Section 27.4.1, “Initialization Sequence,” for exceptions.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. These are discussed in the following sections.

27.3.1 START Signal

When no other device is bus master (I2C_SCL and I2C_SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in Figure 27-7). A START signal is defined as a high-to-low transition of I2C_SDA while I2C_SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

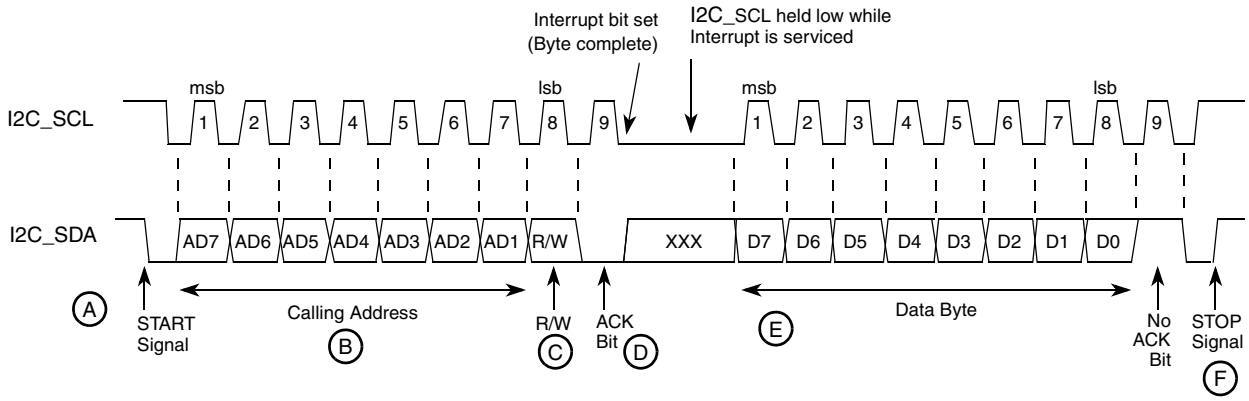


Figure 27-7. I²C Standard Communication Protocol

27.3.2 Slave Address Transmission

The master sends the slave address in the first byte after the START signal (B). After the seven-bit calling address, it sends the R/W bit (C), which tells the slave data transfer direction (0 equals write transfer, 1 equals read transfer).

Each slave must have a unique address. An I²C master must not transmit its own slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls I2C_SDA low at the ninth serial clock (D) to return an acknowledge bit.

27.3.3 Data Transfer

When successful slave addressing is achieved, data transfer can proceed (see E in Figure 27-7) on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

Data can be changed only while I2C_SCL is low and must be held stable while I2C_SCL is high, as Figure 27-7 shows. I2C_SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling I2C_SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses. See Figure 27-8.

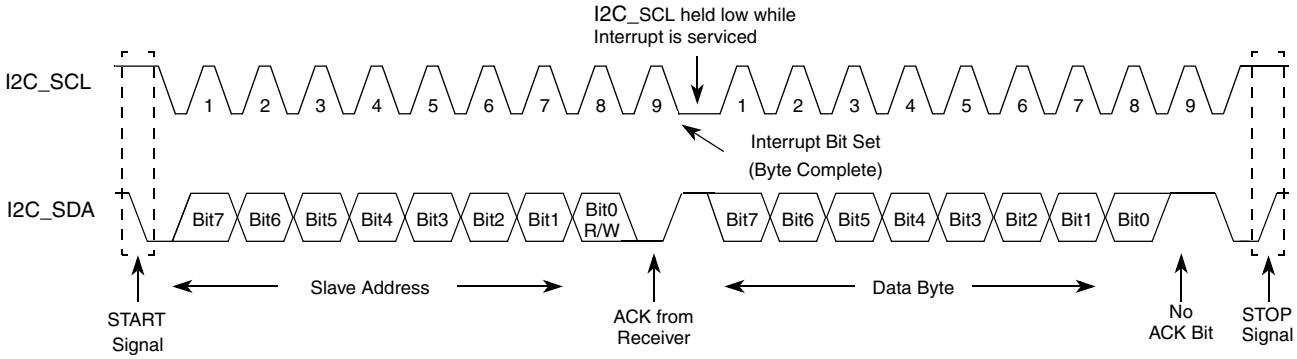


Figure 27-8. Data Transfer

27.3.4 Acknowledge

The transmitter releases the I2C_SDA line high during the acknowledge clock pulse as shown in Figure 27-9. The receiver pulls down the I2C_SDA line during the acknowledge clock pulse so that it remains stable low during the high period of the clock pulse.

If it does not acknowledge the master, the slave receiver must leave I2C_SDA high. The master can then generate a STOP signal to abort data transfer or generate a START signal (repeated start, shown in Figure 27-10 and discussed in Section 27.3.6, “Repeated START”) to start a new calling sequence.

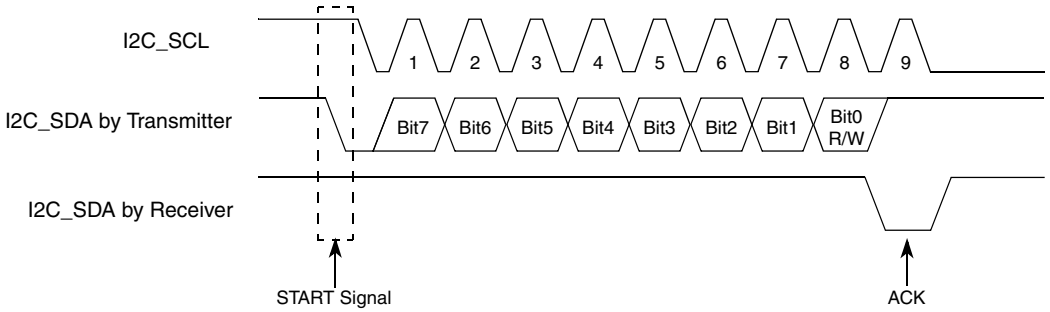


Figure 27-9. Acknowledgement by Receiver

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases I2C_SDA for the master to generate a STOP or START signal (Figure 27-9).

27.3.5 STOP Signal

The master can terminate communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of I2C_SDA while I2C_SCL is at logical high (see F in Figure 27-7). The master can generate a STOP even if the slave has generated an acknowledgment, at which point the slave must release the bus. The master may also generate a START signal following a calling address, without first generating a STOP signal. Refer to Section 27.3.6, “Repeated START.”

27.3.6 Repeated START

A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication, as shown in Figure 27-10. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

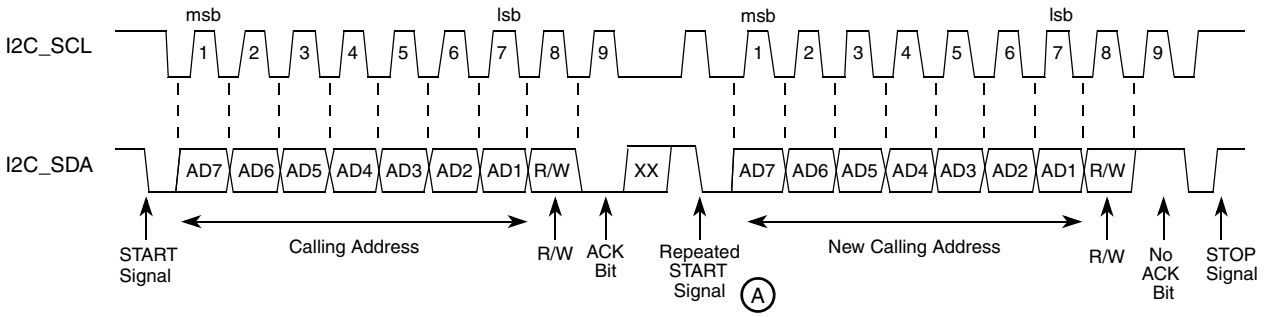


Figure 27-10. Repeated START

Various combinations of read/write formats are then possible:

- The first example in Figure 27-11 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.
- The second example in Figure 27-11 is the master reading the slave immediately after the first byte. At the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes slave-transmitter.
- In the third example in Figure 27-11, START condition and slave address are repeated using the repeated START signal. This is to communicate with same slave in a different mode without releasing the bus. The master transmits data to the slave first, and then the master reads data from slave by reversing the R/$\overline{W}$ bit.

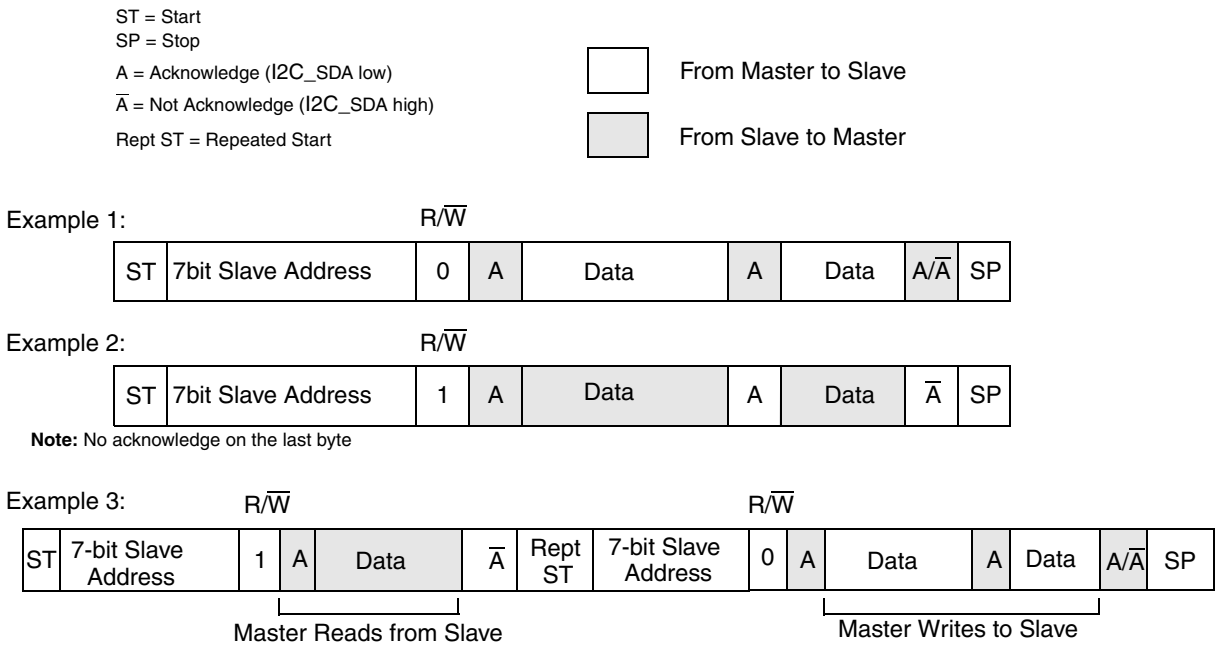


Figure 27-11. Data Transfer, Combined Format

27.3.7 Clock Synchronization and Arbitration

I2C is a true multi-master bus that allows more than one master connected to it. If two or more master devices simultaneously request control of the bus, a clock synchronization procedure determines the bus clock. Because wire-AND logic is performed on the I2C_SCL line, a high-to-low transition on the I2C_SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the I2C_SCL line low until the clock high state is reached. However, change of low to high in this device's clock may not change the state of the I2C_SCL line if another device clock remains within its low period. Therefore, synchronized clock I2C_SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time (see Figure 27-12). When all devices concerned have counted off their low period, the synchronized clock (I2C_SCL) line is released and pulled high. At this point, the device clocks and the I2C_SCL line are synchronized, and the devices start counting their high periods. The first device to complete its high period pulls the I2C_SCL line low again.

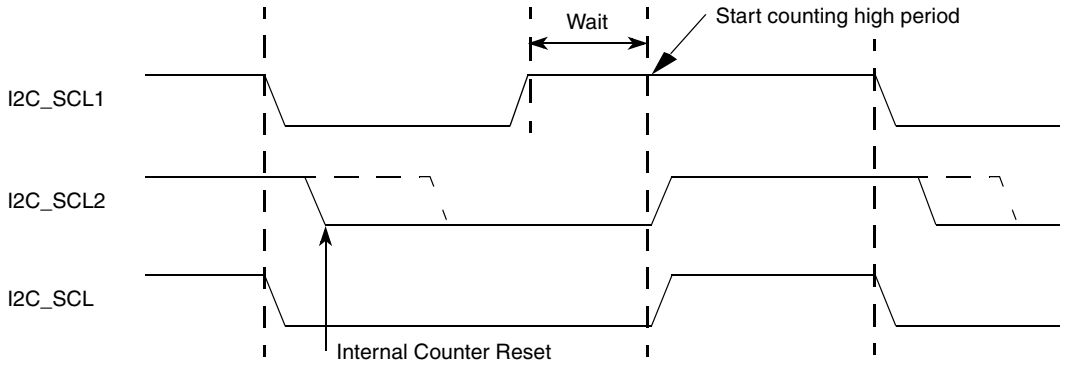


Figure 27-12. Clock Synchronization

A data arbitration procedure determines the relative priority of the contending masters. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving I2C_SDA output (see Figure 27-13). In this case, transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.

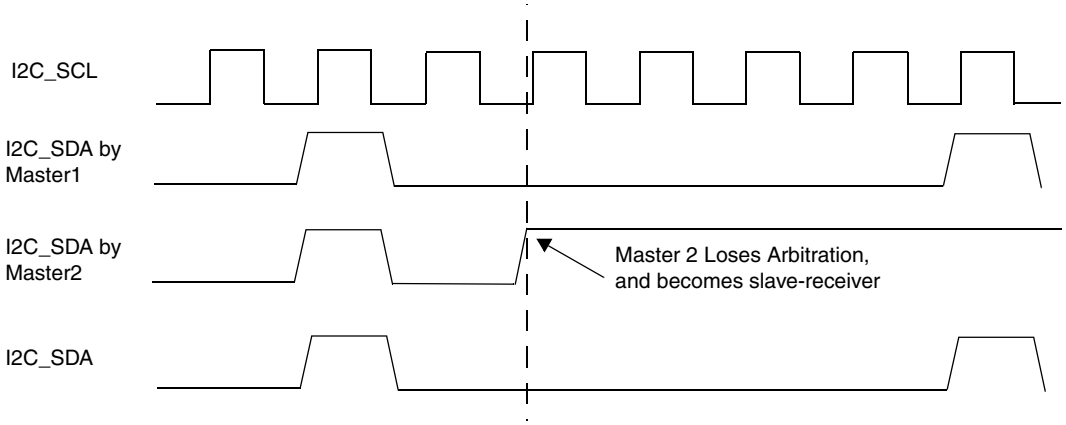


Figure 27-13. Arbitration Procedure

27.3.8 Handshaking and Clock Stretching

The clock synchronization mechanism can acts as a handshake in data transfers. Slave devices can hold I2C_SCL low after completing one byte transfer. In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases I2C_SCL.

Slaves may also slow down the transfer bit rate. After the master has driven I2C_SCL low, the slave can drive I2C_SCL low for the required period and then release it. If the slave I2C_SCL low period is longer than the master I2C_SCL low period, the resulting I2C_SCL bus signal low period is stretched.

27.4 Initialization/Application Information

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

27.4.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized:

1. Set I2FDR[IC] to obtain I2C_SCL frequency from the system bus clock. See Section 27.2.2, “I2C Frequency Divider Register (I2FDR).”
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I²C bus interface system.
4. Modify the I2CR to select or deselect master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE

If I2SR[IBB] is set when the I²C bus module is enabled, execute the following pseudocode sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were power-cycled on.

```
I2CR = 0x0
I2CR = 0xA0
dummy read of I2DR
I2SR = 0x0
I2CR = 0x0
I2CR = 0x80       ; re-enable
```

27.4.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. On a multiple-master bus system, I2SR[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB is cleared), the START signal and the first byte (the slave address) can be sent. The data written to the data register comprises the address of the desired slave and the lsb indicates the transfer direction.

The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the I2C_SCL period, the

processor may need to wait until the I2C is busy after writing the calling address to the I2DR before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

1. Check I2SR[IBB]. If it is set, wait until it is clear.
2. After cleared, set to transmit mode by setting I2CR[MTX].
3. Set master mode by setting I2CR[MSTA]. This generates a START condition.
4. Transmit the calling address via the I2DR.
5. Check I2SR[IBB]. If it is clear, wait until it is set and go to step #1.

27.4.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IIEN]. Software must first clear I2SR[IIF] in the interrupt routine. Reading from I2DR in receive mode or writing to I2DR in transmit mode can clear I2SR[ICF].

Software can service the I²C I/O in the main program by monitoring the IIF bit if the interrupt function is disabled. Polling should monitor IIF rather than ICF, because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; the address is sent. If master receive mode is required, I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see Figure 27-14).

1. Clear the I2CR[IIF] flag.
2. Check if acknowledge has been received, I2SR[RXAK].
3. If no ACK, end transmission. Else, transmit next byte of data via I2DR.

27.4.4 Generation of STOP

A data transfer ends when the master signals a STOP, which can occur after all data is sent, as in the following example.

1. Check if acknowledge has been received, I2SR[RXAK]. If no ACK, end transmission and go to step #5.
2. Get value from transmitting counter, TXCNT. If no more data, go to step #5.
3. Transmit next byte of data via I2DR.
4. Decrement TXCNT and go to step #1
5. Generate a stop condition by clearing I2CR[MSTA].

For a master receiver to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last data byte. This is done by setting I2CR[TXAK] before reading the next-to-last byte. Before the last byte is read, a STOP signal must be generated, as in the following example.

1. Decrement RXCNT.
2. If last byte (RXCNT = 0) go to step #4.
3. If next to last byte (RXCNT = 1), set I2CR[TXAK] to disable ACK and go to step #5.
4. This is last byte, so clear I2CR[MSTA] to generate a STOP signal.
5. Read data from I2DR.
6. If there is more data to be read (RXCNT ≠ 0), go to step #1 if desired.

27.4.5 Generation of Repeated START

If the master wants the bus after the data transfer, it can signal another START followed by another slave address without signaling a STOP, as in the following example.

1. Generate a repeated START by setting I2CR[RSTA].
2. Transmit the calling address via I2DR.

27.4.6 Slave Mode

In the slave interrupt service routine, software must poll the I2SR[IAAS] bit to determine if the controller has received its slave address. If IAAS is set, software must set the transmit/receive mode select bit (I2CR[MTX]) according to the I2SR[SRW]. Writing to I2CR clears IAAS automatically. The only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers have IAAS cleared. A data transfer can now be initiated by writing information to I2DR for slave transmits, or read from I2DR in slave-receive mode. A dummy read of I2DR in slave/receive mode releases I2C_SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[RXAK] must be tested before sending the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which software must switch it from transmitter to receiver mode. Reading I2DR releases I2C_SCL so the master can generate a STOP signal.

27.4.7 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes master. Hardware immediately switches devices that lose arbitration to slave receive mode. Data output to I2C_SDA stops, but I2C_SCL continues generating until the end of the byte during which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR[IAL] set and I2CR[MSTA] cleared.

If a non-master device tries to transmit or execute a START, hardware inhibits the transmission, clears MSTA without signaling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed attempt to engage the bus. When considering these cases, slave service routine should first test IAL and software should clear it if it is set.

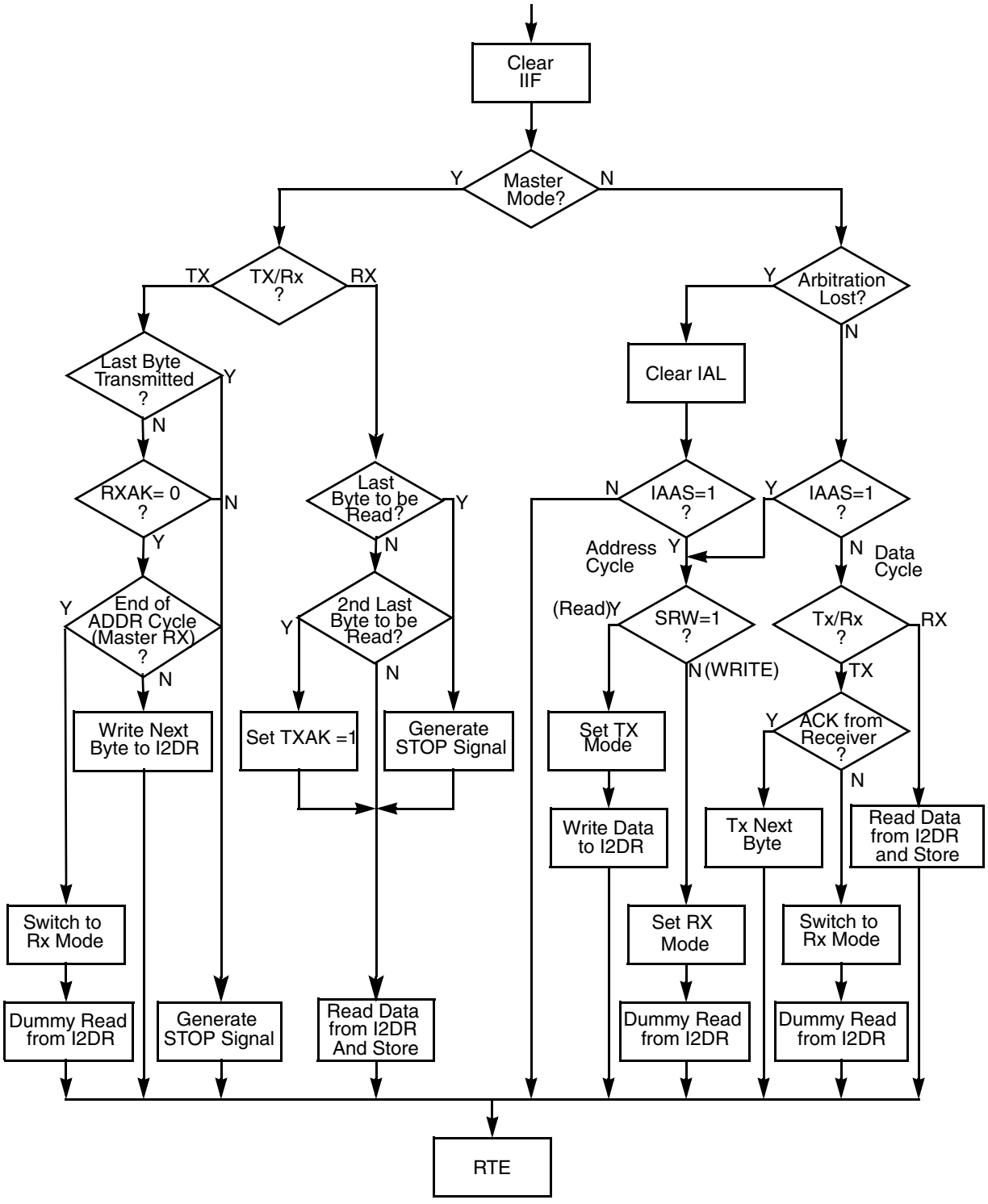


Figure 27-14. Flow-Chart of Typical I²C Interrupt Routine

Chapter 28
Analog-to-Digital Converter (ADC)

28.1 Introduction

The analog-to-digital converter (ADC) consists of two separate and complete ADCs, each with their own sample and hold circuits. The converters share a common voltage reference and common digital control module.

NOTE

The MCF52235 does not have the 8 MHz internal oscillator feature and all references to it should be ignored and all control and status bits should also not be used.

28.2 Features

The ADC's characteristics include the following:

- 12-bit resolution
- Maximum ADC clock frequency of 5.0 MHz, 200 ns period
- Sampling rate up to 1.66 million samples per second[1]
- Single conversion time of 8.5 ADC clock cycles (8.5 × 200 ns = 1.7 μs)
- Additional conversion time of 6 ADC clock cycles (6 × 200 ns = 1.2 μs)
- Eight conversions in 26.5 ADC clocks (26.5 × 200 ns = 5.3 μs) using simultaneous mode
- Ability to simultaneously sample and hold 2 inputs
- Ability to sequentially scan and store up to 8 measurements
- Internal multiplex to select two of 8 inputs
- Power savings modes allow automatic shutdown/startup of all or part of ADC
- Those inputs not selected tolerate injected/sourced current without affecting ADC performance, supporting operation in noisy industrial environments.
- Optional interrupts at the end of a scan, if an out-of-range limit is exceeded (high or low), or at zero crossing
- Optional sample correction by subtracting a pre-programmed offset value
- Signed or unsigned result
- Single ended or differential inputs for all input pins with support for an arbitrary mix of input types

1. In loop mode, the time between each conversion is 6 ADC clock cycles (1.2 μs at 5.0 MHz). Using simultaneous conversion, two samples are captured in 1.2 μs, providing an overall sample rate of 1.66 million samples per second.

28.3 Block Diagram

The ADC function, shown in **Figure 28-1,** consists of two four-channel input select functions, interfacing with two independent Sample and Hold (S/H) circuits, which feed two 12-bit ADCs. The two converters store their results in a buffer, awaiting further processing.

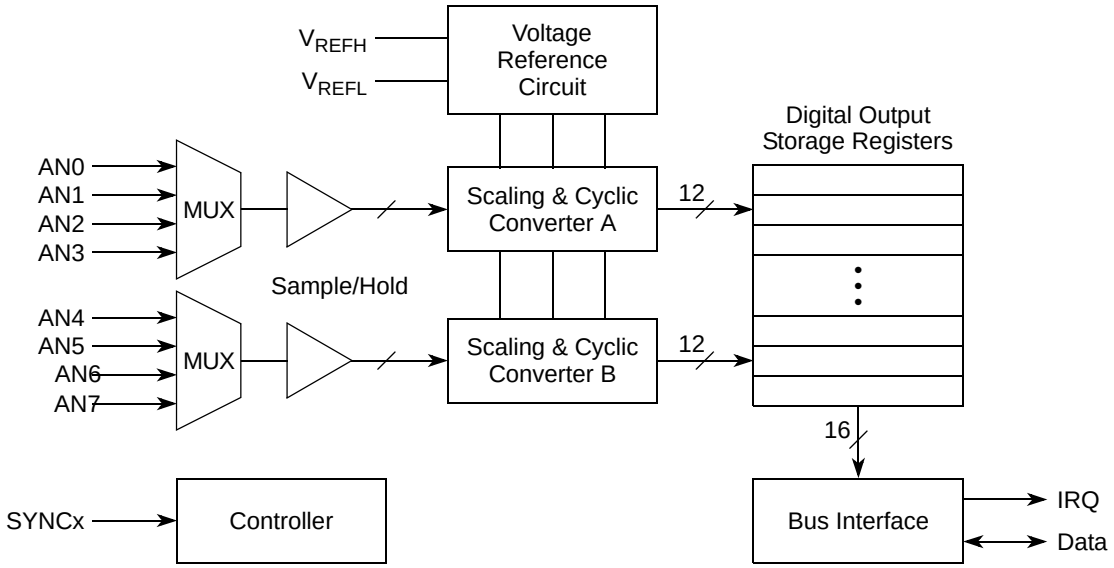


Figure 28-1. Dual ADC Block Diagram

28.4 Memory Map and Register Definition

This section presents the registers of the ADC module. A summary of these registers is given in Table 28-1. All ADC registers are supervisor-mode access only.

Table 28-1. ADC Register Summary

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x19_0000	Control Register 1 (CTRL1)	16	R/W	0x5005	28.4.1/28-3
0x19_0002	Control Register 2 (CTRL2)	16	R/W	0x0002	28.4.2/28-5
0x19_0004	Zero Crossing Control Register (ADZCC)	16	R/W	0x0000	28.4.3/28-8
0x19_0006	Channel List Register 1 (ADLST1)	16	R/W	0x3210	28.4.4/28-8
0x19_0008	Channel List Register 2 (ADLST2)	16	R/W	0x7654	28.4.4/28-8
0x19_000A	Sample Disable Register (ADSDIS)	16	R/W	0x0000	28.4.5/28-10
0x19_000C	Status Register (ADSTAT)	16	R/W	0x0000	28.4.6/28-11
0x19_000E	Limit Status Register (ADLSTAT)	16	R/W	0x0000	28.4.7/28-13
0x19_0010	Zero Crossing Status Register (ADZCSTAT)	16	R/W	0x0000	28.4.8/28-14
0x19_0012–20	Result Registers 0-7 (ADRSLT0-7)	16	R/W	0x0000	28.4.9/28-14
0x19_0022–30	Low Limit Registers 0-7 (ADLLMT0-7)	16	R/W	0x0000	28.4.10/28-15

Table 28-1. ADC Register Summary (continued)

IPSBAR Offset[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x19_0032–40	High Limit Registers 0-7 (ADHLMT0-7)	16	R/W	0x0000	28.4.10/28-15
0x19_0042–50	Offset Registers 0-7 (ADOFS0-7)	16	R/W	0x0000	28.4.11/28-17
0x19_0052	Power Control Register (POWER)	16	R/W	0x00D7	28.4.12/28-17
0x19_0054	Voltage Reference Register (CAL)	16	R/W	0x0000	28.4.13/28-20

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion.

28.4.1 Control 1 Register (CTRL1)

The CTRL1 register, shown in Figure 28-2, is used to configure and control the ADC module. The associated field descriptions are given in Table 28-2. Please see Section 28.5.6, "Scan Configuration and Control" for details on the functionality controlled by this register.

IPSBAR Access: read/write
Offset: 0x19_0000 (CTRL1)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	STOP0		SYNC0	EOSIE0	ZCIE	LLMTIE	HLMTIE	CHNCFG					SMODE		
W			START0													
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1

Figure 28-2. Control 1 Register (CTRL1)

Table 28-2. CTRL1 Field Descriptions

Field	Description
15	Reserved, should be cleared.
14 STOP0	Stop Conversion 0 bit. When STOP0 is set, the current scan is stopped and no further scans can start. Any further SYNC0 input pulses (see the SYNC0 field description) or writes to START0 are ignored until STOP0 is cleared. After the ADC is in stop mode, the result registers can be modified by the processor. Any changes to the result registers in stop mode are treated as if the analog core supplied the data. Therefore, limit checking, zero crossing, and associated interrupts can occur if enabled. 0 Normal operation 1 Stop mode **Note:** This is not the same as the device's STOP mode.
13 START0	Start Conversion 0 bit. A scan is started by writing a 1 to this bit. START0 is write-only. Writing 1 to the START0 bit again is ignored until the end of the current scan. The ADC must be in a stable power configuration prior to writing to START0 (see Section 28.5.8, "Power Management"). 0 No action 1 Start command is issued

Table 28-2. CTRL1 Field Descriptions (continued)

Field	Description
12 SYNC0	Synchronization 0 Enable bit. When this bit is set, a conversion may be initiated by asserting a positive edge on the SYNC0 input. Any subsequent SYNC0 input pulses that occur during the scan are ignored. In once sequential and once parallel scan modes, only the first SYNC0 input pulse is honored. Subsequent SYNC0 input pulses are ignored until SYNC0 input is re-armed by setting SYNC0. This can be done at any time, even during the execution of the scan. The ADC must be in a stable power configuration prior to writing to START0 (see Section 28.5.8, “Power Management”). 0 Scan is initiated by a write to the START0 bit only 1 Scan is initiated by a SYNC0 input pulse or a write to the START0 bit
11 EOSIE0	End of Scan Interrupt 0 Enable bit. This bit enables an EOSI0 interrupt to be generated upon completion of the scan. For looping scan modes, the interrupt triggers after the completion of each iteration of the loop. 0 Interrupt disabled 1 Interrupt enabled
10 ZCIE	Zero Crossing Interrupt Enable bit. This bit enables the zero crossing interrupt if the current result value has a sign change from the previous result as configured by the ADZCC register. 0 Interrupt disabled 1 Interrupt enabled
9 LLMTIE	Low Limit Interrupt Enable bit. This bit enables the low limit exceeded interrupt when the current result value is less than the low limit register value. The raw result value is compared to ADLLMT*n*[LLMT] before the offset register value is subtracted. 0 Interrupt disabled 1 Interrupt enabled
8 HLMTIE	High Limit Interrupt Enable bit. This bit enables the high limit exceeded interrupt if the current result value is greater than the high limit register value. The raw result value is compared to ADHLMT[HLMT] before the offset register value is subtracted. 0 Interrupt disabled 1 Interrupt enabled

Table 28-2. CTRL1 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–4
CHNCFG</td><td>Channel Configure. This field configures the inputs for single-ended or differential conversions:
<table>
<tr><th>CHNCFG</th><th>Inputs</th><th>Description</th></tr>
<tr><td>xxx1</td><td rowspan="2">AN0–AN1</td><td>Configured as differential pair (AN0 is + and AN1 is –)</td></tr>
<tr><td>xxx0</td><td>Both configured as single ended inputs</td></tr>
<tr><td>xx1x</td><td rowspan="2">AN2–AN3</td><td>Configured as differential pair (AN2 is + and AN3 is –)</td></tr>
<tr><td>xx0x</td><td>Both configured as single ended inputs</td></tr>
<tr><td>x1xx</td><td rowspan="2">AN4–AN5</td><td>Configured as differential pair (AN4 is + and AN5 is –)</td></tr>
<tr><td>x0xx</td><td>Both configured as single ended inputs</td></tr>
<tr><td>1xxx</td><td rowspan="2">AN6–AN7</td><td>Configured as differential pair (AN6 is + and AN7 is –)</td></tr>
<tr><td>0xxx</td><td>Both configured as single ended inputs</td></tr>
</table></td></tr>
<tr><td>2–0
SMODE</td><td>Scan Mode Control. This field controls the scan mode of the ADC module. See Section 28.5.6, “Scan Configuration and Control” for details on each scan mode.
000 Once sequential
001 Once parallel
010 Loop sequential
011 Loop parallel
100 Triggered sequential
101 Triggered parallel (default)
110 Reserved; do not use
111 Reserved; do not use</td></tr>
</table>

28.4.2 Control 2 Register (CTRL2)

The structure of the CTRL2 register depends on whether the ADC is operating in sequential or parallel mode (see Section 28.4.1, “Control 1 Register (CTRL1)”).

28.4.2.1 CTRL2 Under Sequential Scan Modes

IPSBAR Access: read/write

Offset: 0x19_0002 (CTRL2)

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

Figure 28-3. Control 2 Register (CTRL2) Under Sequential Scan Modes

Table 28-3. CTRL2 Field Descriptions Under Sequential Scan Modes

Field	Description
15–5	Reserved, should be cleared.
4–0 DIV	Clock Divisor Select. This field controls the divider circuit, which generates the ADC clock by dividing the system clock by 2×(DIV+1). DIV must be chosen so the ADC clock does not exceed 5.0 MHz. See Table 28-5 for a listing of ADC clock frequency based on the value of DIV for several configurations.

28.4.2.2 CTRL2 Under Parallel Scan Modes

When the ADC operates in a parallel scan mode, the CTRL2 register is used to control the operation of converter B. The interaction between converters A and B (and hence CTRL1 and CTRL2) is determined by the CTRL2[SIMULT] bit. By default, CTRL2[SIMULT] equals 1 and converter B operates together with converter A. In this case, the STOP1, START1, SYNC1, and EOSIE1 bits in the CTRL2 register do not affect converter B operation. If CTRL2[SIMULT] equals 0, these bits and the SYNC1 input are used to control the converter B scan. In this case, EOSIE1 enables the EOSI1 interrupt, signaling the end of a B converter scan. In addition, ADSTAT[CIP1] is used to indicate a converter B scan is active.

IPSBAR Access: read/write

Offset: 0x19_0002 (CTRL2)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	STOP1		SYNC1	EOSIE1	0	0	0	0	0	SIMULT	DIV				
W			START1													
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0

Figure 28-4. Control 2 Register (CTRL2) Under Parallel Scan Modes

Table 28-4. CTRL2 Field Descriptions Under Parallel Scan Modes

Field	Description
15	Reserved, should be cleared.
14 STOP1	Stop Conversion 1bit. In parallel-scan modes when SIMULT equaling 0, setting STOP1 stops parallel scans in the B converter and prevents new scans from starting. Any further SYNC1 input pulses (see the SYNC1 field description) or writes to START1 are ignored until STOP1 is cleared. After the ADC is in stop mode, the result registers can be modified by the processor. Any changes to the result registers in stop mode are treated as if the analog core supplied the data. Therefore, limit checking, zero crossing, and associated interrupts can occur if enabled. 0 Normal operation 1 Stop mode **Note:** This is not the same as the device's STOP mode.
13 START1	Start Conversion 1 bit. In parallel-scan modes when SIMULT equaling 0, a scan by the B converter is started by writing a 1 to this bit. START1 is write-only. Writing 1 to the START1 bit again is ignored until the end of the current scan. The ADC must be in a stable power configuration prior to writing to START1 (see Section 28.5.8, "Power Management"). 0 No action 1 Start command is issued

Table 28-4. CTRL2 Field Descriptions Under Parallel Scan Modes (continued)

Field	Description
12 SYNC1	Synchronization 1 Enable bit. In parallel-scan modes when SIMULT equaling 0, setting SYNC1 allows a conversion to be initiated by asserting a positive edge on the SYNC1 input. Any subsequent SYNC1 input pulses that occur during the scan are ignored. In once sequential and once parallel scan modes, only the first SYNC1 input pulse is honored. Subsequent SYNC1 input pulses are ignored until SYNC1 input is re-armed by setting SYNC1. This can be done at any time, even during the execution of the scan. The ADC must be in a stable power configuration prior to writing to START0 (see Section 28.5.8, “Power Management”). 0 Scan is initiated by a write to the START1 bit only 1 Scan is initiated by a SYNC1 input pulse or a write to the START1 bit
11 EOSIE1	End of Scan Interrupt 1 Enable bit. In parallel-scan modes when SIMULT equaling 0, this bit enables an EOSI1 interrupt to be generated upon completion of the scan. For looping scan modes, the interrupt triggers after the completion of each iteration of the loop. 0 Interrupt disabled 1 Interrupt enabled
10–6	Reserved, should be cleared.
5 SIMULT	Simultaneous Mode bit. This bit only affects parallel scan modes. When SIMULT equals 1, parallel scans operate in simultaneous mode. The scans in the A and B converter operate simultaneously and always result in pairs of simultaneous conversions in the A and B converter. START0, STOP0, SYNC0, and EOSIE0 control bits and the SYNC0 input are used to start and stop scans in both converters simultaneously. A scan ends in both converters when either converter encounters a disabled sample slot. When the parallel scan completes, the EOSI0 triggers if EOSIE0 is set. The CIP0 status bit indicates that a parallel scan is in process. When SIMULT equals 0, parallel scans in the A and B converters operate independently. The B converter has its own independent set of the above controls (START1, STOP1, SYNC1, EOSIE1, SYNC1) designed to control its operation and report its status. Each converter′s scan continues until its sample list is exhausted (four samples) or a disabled sample is encountered. For looping parallel scan mode, each converter starts its next iteration as soon as the previous iteration in that converter is complete and continues until the STOP bit for that converter is asserted. 0 Parallel scans occur independently 1 Parallel scans occur simultaneously (default)
4–0 DIV	Clock Divisor Select. This field controls the divider circuit, which generates the ADC clock by dividing the system clock by 2×DIV+1. DIV must be chosen so the ADC clock does not exceed 5.0 MHz. See Table 28-5 for a listing of ADC clock frequency based on the value of DIV for several configurations.

Table 28-5. ADC Clock Frequency for Various Conversion Clock Sources

DIV	Divisor	ROSC Standby 400 kHz	ROSC Normal 8 MHz	PLL 64 MHz	External CLK
		200 kHz Sys Clock	4 MHz Sys Clock	32 MHz Sys Clock	CLK/2 Sys Clock
00000	2	100 kHz	2.00 MHz	16.0 MHz	CLK/4
00001	4	100 kHz	1.00 MHz	8.00 MHz	CLK/8
00010	6	100 kHz	500 kHz	5.33 MHz	CLK/12
00011	8	100 kHz	250 kHz	4.00 MHz	CLK/16
00100	10	100 kHz	125 kHz	3.20 MHz	CLK/20
—	—	—	—	—	—

Table 28-5. ADC Clock Frequency for Various Conversion Clock Sources (continued)

—	—	—	—	—	—
11111	64	100 kHz	62.5 kHz	500 kHz	CLK/128

28.4.3 Zero Crossing Control Register (ADZCC)

The ADC zero crossing control (ADZCC) register provides the ability to monitor the selected channels and determine the direction of zero crossing triggering the optional interrupt. Zero crossing logic monitors only the sign change between current and previous sample. The ZCE0 bit monitors the sample stored in ADRSLT0, ZCE1 bit monitors ADRSLT1, and ZCE7 bit monitors ADRSLT7. When the zero crossing is disabled for a selected result register, sign changes are not monitored or updated in the ADZCSTAT register.

IPSBAR Access: read/write
Offset: 0x19_0004 (ADZCC)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ZCE7		ZCE6		ZCE5		ZCE4		ZCE3		ZCE2		ZCE1		ZCE0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. Zero Crossing Control Register (ADZCC)

Table 28-6. ADZCC Field Descriptions

Field	Description
15–0 ZCE*n*	Zero Crossing Enable. For each channel *n*, setting the ZCE*n* field allows detection of the indicated zero crossing condition, provided the corresponding offset register (ADOFS*n*) has a value *offset*, 0 < *offset* < 0x7FF8. 00 Zero crossing disabled 01 Zero crossing enabled for positive to negative sign change 10 Zero crossing enabled for negative to positive sign change 11 Zero crossing enabled for any sign change

28.4.4 Channel List 1 and 2 Registers (ADLST1 and ADLST2)

The channel list register contains an ordered list of the analog input channels to be converted when the next scan is initiated. If all samples are enabled in the ADSDIS register, a sequential scan of inputs proceeds in order of SAMPLE0 through SAMPLE7. If one of the parallel sampling modes is selected instead, the converter A sampling order is SAMPLE0-3, and the converter B sampling order is SAMPLE4-7.

In sequential modes, the sample slots are converted in order from SAMPLE0 to SAMPLE7. Analog input pins can be sampled in any order, including sampling the same input pin more than once.

In parallel modes, converter A processes sample slots SAMPLE0 through SAMPLE3, while converter B processes sample slots SAMPLE4 through SAMPLE7. Because converter A only has access to analog inputs AN0 through AN3, sample slots SAMPLE0-3 should only contain binary values between 000 and 011. Likewise, because converter B only has access to analog inputs AN4 through AN7, sample slots

SAMPLE4-7 should only contain binary values between 100 and 111. No damage occurs if this constraint is violated, but results are undefined.

When inputs are configured as differential pairs, a reference to either analog input in a differential pair by a sample slot implies a differential measurement on the pair. The details of single ended and differential measurement are described in Section 28.5.2.1, “Single-Ended Samples” and Section 28.5.2.2, “Differential Samples”. Sample slots are disabled using the ADSDIS register.

IPSBAR Access: read/write
Offset: 0x19_0006 (ADLST1)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		SAMPLE3				SAMPLE2				SAMPLE1				SAMPLE0		
W																
Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 28-6. Channel List 1 Register (ADLST1)

Table 28-7. ADLST1 Field Descriptions

Field	Description
15	Reserved, should be cleared.
14–12 SAMPLE3	Sample input channel select 3. The settings for this field are given in Table 28-9.
11	Reserved, should be cleared.
10–8 SAMPLE2	Sample input channel select 2. The settings for this field are given in Table 28-9.
7	Reserved, should be cleared.
6–4 SAMPLE1	Sample input channel select 1. The settings for this field are given in Table 28-9.
3	Reserved, should be cleared.
2–0 SAMPLE0	Sample input channel select 0. The settings for this field are given in Table 28-9.

IPSBAR Access: read/write
Offset: 0x19_0008 (ADLST2)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R		SAMPLE7				SAMPLE6				SAMPLE5				SAMPLE4		
W																
Reset	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0

Figure 28-7. Channel List 2 Register (ADLST2)

Table 28-8. ADLST2 Field Descriptions

Field	Description
15	Reserved, should be cleared.
14–12 SAMPLE7	Sample input channel select 7. The settings for this field are given in Table 28-9.
11	Reserved, should be cleared.
10–8 SAMPLE6	Sample input channel select 6. The settings for this field are given in Table 28-9.
7	Reserved, should be cleared.
6–4 SAMPLE5	Sample input channel select 5. The settings for this field are given in Table 28-9.
3	Reserved, should be cleared.
2–0 SAMPLE4	Sample input channel select 4. The settings for this field are given in Table 28-9.

Table 28-9. ADC Input Conversion for Sample Bits

SAMPLE*n*[2:0]			ADC Input Pins Selected	
Sequential Mode	Parallel Mode			
n=0,1,2,...,7	*n*=0,1,2,3 (Conv. A)	*n*=4,5,6,7 (Conv. B)	Single Ended	Differential
000	000		AN0	AN0+, AN1–
001	001		AN1	
010	010		AN2	AN2+, AN3–
011	011		AN3	
100		100	AN4	AN4+, AN5–
101		101	AN5	
110		110	AN6	AN6+, AN7–
111		111	AN7	

28.4.5 Sample Disable Register (ADSDIS)

This register is an extension to the ADLST1and ADLST2, providing the ability to enable only the desired samples programmed in the SAMPLE0–SAMPLE7. At reset, all samples are enabled. For example, if in sequential mode and bit DS5 is set to 1, SAMPLE0 through SAMPLE4 are sampled. However, if in parallel mode and bits DS5 or DS1 are set to 1, only SAMPLE0 and SAMPLE4 are sampled.

IPSBAR Access: read/write

Offset: 0x19_000A (ADSDIS)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. Sample Disable Register (ADSDIS)

Table 28-10. ADSDIS Field Descriptions

Field	Description
15–8	Reserved, should be cleared.
7–0 DS*n*	Disable Sample bits. Setting or clearing DS*n* enables or disables the corresponding SAMPLE*n* field. 0 Enable SAMPLE*n* 1 Disable SAMPLE*n* and all subsequent samples. Which samples are actually disabled depends on the conversion mode, sequential/parallel, and the value of SIMULT.

28.4.6 Status Register (ADSTAT)

This register provides the current status of the ADC module. RDY*n* bits are cleared by reading their corresponding result (ADRSLT*n*) registers. The HLMTI and LLMTI bits are cleared by writing 1 to each asserted bit in the ADC limit status (ADLSTAT) register. Likewise, the ZCI bit is cleared by writing 1 to each asserted bit in the ADC zero crossing status (ADZCSTAT) register. The EOSI*n* bits are cleared by writing 1 to them.

Except for CIP0 and CIP1 all bits in ADSTAT are sticky – after being set, they require some specific action to be cleared. They are not cleared automatically on the next scan sequence.

IPSBAR Access: read/write

Offset: 0x19_000C (ADSTAT)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CIP0	CIP1	0	EOSI1	EOSI0	ZCI	LLMTI	HLMTI	RDY7	RDY6	RDY5	RDY4	RDY3	RDY2	RDY1	RDY0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-9. Status Register (ADSTAT)

Table 28-11. ADSTAT Field Descriptions

Field	Description
15 CIP0	Conversion in Progress 0 bit. This bit indicates when a scan is in progress. This bit supports any sequential scan or parallel scan with SIMULT equaling 1. When executing a parallel scan with SIMULT equaling 0, this bit services the scan of converter A, and the CIP1 bit services the scan of converter B. 0 Idle state 1 A scan cycle is in progress (the ADC ignores all sync pulses or start commands)
14 CIP1	Conversion in Progress 1 bit. This refers only to a B converter scan in non-simultaneous (SIMULT=0) parallel scan modes. 0 Idle state 1 A scan cycle is in progress (the ADC ignores all sync pulses or start commands)
13	Reserved, should be cleared.
12 EOSI1	End of Scan Interrupt 1 bit. This bit indicates whether a scan of analog inputs has been completed since the last read of ADSTAT or a reset. The EOSI1 bit is cleared by writing a 1 to it. This bit cannot be set by software. In looping scan modes, this interrupt is triggered at the completion of each iteration of the loop. This interrupt is triggered only by the completion of a B converter scan in non-simultaneous (SIMULT=0) parallel scan modes. In this case the EOSI0 interrupt is triggered when converter A completes its scan. 0 A scan cycle has not been completed, no end of scan IRQ pending 1 A scan cycle has been completed, end of scan IRQ pending
11 EOSI0	End of Scan Interrupt 0 bit. This bit indicates whether a scan of analog inputs has been completed since the last read of ADSTAT or a reset. The EOSI0 bit is cleared by writing a 1 to it. This bit cannot be set by software. EOSI0 is the preferred bit to poll for scan completion if interrupts are not enabled. In looping scan modes, this interrupt is triggered at the completion of each iteration of a loop. This interrupt is triggered upon the completion of any sequential scan or parallel scan with SIMULT equaling 1. When executing parallel scans with SIMULT equaling 0, this interrupt is triggered when converter A completes its scan while the EOSI1 interrupt services converter B. 0 A scan cycle has not been completed, no end of scan IRQ pending 1 A scan cycle has been completed, end of scan IRQ pending
10 ZCI	Zero Crossing Interrupt bit. This bit is asserted at the completion of an individual conversion experiencing a zero crossing enabled in the ADC zero crossing control (ADZCC) register. The bit is set as soon as an enabled zero crossing event occurs rather than at the end of the ADC scan. ZCI is cleared by writing 1 to all active ADZCSTAT[ZCS] bits. 0 No ZCI interrupt request 1 Zero crossing encountered; IRQ pending if CTRL1[ZCIE] is set
9 LLMTI	Low Limit Interrupt bit. If any low limit register (ADLLMT*n*) is enabled by having a value other than 0x0, low limit checking is enabled. This bit is set at the completion of an individual conversion which may or may not be the end of a scan. It is cleared by writing 1 to all active ADLSTAT[LLS] bits. 0 No low limit interrupt request 1 Low limit exceeded, IRQ pending if CTRL1[LLMTIE] is set

Table 28-11. ADSTAT Field Descriptions (continued)

Field	Description
8 HLMTI	High Limit Interrupt bit. If any high limit register (ADHLMT*n*) is enabled by having a value other than 0x7FF8, high limit checking is enabled. This bit is set at the completion of an individual conversion which may or may not be the end of a scan. It is cleared by writing 1 to all active ADLSTAT[HLS] bits. 0 No high limit interrupt request 1 High limit exceeded, IRQ pending if CTRL1[HLMTIE] is set
7–0 RDY*n*	Ready Sample bits. These bits indicate samples 7-0 are ready to be read. The RDY*n* bits are set as the individual channel conversions are completed and stored in a ADRSLT*n* register. These bits are cleared after a read from the corresponding ADC results (ADRSLT*n*) register. If polling the RDY*n* bits to determine if a particular sample is executed, care should be taken not to start a new scan until all enabled samples are completed. 0 Sample not ready or has been read 1 Sample ready to be read **Note:** RDY*n* bits can be cleared when the debugger reads the corresponding results register during a debug session.

28.4.7 Limit Status Register (ADLSTAT)

The ADC limit status (ADLSTAT) register latches in the result of the comparison between the result of the sample in the ADRSLT*n* register and the respective limit register, ADHLMT*n* or ADLLMT*n*.

For example, if the result for ADRSLT0 is greater than the value programmed into ADHLMT0, then the the HLS0 bit is set. An interrupt is generated if CTRL1[HLMTIE] is set.

These bits are sticky—they are not cleared automatically by subsequent conversions. A bit may only be cleared by writing a 1 to it.

IPSBAR Access: read/write
Offset: 0x19_000E (ADLSTAT)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	HLS7	HLS6	HLS5	HLS4	HLS3	HLS2	HLS1	HLS0	LLS7	LLS6	LLS5	LLS4	LLS3	LLS2	LLS1	LLS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-10. Limit Status Register (ADLSTAT)

Table 28-12. ADLSTAT Field Descriptions

Field	Description
15–8 HLS*n*	High Limit Status bits. These bits hold the result of a comparison between the sample (stored in ADRSLT*n*) and the high-limit value (stored in ADHLMT*n*). 0 Sample *n* is less than or equal to the associated high-limit value 1 Sample *n* is greater than the associated high-limit value **Note:** These bits are sticky, and can only be cleared by writing a 1 to them.
7–0 LLS*n*	Low Limit Status bits. These bits hold the result of a comparison between the sample (stored in ADRSLT*n*) and the low-limit value (stored in ADLLMT*n*). 0 Sample *n* is greater than or equal to the associated low-limit value 1 Sample *n* is less than the associated low-limit value **Note:** These bits are sticky, and can only be cleared by writing a 1 to them.

28.4.8 Zero Crossing Status Register (ADZCSTAT)

The ADC zero crossing status (ADZCSTAT) register latches in the result of a sign comparison between the current and previous sample. The type of comparison is controlled by the ADZCC register (see Section 28.4.3, “Zero Crossing Control Register (ADZCC)”).

For example, if the result for the channel programmed in SAMPLE0 changes sign from the previous conversion, and the respective ZCE bit in the ADZCC register is set to 0b11 (any edge change), then the ZCS0 bit is set. An interrupt is generated if CTRL1[ZCIE] is set.

These bits are sticky—they are not cleared automatically by subsequent conversions. A bit may only be cleared by writing a 1 to it.

IPSBAR Access: read/write
Offset: 0x19_0010 (ADZCSTAT)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	ZCS7	ZCS6	ZCS5	ZCS4	ZCS3	ZCS2	ZCS1	ZCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-11. Zero Crossing Status Register (ADZCSTAT)

Table 28-13. ADLSTAT Field Descriptions

Field	Description
15–8	Reserved, should be cleared.
7–0 ZCS*n*	Zero Crossing Status bits. These bits hold the result of a sign comparison between the current and previous sample. The type of comparison is controlled by the ADZCC register (see Section 28.4.3, “Zero Crossing Control Register (ADZCC)”). 0 Sample did not change sign, or sign comparison is disabled 1 Sample changed sign **Note:** These bits are sticky, and can only be cleared by writing a 1 to them.

28.4.9 Result Registers (ADRSLT*n*)

The 8 result registers contain the converted results from a scan. The SAMPLE*n* result is loaded into ADRSLT*n*. In a simultaneous parallel scan mode, the first channel pair, designated by SAMPLE0 and SAMPLE4 in register LIST1/2, is stored in ADRSLT0 and ADRSLT4, respectively.

When writing to this register, only the RSLT portion of the value written is used. This value is modified as shown in Figure 28-23 and the result of the subtraction is stored. The SEXT bit is only set as a result of this subtraction and is not directly determined by the value written.

RSLT can be interpreted as a signed integer or a signed fixed point (fractional) number. As a fixed point number, RSLT can be used directly. If RSLT is interpreted as a signed integer, you have two options:

- Right shift with sign extend (ASR) three places to fit it into the range [0,4095]
- Accept the number as presented in the register, knowing there are missing codes, because the lower three LSBs are always zero

Negative results (SEXT = 1) are always presented in twos-complement format. If an application requires that the result be always positive, the corresponding offset register (ADOFS*n*) must be set to 0x0.

The interpretation of the numbers programmed into the ADC limit and offset registers (ADLLMT*n*, ADHLMT*n*, and ADOFS*n*) must match your interpretation of the result register.

IPSBAR Offsets: 0x19_0012 (ADRSLT0)
0x19_0014 (ADRSLT1)
0x19_0016 (ADRSLT2)
0x19_0018 (ADRSLT3)
0x19_001A (ADRSLT4)
0x19_001C (ADRSLT5)
0x19_001E (ADRSLT6)
0x19_0020 (ADRSLT7)

Access: read/write

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

Figure 28-12. Result Registers (ADRSLT*n*)

Table 28-14. ADRSLT*n* Field Descriptions

Field	Description
15 SEXT	Sign Extend bit. 0 Result is positive 1 Result is negative **Note:** If only positive results are required, then the respective offset register (ADOFS*n*) must be set to 0x0.
14–3 RSLT	Result of the conversion.
2–0	Reserved, should be cleared.

28.4.10 Low and High Limit Registers (ADLLMT*n* and ADHLMT*n*)

Each ADC sample is compared against the values in the limit registers. The comparison is based upon the raw conversion value before the offset correction is applied. Refer to Figure 28-23**.** ADC limit registers (ADLLMT*n* and ADHLMT*n*) correspond to result registers (ADRSLT*n*). The high limit register is used for the comparison of result > high limit. The low limit register is used for the comparison of result < low limit.

Limit checking can be disabled by programming the respective limit register with 0x7FF8 for the high limit and 0x0000 for the low limit. At reset, limit checking is disabled.

IPSBAR 0x19_0022 (ADLLMT0) Access: read/write
Offsets: ox19_0024 (ADLLMT1)
0x19_0026 (ADLLMT2)
0x19_0028 (ADLLMT3)
0x19_002A (ADLLMT4)
0x19_002C (ADLLMT5)
0x19_002E (ADLLMT6)
0x19_0030 (ADLLMT7)

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

Figure 28-13. Low Limit Registers (ADLLMT*n*)

Table 28-15. ADLLMT*n* Field Descriptions

Field	Description
15	Reserved, should be cleared.
14–3 LLMT	Low limit.
2–0	Reserved, should be cleared.

IPSBAR 0x19_0032 (ADHLMT0) Access: read/write
Offset: 0x19_0034 (ADHLMT1)
0x19_0036 (ADHLMT2)
0x19_0038 (ADHLMT3)
0x19_003A (ADHLMT4)
0x19_003C (ADHLMT5)
0x19_003E (ADHLMT6)
0x19_0040 (ADHLMT7)

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

Figure 28-14. High Limit Registers (ADHLMT*n*)

Table 28-16. ADHLMT*n* Field Descriptions

Field	Description
15	Reserved, should be cleared.
14–3 HLMT	High limit.
2–0	Reserved, should be cleared.

28.4.11 Offset Registers (ADOFS*n*)

The values in the offset registers (ADOFS*n*) are subtracted from the raw ADC values, and the results are stored in the ADRSLT*n* registers. To obtain unsigned results, the respective offset register must be programmed with a value of 0x0 to yield a resulting range of 0x0 to 0x7FF8.

IPSBAR 0x19_0042 (ADOFS0) Access: read/write
Offsets: 0x19_0044 (ADOFS1)
0x19_0046 (ADOFS2)
0x19_0048 (ADOFS3)
0x19_004A (ADOFS4)
0x19_004C (ADOFS5)
0x19_004E (ADOFS6)
0x19_0050 (ADOFS7)

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

Figure 28-15. Offset Registers (ADOFS*n*)

Table 28-17. ADOFS*n* Field Descriptions

Field	Description
15	Reserved, should be cleared.
14–3 OFFSET	Offset value. This value is subtracted from the raw ADC value, and the result is stored in the respective ADRSLT*n* register.
2–0	Reserved, should be cleared.

28.4.12 Power Control Register (POWER)

This register controls the power management features of the ADC module. There are manual power-down control bits for the two ADC converters and the shared voltage reference generator. There are also 5 distinct power modes with related controls:

1. Powered-down state
   Each converter and the voltage reference generator can individually be put into a powered down state. When powered down, the unit consumes no power. Results of scans referencing a powered down converter are undefined. The voltage reference generator and at least one converter must be powered up to use the ADC module.
2. Manual power-down controls
   Each converter and the voltage reference generator have a manual power control bit capable of forcing that component into the power down state. Also, each converter and the voltage reference generator can be powered up/down automatically as part of ADC operation.
3. Idle state
   The ADC module is idle when neither of the two converters has a scan in process.
4. Active state
   The ADC module is active when at least one of the two converters has a scan in process.

5. Current mode

- Normal current mode is used to power the converters at clock rates above 100 kHz.
- Standby current mode uses less power and is engaged only when the ADC clock is at 100 kHz. The current mode active does not affect the number of ADC clock cycles required to do a conversion or the accuracy of a conversion. The ADC module may change the current mode when idle as part of the power saving strategy. Both converters are in the same current mode at all times.

In addition to the power modes, there is startup delay:

- Auto power-down and auto standby power modes cause a startup delay when the ADC module goes between the idle and active states to allow time to switch clocks or power configurations. The number of ADC clocks used in the startup delay is defined by the PUDELAY field.

See the discussion of power modes in the Functional Description Section 28.5, “Functional Description” for details of the 5 power modes and how to configure them. See Section 28.5.9, “ADC Clock,” for a more detailed description of the clocking system and the control of current mode.

IPSBAR Access: read/write

Offset: 0x19_0052 (POWER)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ASB	0	0	PSTS2	PSTS1	PSTS0	PUDELAY						APD	PD2	PD1	PD0
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	1	0	1	1	1

Figure 28-16. Power Control Register (POWER)

Table 28-18. POWER Field Descriptions

Field	Description
15 ASB	Auto Standby bit. This bit selects auto standby mode. ASB is ignored if APD is set. When the ADC is idle, auto standby mode selects the standby clock as the ADC clock source and puts the converters into standby current mode. At the start of any scan, the conversion clock is selected as the ADC clock and a delay of PUDELAY ADC clock cycles is imposed for current levels to stabilize. After this delay, the ADC initiates the scan. When the ADC returns to the idle state, the standby clock is again selected and the converters revert to the standby current state. 0 Auto standby mode disabled 1 Auto standby mode enabled
14–13	Reserved, should be cleared.
12 PSTS2	Voltage Reference Power Status bit. 0 Voltage reference circuit is currently enabled 1 Voltage reference circuit is currently disabled
11 PSTS1	Converter B Power Status bit. This bit is asserted immediately after PD1 is set. It is deasserted PUDELAY ADC clock cycles after PD1 is cleared if APD is 0. This bit can be read as a status bit to determine when the ADC is ready for operation. During auto power-down mode, this bit indicates the current powered state of converter B. 0 ADC converter B is currently enabled 1 ADC converter B is currently disabled

Table 28-18. POWER Field Descriptions (continued)

Field	Description
10 PSTS0	Converter A Power Status bit. This bit is asserted immediately after PD0 is set. It is deasserted PUDELAY ADC clock cycles after PD0 is cleared if APD is 0. This bit can be read as a status bit to determine when the ADC is ready for operation. During auto power-down mode, this bit indicates the current powered state of converter A. 0 = ADC converter A is currently enabled 1 = ADC converter A is currently disabled
9–4 PUDELAY	Power-Up Delay. This field determines the number of ADC clock cycles provided to enable an ADC converter (after clearing PD0 or PD1) before allowing a scan to start. It also determines the number of ADC clock cycles of delay provided in auto power-down (APD) and auto standby (ASB) modes between when the ADC goes from the idle to active state and when the scan is allowed to start. The default value is 13 ADC clock cycles. Accuracy of the initial conversions in a scan is degraded if PUDELAY is too low. **Note:** PUDELAY defaults to a value typically sufficient for any power mode. The latency of a scan can be reduced by reducing PUDELAY to the lowest value for which accuracy is not degraded. Please refer to the *Device Data Sheet* for further details.
3 APD	Auto Power-Down Mode bit. Auto power-down mode disables converters when they are not in use for a scan. APD takes precedence over ASB. When a scan is started in APD mode, a delay of PUDELAY ADC clock cycles is imposed during which the needed converter(s), if idle, are enabled. The ADC then initiates a scan equivalent to when APD is not active. When the scan is completed, the converter(s) are disabled again. 0 Auto power-down mode is not active 1 Auto power-down mode is active **Note:** If ASB or APD is asserted while a scan is in progress, that scan is unaffected and the ADC waits to enter its low-power state until after all conversions are complete and both ADCs are idle. **Note:** ASB and APD are not useful in looping modes. The continuous nature of scanning means the ADC can never enter the low-power state.
2 PD2	Power-Down Control for Voltage Reference Circuit bit. This bit controls the power-down of the ADC's voltage reference circuit. This circuit is shared by both converters. When PD2 is set, the voltage reference is activated when PD1 or PD0 are enabled. It is not usually necessary to modify this bit, because disabling (powering-down) converter A and converter B automatically powers-down the voltage reference. 0 Manually power-up voltage reference circuit 1 Power-down voltage reference circuit is controlled by PD0 and PD1 (default)

Table 28-18. POWER Field Descriptions (continued)

Field	Description
1 PD1	Manual Power-Down for Converter B bit. This bit forces Converter B to power-down. Setting PD1 powers-down converter B immediately. The results of a scan using converter B is invalid when PD1 is set. When PD1 is cleared, converter B is continuously powered-up (APD = 0) or automatically powered-up when needed (APD = 1). 0 Power-up ADC converter B 1 Power-down ADC converter B **Note:** When clearing PD1 in any power mode except auto power-down (APD = 1), wait PUDELAY ADC clock cycles before initiating a scan to stabilize power levels within the converter. The PSTS1 bit can be polled to determine when the PUDELAY time has elapsed. Failure to follow this procedure can result in loss of accuracy of the first two samples.
0 PD0	Manual Power-Down for Converter A bit. This bit forces Converter A to power-down. Setting PD0 powers-down converter A immediately. The results of a scan using converter A is invalid when PD0 is set. When PD0 is cleared, converter A is continuously powered-up (APD = 0) or automatically powered-up when needed (APD = 1). 0 = Power-up ADC converter A 1 = Power-down ADC converter A **Note:** When clearing PD0 in any power mode except auto power-down (APD = 1), wait PUDELAY ADC clock cycles before initiating a scan to stabilize power levels within the converter. The PSTS0 bit can be polled to determine when the PUDELAY time has elapsed. Failure to follow this procedure can result in loss of accuracy of the first two samples.

28.4.13 Voltage Reference Register (CAL)

In earlier series, this register supported ADC calibration and had a different name. Improvements in ADC performance have eliminated the need for on-chip calibration support, hence the new name.

IPSBAR Access: read/write

Offset: 0x19_0054 (CAL)

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

Figure 28-17. Voltage Reference Register (CAL)

Table 28-19. CAL Field Descriptions

Field	Description
15 SEL_VREFH	Select V_{REFH} Source bit. This bit selects the source of the V_{REFH} reference for conversions. 0 VRH 1 AN2
14 SEL_VREFL	Select V_{REFL} Source bit. This bit selects the source of the V_{REFL} reference for conversions. 0 VRL 1 AN6
13–0	Reserved, should be cleared.

28.5 Functional Description

The ADC's conversion process is initiated by a sync signal from one of two input pins (SYNCx) or by writing 1 to a START*n* bit.

Starting a single conversion actually begins a sequence of conversions, or a scan of up to 8 single-ended or differential samples one at a time in sequential scan mode. The operation of the module in sequential scan mode is shown in Figure 28-18.

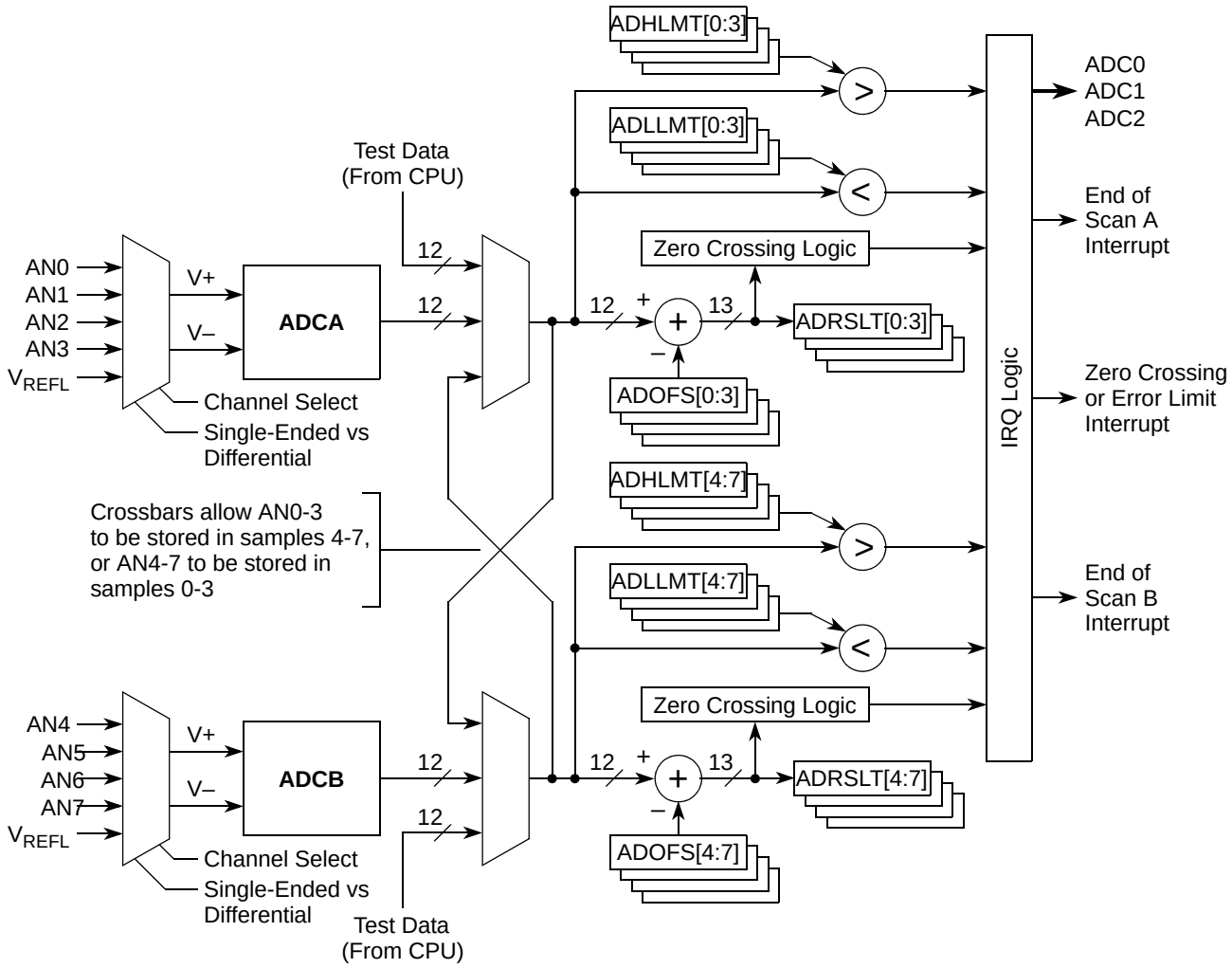


Figure 28-18. Sequential Mode Operation of the ADC

Scan sequence is determined by defining eight sample slots in ADLST1/2 registers, processed in order SAMPLE0-7 during sequential scan or in order SAMPLE0-3 by converter A and in order SAMPLE4-7 by converter B in parallel scan. SAMPLE slots may be disabled using the SDIS register.

The following pairs of analog inputs can be configured as a differential pair: AN0-1, AN2-3, AN4-5, and AN6-7. When configured as a differential pair, a reference to either member of the differential pair by a sample slot results in a differential measurement using that differential pair.

Parallel scan can be simultaneous or non-simultaneous. During simultaneous scan, the scans in the two converters are done simultaneously and always result in simultaneous pairs of conversions, one by converter A and one by converter B. The two converters share the same start, stop, sync, end-of-scan interrupt enable control, and interrupt. Scanning in both converters is terminated when either converter encounters a disabled sample. In non-simultaneous scan, the parallel scans in the two converters are achieved independently. The two converters have their own start, stop, sync, end-of-scan interrupt enable controls, and end-of-scan interrupts. Scanning in either converter terminates only when that converter encounters a disabled sample in its part of SDIS register (DS0-DS3 for A, DS4-DS7 for B).

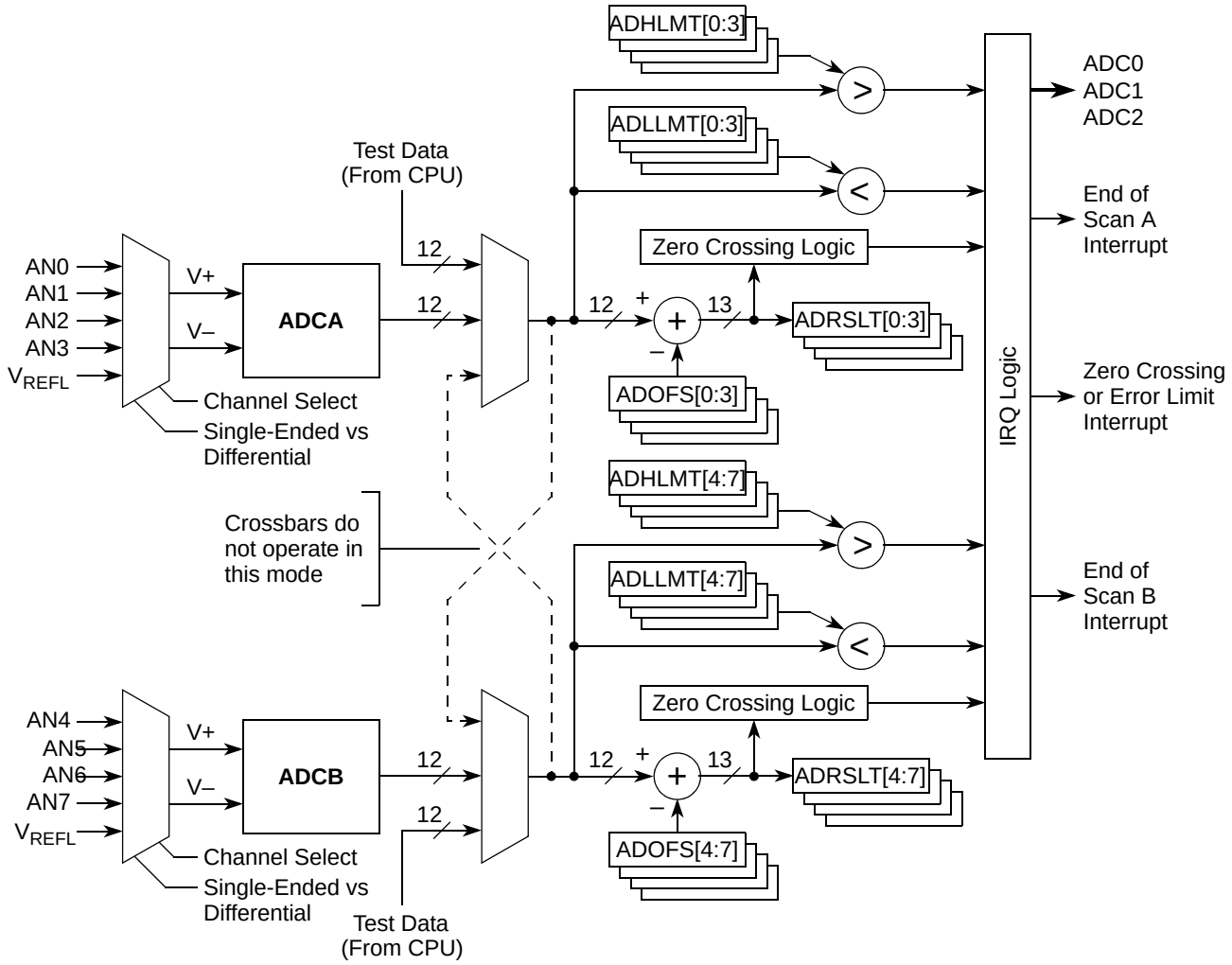


Figure 28-19. Parallel Mode Operation of the ADC

The ADC can be configured to perform a single scan and halt, perform a scan when triggered, or perform the scan sequence repeatedly until manually stopped. The single scan (once mode) differs from the triggered mode only in that SYNC input signals must be re-armed after each using a once mode scan, and subsequent SYNC inputs are ignored until the SYNC input is re-armed. This arming can occur anytime after the SYNC pulse occurs, even while the scan it initiated remains in process.

Optional interrupts can be generated at the end of a scan sequence. Interrupts are available simply to indicate the scan ended, that a sample was out of range, or at several different zero crossing conditions. Out-of-range is determined by the high and low limit registers.

To understand the operation of the ADC, it is important to understand the feature and limitations of each of the functional parts.

28.5.1 Input MUX Function

The input MUX function is shown in Figure 28-20**.** The channel select and single ended vs. differential switches are indirectly controlled based on settings within the LIST1, LIST2, and SDIS registers, and the CHNCFG field of the CTRL1 register.

1. MUXing for Sequential mode, single-ended conversions—During each conversion cycle (sample), any one input of the two muxes can be directed to any ADRSLT*n* register.
2. MUXing for sequential mode, differential conversions—During any conversion cycle (sample), either member of a differential pair may be referenced as a SAMPLE, resulting in a differential measurement on that pair being stored in the corresponding ADRSLT*n* register.
3. MUXing for parallel mode, single-ended conversions—During any conversion cycle (sample), any of AN0-AN3 can be directed to ADRSLT0-3 and any of AN4-AN7 can be directed to ADRSLT4-7.
4. MUXing for parallel mode, differential conversions—During any conversion cycle (sample), either member of differential pair AN0/1 or either member of differential pair AN2/3 can be referenced as a SAMPLE, resulting in a differential measurement of that pair being stored in one of the ADRSLT0-3 registers. Likewise, either member of differential pair AN4/5 or either member of differential pair AN6/7 can be referenced as a SAMPLE, resulting in a differential measurement of that pair being stored in one of the ADRSLT4-7 registers.

Details of switch operation is shown in Table 28-20**.** Internally, all measurements are performed differentially. During single ended measurements, V_{REFL} is used as the negative (-) input voltage, while the selected analog input is used as the positive (+) input.

Table 28-20. Analog MUX Controls for Each Conversion Mode

Conversion Mode	Channel Select Switches	Single Ended Differential Switches
Sequential, Single Ended	The two 1-of-4 select muxes can be set for the appropriate input line.	The lower switch selects V_{REFL} for the V- input of the A/D. The upper switch is always closed so that any of the four inputs can get to the V+ A/D input.
Sequential, Differential	The channel select switches are turned on in pairs, providing a dual 1-of-2 select function, such that either of the two differential channels can be routed to the A/D input.	The upper switch is open and the bottom switch selects the differential channel for the V- input of the A/D.

Table 28-20. Analog MUX Controls for Each Conversion Mode (continued)

Conversion Mode	Channel Select Switches	Single Ended Differential Switches
Parallel, Single Ended	The two 1-of-4 select muxes can be set for the appropriate input line.	The lower switch selects V_{REFL} for the V- input of the A/D. The upper switch is always closed so that any of the four inputs can get to the V+ A/D input.
Parallel, Differential	The channel select switches are turned on in pairs, providing a dual 1-of-2 select function, such that either of the two differential channels can be routed to the A/D input.	The upper and lower switches are open and the middle switch is closed, providing the differential channel to the differential input of the A/D.

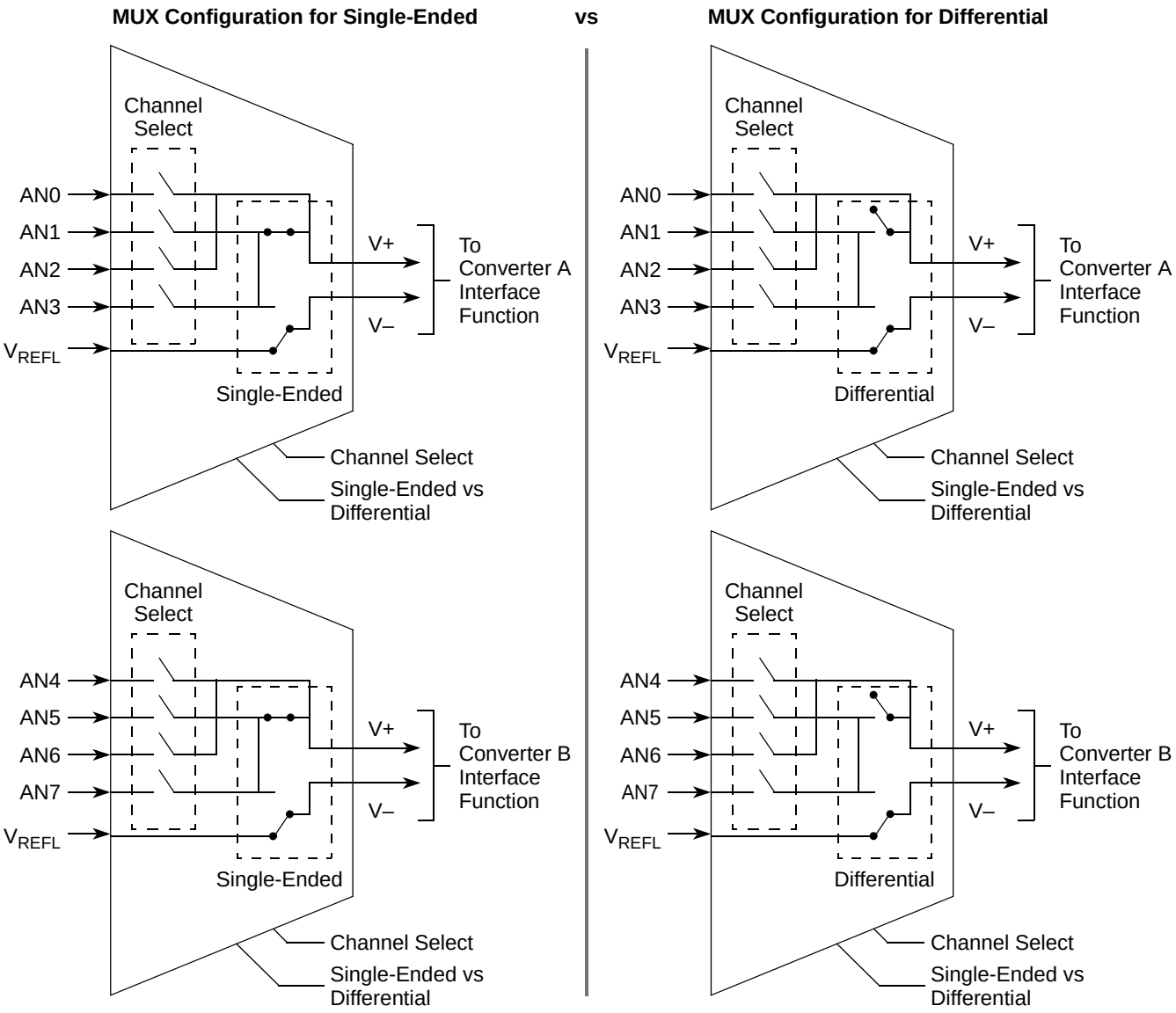


Figure 28-20. Input Select Mux

28.5.2 ADC Sample Conversion

The ADC consists of a cyclic, algorithmic architecture using two recursive sub-ranging sections (RSD#1 and RSD#2), shown in Figure 28-21**.** Each sub-ranging section resolves a single bit for each conversion clock, resulting in an overall conversion rate of two bits per clock cycle. Each sub-ranging section is designed to run at a maximum clock speed of 5.0 MHz. Thus a complete 12-bit conversion takes 6 ADC clocks (1.2 µs), not including sample or post processing time.

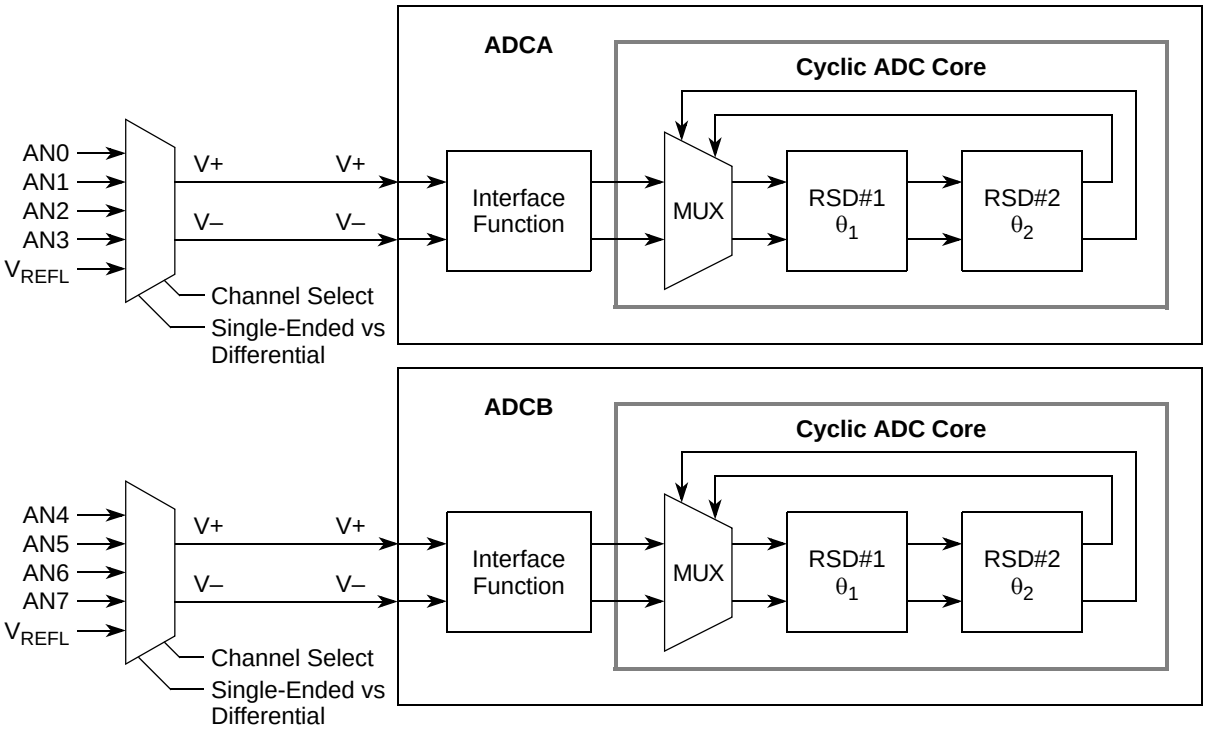


Figure 28-21. Cyclic ADC – Top Level Block Diagram

The input mode for a given sample is determined by the CHNCFG field of the CTRL1 register. The ADC has two input modes:

1. Single-ended mode (CHNCFG bit=0)—In single-ended mode, input mux of the ADC selects one of the analog inputs and directs it to the plus terminal of the A/D core. The minus terminal of the A/D core is connected to the V_{REFL} reference during this mode. The ADC measures the voltage of the selected analog input and compares it against the (V_{REFH} - V_{REFL}) reference voltage range.
2. Differential mode (CHNCFG bit = 1)—In differential mode, the ADC measures the voltage difference between two analog inputs and compares that against the (V_{REFH} - V_{REFL}) voltage range. The input is selected as an input pair: AN0/1, AN2/3, AN4/5, or AN6/7. In this mode, the plus terminal of the A/D core is connected to the even analog input, while the minus terminal is connected to the odd analog input.

A mix and match combination of differential and single-ended configurations may exist.

Examples:

- AN0 and AN1 differential, AN2 and AN3 single-ended
- AN4 and AN5 differential, AN6 and AN7 single-ended

28.5.2.1 Single-Ended Samples

The ADC module performs a ratio metric conversion. For single ended measurements, the digital result is proportional to the ratio of the analog input to the reference voltage in the following formula:

$$\text{SingleEndedValue} = \text{round}\left(\frac{V_{IN} - V_{REFL}}{V_{REFH} - V_{REFL}} \times 4095\right) \times 8$$

VIN = Applied voltage at the input pin

VREFH and VREFL = Voltage at the external reference pins on the device (typically $V_{REFH} = V_{DDA}$ and $V_{REFL} = V_{SSA}$)

Note: The 12-bit result is rounded to the nearest LSB.

Note: The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32760.

Single-ended measurements return the max value 32760 when the input is at V_{REFH}, return 0 when the input is at V_{REFL}, and scale linearly between based on the amount by which the input exceeds V_{REFL}.

28.5.2.2 Differential Samples

For differential measurements, the digital result is proportional to the ratio of the difference in the inputs to the difference in the reference voltages (V_{REFH} and V_{REFL}). Figure 28-22 shows typical configurations for differential inputs.

When converting differential measurements, the following formula is useful:

$$\text{DifferentialValue} = \text{round}\left(\frac{V_{IN1} - V_{IN2}}{V_{REFH} - V_{REFLO}} \times 4095\right) \times 8$$

V_{IN} = Applied voltage at the input pin

V_{REFH} and V_{REFL} = Voltage at the external reference pins on the device (typically $V_{REFH} = V_{DDA}$ and $V_{REFL} = V_{SSA}$)

Note: The 12-bit result is rounded to the nearest LSB.

Note: The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32760.

Differential measurements return the max value 32760 (= 4095 × 8) when the plus (+) input is V_{REFH} and the minus (–) input is V_{REFL}, return 0 when the plus (+) input is at V_{REFL} and the minus (–) input is at V_{REFL}, and scale linearly between based on the voltage difference between the two signals.

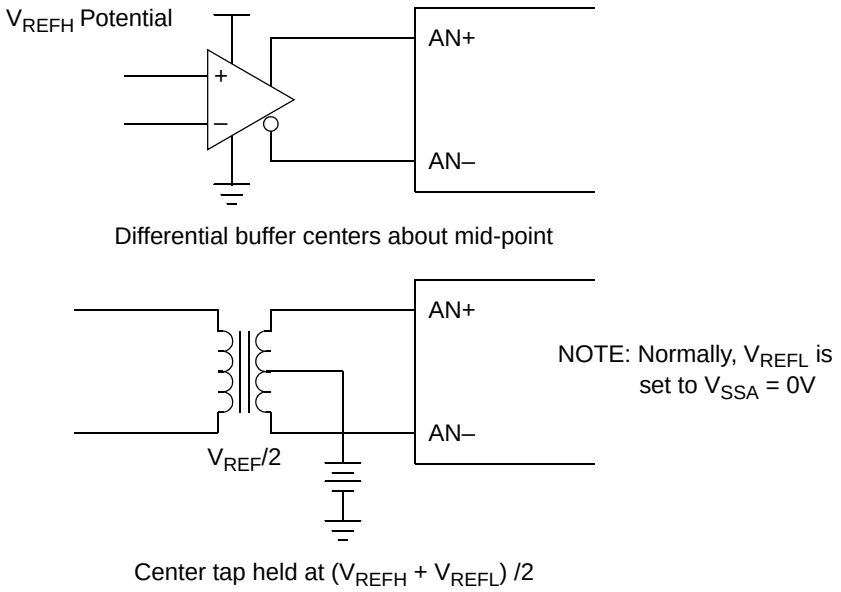


Figure 28-22. Typical Connections for Differential Measurements

28.5.3 ADC Data Processing

As shown in Figure 28-23, the raw result of the ADC conversion process is sent to an adder for offset correction. The adder subtracts the ADOFS*n* register value from each sample and the result is stored in the corresponding result register (ADRSLT*n*). Concurrent to this the raw ADC value is checked for limit violations, and the ADRSLT*n* values are checked for zero-crossing. Appropriate interrupts are asserted, if enabled.

The sign of the result is calculated from the ADC unsigned result minus the respective offset register. If the offset register is programmed with a value of zero, the result register value is unsigned and equals the cyclic converter unsigned result. The range of the result registers (ADRSLT*n*) is 0x0000–0x7FF8, assuming the offset (ADOFS*n*) registers are set to zero.

The processor can write to the result registers when the ADC is in stop mode or powered down. The data from this write operation is treated as if it came from the ADC analog core; so the limit checking, zero crossing, and the offset registers function as if in normal mode. For example, if the ADC is stopped and the processor writes to ADRSLT5, the data written to ADRSLT5 is muxed to the ADC digital logic inputs, processed, and stored into ADRSLT5, as if the analog core had provided the data. This test data must be left justified by 3 bits (as shown in the ADRSLT register definition) and does not include the sign bit. The sign bit (SEXT) is calculated during subtraction of the corresponding ADOFS*n* offset value.

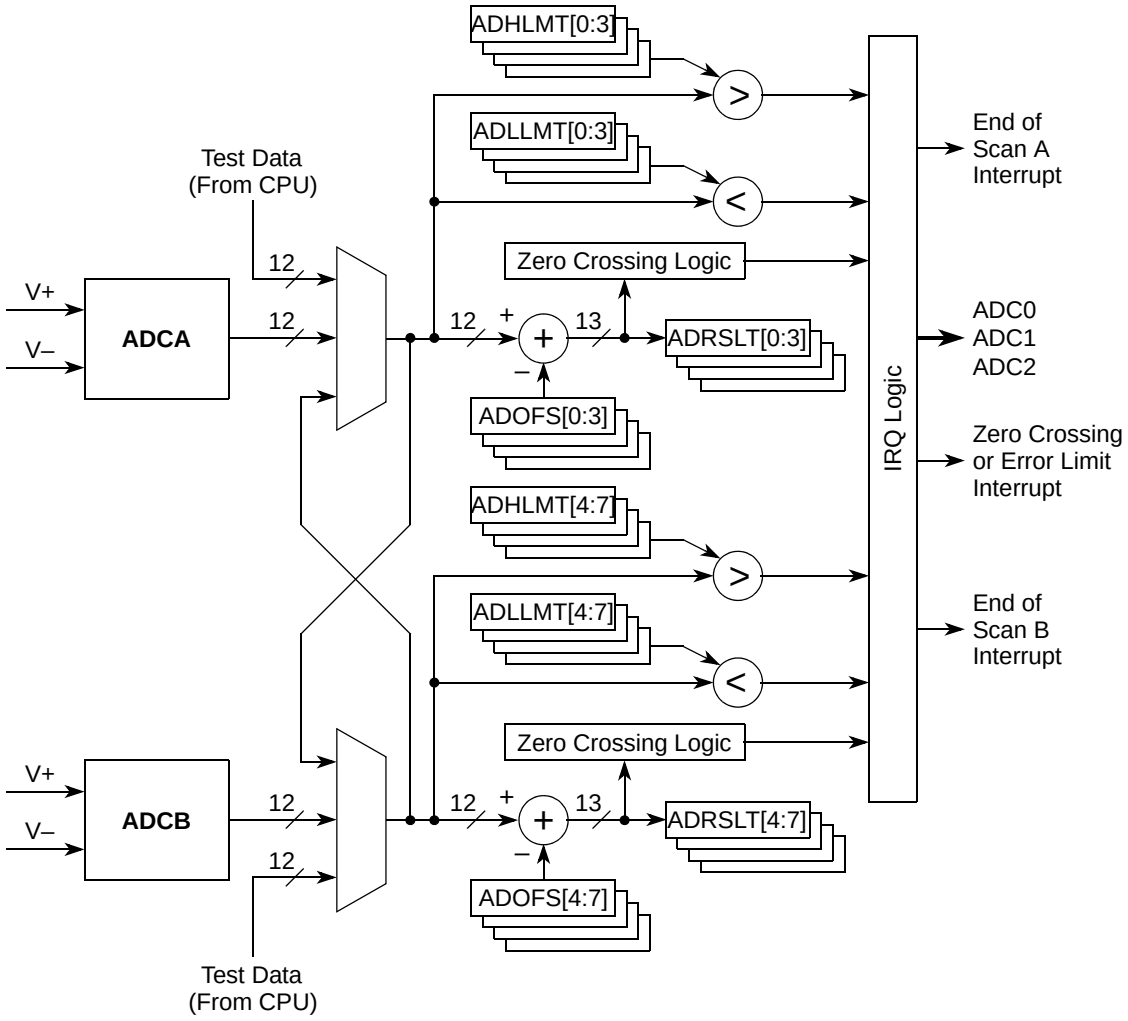


Figure 28-23. Result Register Data Manipulation

28.5.4 Sequential vs. Parallel Sampling

All scan modes make use of the 8 SAMPLE slots in the ADLST1 and ADLST2 registers. These slots are used to define which single-ended input or differential input pair is measured at each step in a scan sequence. The SDIS register is used to disable unneeded slots.

Differential measurements are made on input pairs AN0/1, AN2/3, AN4/5, and AN6/7 using the CHNCFG field of the CTRL1 register. A single ended measurement is made if a SAMPLE slot refers to an input not configured as a member of a differential pair by CHNCFG. A differential measurement is made if a SAMPLE slot refers to either member of a differential pair. Refer to the CHNCFG field description in the CTRL1 register for details of differential and single ended measurement.

Scan modes are sequential or parallel, as defined by the SMODE field of the CTRL1 register. In sequential scans, up to 8 SAMPLE slots are sampled one at a time in the order SAMPLE 0-7. Each SAMPLE slot may refer to any of the 8 analog inputs (AN0-7), thus the same input may be referenced by more than one SAMPLE slot. Scanning is initiated when the START0 bit is written as 1 or, if the SYNC0

bit is 1, when the SYNC0 input goes high. A scan ends when the first disabled sample slot is encountered in the SDIS register. Completion of the scan triggers the EOSI0 interrupt if the interrupt is enabled by the EOSIE0 bit. The START0 bit and SYNC0 input are ignored while a scan is in process. Scanning stops and cannot be initiated when the STOP0 bit is set.

Parallel scans differ in that converter A collects up to 4 samples (SAMPLE 0-3) in parallel to converter B collecting up to 4 samples (SAMPLE 4-7). SAMPLEs 0-3 may only reference inputs AN0-3, and SAMPLEs 4-7 may only reference inputs AN4-7. Within these constraints, any sample may reference any pin and the same input may be referenced by more than one sample slot.

By default (when SIMULT=1), parallel scans of the converters are initiated together when the START0 bit is written as 1 or, if the SYNC0 bit is 1, when the SYNC0 input goes high. The scan in both converters terminates when either converter encounters a disabled sample slot in SDIS. Completion of a scan triggers the EOSI0 interrupt provided the EOSIE0 interrupt enable is set. Samples are always taken simultaneously in the A and B converters. Setting the STOP0 bit stops and prevents the initiation of scanning in both converters.

Setting SIMULT equal to 0 (non-simultaneous mode) causes parallel scanning to operate independently in the A and B converter. Each converter has its own set of START*n*, STOP*n*, SYNC*n*, and EOSIE*n* control bits, SYNC*n* input, EOSI*n* interrupt, and CIP*n* status indicators (*n* = 0 for converter A, *n* = 1 for converter B). Although continuing to operate in parallel, the scans in the A and B converter start and stop independently according to their own controls. They may be simultaneous, phase shifted, or asynchronous, depending on when scans are initiated on the respective converters. The A and B converter may be of different length (up to a maximum of four) and each converter's scan completes when a disabled sample is encountered in that converters sample list only. STOP0 only stops the A converter, and STOP1 only stops the B converter. Looping scan modes repeat independently, with the A converter capturing SAMPLE 0-3, and B converter capturing SAMPLE 4-7. In loop modes, each converter independently restarts its scan after capturing its samples.

28.5.5 Scan Sequencing

Scan modes break down into three types based on how they repeat: once, triggered, or loop. Be certain to read Section 28.5.4, "Sequential vs. Parallel Sampling" to understand the operation of sequential and parallel scan modes before proceeding.

During a once mode scan, a single sequential or parallel scan is executed. Once scan modes differ from triggered scan modes in that they must be re-armed after each use. While all scan modes ignore sync pulses occurring while a scan is in process, once scan modes continues to ignore sync pulses even after the scan completes until re-armed. However, re-arming can occur any time, including during the scan, by writing to a CTRL*n* register. If operating in a sequential mode or simultaneous parallel, write to the CTRL1 register. If operating in a non-simultaneous parallel mode, re-arm converter A by writing to the CTRL1 register and converter B by writing to the CTRL2 register.

Triggered scan modes are identical to the corresponding once scan modes, except that re-arming of sync inputs is not necessary.

Loop scan modes automatically restart a scan as soon as the previous scan completes. In the loop sequential mode, up to 8 samples are captured in each loop, and the next scan starts immediately after the

completion of the previous scan. In loop parallel scan modes, both converters restart together if SIMULT equals 1 and restart independently if SIMULT equals 0. All subsequent start and sync pulses are ignored after the scan begins. Scanning can only be terminated by setting a STOP*n* bit. Use STOP0 in the CTRL1 register if operating in a sequential or simultaneous parallel mode. If operating in a non-simultaneous parallel mode, use STOP0 to stop converter A and STOP1 in the CTRL2 register to stop converter B.

28.5.6 Scan Configuration and Control

The operation of the ADC module is controlled by the CTRL1 and CTRL2 registers. The CTRL1 register is described in Section 28.4.1, "Control 1 Register (CTRL1)". The structure of the CTRL2 register depends on whether the ADC is in sequential-scan or parallel-scan mode (see Section 28.4.2.1, "CTRL2 Under Sequential Scan Modes" and Section 28.4.2.2, "CTRL2 Under Parallel Scan Modes", respectively). These are used to set the scan mode, configure channels, and start/stop scans.

The ADC can operate in several sequential or parallel scan modes, as determined by CTRL1[SMODE]. These are summarized in Table 28-21. When the ADC operates in a parallel scan mode, its functionality can be further controlled by CTRL2[SIMULT].

All scan modes make use of the 8 sample slots defined by the ADLST1 and ADLST2 registers. A scan is the process of stepping through these sample slots, converting the analog input indicated by that slot, and storing the result. Slots that are not required may be disabled by writing 1 to the appropriate bits of the SDIS register.

Input pairs AN0-1, AN2-3, AN4-5, and AN6-7 may be configured as differential pairs using CTRL1[CHNCFG]. When a slot in ADLST*n* refers to either member of a differential pair, a differential measurement on that pair is made; otherwise, a single-ended measurement is taken on that input. The details of single-ended and differential measurements are described in Section 28.5.2.1, "Single-Ended Samples" and Section 28.5.2.2, "Differential Samples", respectively.

CTRL1[SMODE] determines whether the slots are used to perform a sequential scan of up to 8 samples or 2 parallel scans up to 4 samples. It also controls how these scans are initiated/terminated and whether the scans are performed one time or repetitively. For more details, please see Figure 28-18 and Figure 28-19.

Parallel scans may be simultaneous or non-simultaneous depending on CTRL2[SIMULT]. This bit only applies to parallel operating modes and is ignored during sequential operating modes. During simultaneous parallel scans, A and B converters scan synchronously using one set of shared controls (CTRL1 register). During non-simultaneous scans, the A and B converters operate asynchronously with each converter using its own independent set of controls (CTRL1 for A and CTRL2 for B). Refer to Section 28.4.2.2, "CTRL2 Under Parallel Scan Modes," for more information.

Table 28-21. ADC Scan Modes

Scan Mode	Description
Once sequential	Upon START or an enabled sync signal, samples are taken one at a time starting with SAMPLE0 until a first disabled sample is encountered. If no disabled sample is encountered in the ADSDIS register, conversion concludes after SAMPLE7. If the scan is initiated by a sync signal, only one scan is completed until the converter is rearmed by writing to the CTRL1 register.
Once parallel	Upon START or an armed and enabled sync signal, converter A captures samples 0-3 and converter B captures samples 4-7. By default (CTRL2[SIMULT]=1), samples are taken simultaneously (synchronously), and scanning stops when either converter encounters a disabled sample or both converters complete all four samples. When SIMULT equals 0, samples are taken asynchronously, and scanning stops when each converter encounters a disabled sample in its part of the SDIS register or completes all 4 samples. If the scan is initiated by a sync signal, only one scan is completed until the converter is re-armed by writing to the CTRL1 register. (When SIMULT equals 0, the B converter must be re-armed separately by writing to the CTRL2 register.)
Loop sequential	Upon an initial start or enabled sync pulse, up to 8 samples are taken one at a time until a disabled sample is encountered. The process repeats until the STOP0 bit is set. While a loop mode is running, any additional start commands or sync pulses are ignored. If auto standby (POWER[ASB]=1) or auto power-down (POWER[APD]=1) is the selected power mode control, the power-up delay defined by PUDELAY is applied only on the first conversion.
Loop parallel	Upon an initial start or enabled sync pulse, converter A captures Samples 0-3, and converter B captures Samples 4-7. Each time a converter completes its current scan, it immediately restarts its scan sequence. This continues until a STOP*n* bit is asserted. While a loop is running, any additional start commands or sync pulses are ignored. By default (CTRL2[SIMULT]=1), samples are taken simultaneously (synchronously), and scanning stops when either converter encounters a disabled sample or both converters complete all four samples. When SIMULT equals 0, samples are taken asynchronously, and scanning stops when each converter encounters a disabled sample in its part of the SDIS register or completes all 4 samples. If auto standby or auto power-down is the selected power mode control, the power-up delay defined by PUDELAY is applied only on the first conversion.
Triggered sequential	Upon START or an enabled sync signal, samples are taken one at a time starting with SAMPLE0 until a first disabled sample is encountered. If no disabled sample is encountered, conversion concludes after SAMPLE7. If external sync is enabled, new scans are started for each sync pulse that is non-overlapping with a current scan in progress.
Triggered parallel (default)	Upon START or an enabled sync signal, converter A converts Samples 0-3, and converter B converts Samples 4-7 in parallel. By default (CTRL2[SIMULT]=1), samples are taken simultaneously (synchronously), and scanning stops when either converter encounters a disabled sample or both converters complete all four samples. When CTRL2[SIMULT] equals 0, samples are taken asynchronously, and scanning stops when each converter encounters a disabled sample in its part of the ADSDIS register or completes all 4 samples. If external sync is enabled (SYNC0=1), new scans are started for each sync pulse as long as the ADC has completed the previous scan (STAT[CIP*n*]=0).

28.5.7 Interrupt Sources

Figure 28-24 illustrates how five interrupt sources are combined into three entries in the interrupt vector table.

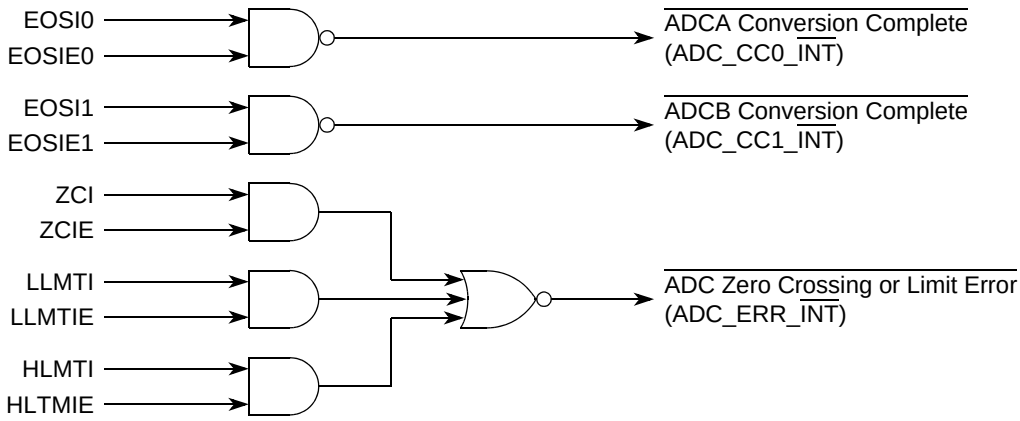


Figure 28-24. ADC Interrupt Sources

28.5.8 Power Management

The five supported power modes are described below. They are in order of highest to lowest power utilization at the expense of increased conversion latency and/or startup delay. Please see Section 28.5.9, "ADC Clock," for details of the various clocks referenced below.

28.5.8.1 Power Management Modes

1. Normal power mode
   This mode operates when:
   - At least one ADC converter is powered up (PD0 or PD1=0 in the POWER register);
   - Auto power-down and auto standby modes are disabled (APD=0, ASB=0 in the POWER register);
   - The ADC's clock is enabled (ADC=1 in the SIM module's SIM_PCE register).

In this mode, the ADC uses the conversion clock as the ADC clock source when active or idle. To minimize conversion latency, it is recommended the conversion clock be configured to 5.0 MHz. No startup delay (defined by PUDELAY in the POWER register) is imposed.

2. Auto power-down mode
   This mode operates when:
   - At least one ADC converter is powered up (PD0 or PD1=0 in the POWER register);
   - Auto power-down mode is enabled (APD=1 in the POWER register);
   - The ADC's clock is enabled (ADC=1 in the SIM module's SIM_PCE register).

Auto power-down and standby modes can be used together by setting APD equal to 1 in the above configuration. This hybrid mode converts at an ADC clock rate of 100 kHz using standby current mode when active, and gates off the ADC clock and powers down the converters when idle. A startup delay of

PUDELAY ADC clock cycles execute at the start of all scans while the ADC engages the conversion clock and the ADC powers up, stabilizing in the standby current mode. This provides the lowest possible power configuration for ADC operation.

3. Auto standby mode
   This mode operates when:
   - At least one ADC converter is powered up (PD0 or PD1=0 in the POWER register);
   - Auto power-down is disabled (APD=0 in the POWER register);
   - Auto standby is enabled (ASB=1 in the POWER register);
   - The ADC's clock is enabled (ADC=1 in the SIM module's SIM_PCE register);
   - The relaxation oscillator must be enabled for 8-MHz operation or the external oscillator clock must be running at 8 MHz in this mode.

In auto standby mode, the ADC uses the conversion clock when active and the100 kHz Standby clock when idle. The standby (low current) state automatically engages when the ADC is idle. The ADC executes a startup delay of PUDELAY ADC clocks at the start of all scans, allowing the ADC to switch to the Conversion clock and to revert from standby to normal current mode.

It is recommended the conversion clock be configured at or near 5.0 MHz to minimize conversion latency when active. In this mode, the ADC uses the conversion clock when active and gates off the conversion clock and powers down the converters when idle. A startup delay of PUDELAY ADC clocks is executed at the start of all scans, allowing the ADC to stabilize when switching to normal current mode from a completely powered off condition. This mode uses less power than normal and more power than auto standby. It requires more startup latency than auto standby when leaving the idle state to start a scan (higher PUDELAY value).

4. POWER-DOWN MODE
   This mode operates when:
   - Both ADC converters are powered down (PD0=PD1=1 in the POWER register);
   - The ADC's clock is disabled (ADC=0 in the SIM module's SIM_PCE register).

   In this configuration, the clock trees to the ADC and all of its analog components are shut down and the ADC uses no power.

28.5.8.2 Power Management Details

The ADC voltage reference and converters are powered down (PDn=1 in the POWER register) on reset. Individual converters can be manually powered down when not in use (PD0=1 or PD1=1), and the voltage reference can be automatically powered down when no converter is in use (PD2=1) or manually powered up when no converters are powered (PD2=0). When the ADC voltage reference is powered down, output reference voltages are set to low (V_{SSA}).

A delay of PUDELAY ADC clock cycles is imposed when PD0 or PD1 are cleared to power-up a converter and when the ADC goes from an idle (neither converter has a scan in process) to an active state when not operating in normal power mode. The ADC is active when at least one converter has a scan in process. A device recommends the use of two PUDELAY values: a large value for full power-up and a smaller value for going from standby current levels to full power-up. The following paragraphs provide an explanation of how to use PUDELAY when starting the ADC up or changing modes.

When starting up in normal mode, first set PUDELAY to the large power-up value. Next, clear the PD0 and or PD1 bits to power-up the required converters. Poll the status bits (PSTS*n* in the POWER register) until all required converters are powered up. Following polling, start scan operations. The value in PUDELAY provides a power-up delay before scans begin. Because normal mode does not use PUDELAY at start of scans, no further delays are imposed.

When starting up using auto standby mode, first use the normal mode startup procedure. Before starting scan operations, set PUDELAY to the smaller value, then set ASB in the POWER register. Auto standby mode automatically reduces current levels until active and then impose a PUDELAY wait to allow current levels to rise from standby to normal levels.

When starting up using auto power-down mode, first use the normal mode startup procedure. Before starting scan operations, set PUDELAY to the large power-up value. Next, set APD in the POWER register. Finally, clear the PD0 and or PD1 bits for the required converters. Converters remain powered off until scanning goes active, at which time the large PUDELAY executes as the ADC goes from powered down to fully powered at the start of the scan.

In auto power-down mode, when the ADC goes from idle to active, a converter is only powered up if it is required for the scan, as determined by the ADLST1, ADLST2, and SDIS registers.

It is recommended to power-off both converters (PD0=PD1=1 in the POWER register) when re-configuring clocking or power controls to avoid generating bad samples and ensure proper delays are applied when powering up or starting scans.

Attempts to start a scan during the PUDELAY time-out are ignored until the appropriate PSTS*n* bits are cleared in the POWER register.

Any attempt to use a converter when powered down or with the voltage reference disabled results in invalid results. It is possible to read ADC result registers after converter power down to see results calculated before power-down. However, a new scan sequence must be started with a SYNC*n* pulse or a write to the START*n* bit before new results are available.

28.5.8.3 ADC STOP Mode of Operation

Any conversion sequence in progress can be stopped by setting the relevant STOP*n* bit. Any further sync pulses or writes to the START*n* bit are ignored until the STOP*n* bit is cleared. In this stop mode, the results registers can be modified by writes from the processor. Any write to ADRSLT*n* in the ADC stop mode is treated as if the analog core supplied the data, so limit checking, zero crossing, and associated interrupts can occur if enabled.

28.5.9 ADC Clock

28.5.9.1 General

The ADC has two external clock inputs used to drive two clock domains within the ADC module.

Table 28-22. ADC Clock Summary

Clock input	Source	Characteristics
Peripheral Clock (=System Clock)	1/2 Core clock	Maximum rate is PLL output divided by 2 if PLL enabled. When PLL disabled, max rate is oscillator clock divided by 2.
ADC 8MHz Clock	Relaxation Oscillator (8MHz), Crystal Oscillator (1-16MHz), or external Oscillator	Provides 8MHz for auto standby power saving mode.

28.5.9.2 Description of Clock Operation

As shown in Figure 28-25**,** the conversion clock is the primary source for the ADC clock and is always selected as the ADC clock when conversions are in process. The DIV value in the CTRL2 register should be configured so the conversion clock frequency falls between 100 kHz and 5.0 MHz. Operating the ADC at out-of-spec clock frequencies degrades conversion accuracy. Similarly, modifying the parameters affect clock rates or power modes while the regulators are powered up (PD0=0 or PD1=0) also degrades conversion accuracy.

The conversion clock ADC uses for sampling is calculated using the IPBus clock and the clock divisor bits within the CTRL2 register. Please see Section 28.4.1, “Control 1 Register (CTRL1)” or Section 28.4.2, “Control 2 Register (CTRL2)”**.** The ADC clock is active 100% of the time while in loop modes, or if power management is set to normal. It is also active during all ADC power-up for a period of time determined by the PUDELAY field in the power (POWER) register. After the power-up delay times out, the ADC clock continues until the completion of the ADC*n* scan when operating in auto standby or auto power-down modes.

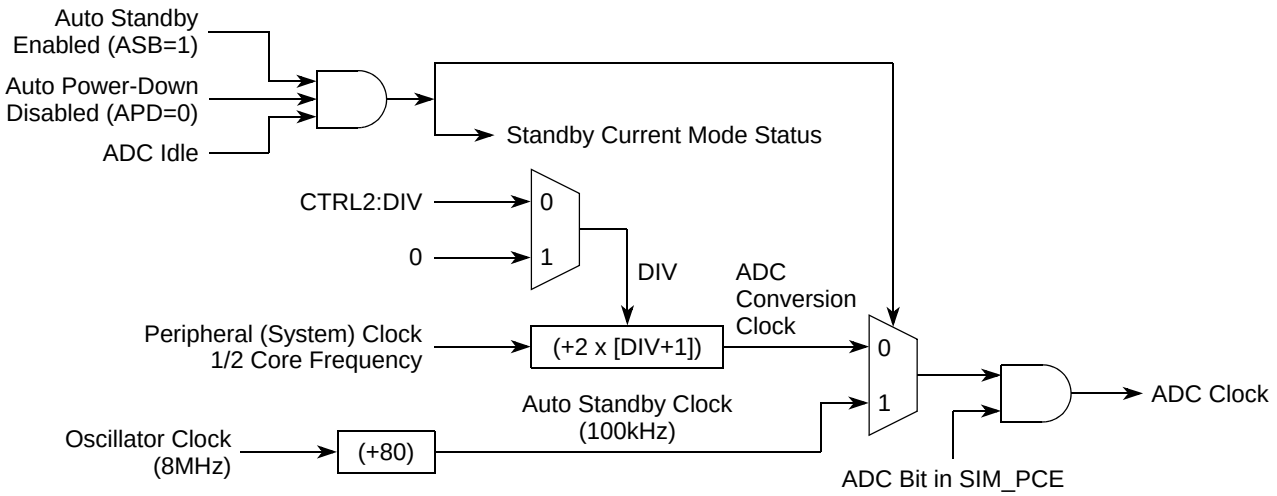


Figure 28-25. ADC Clock Generation

The oscillator clock feeds an 80:1 divider, generating the auto standby clock. The auto standby clock is selected as the ADC clock during the auto standby power mode when both converters are idle. The auto

standby power mode requires an 8 MHz oscillator clock from the relaxation oscillator, crystal oscillator, or external oscillator.

28.5.9.3 ADC Clock Resynchronization at Start of Scan

At the fastest ADC speed, each ADC clock period is 6 system clock periods long. When asserting the start of a scan, by writing to a START*n* bit or by a SYNC*n* signal, the ADC clock is re-synchronized to align it to the system clock. This allows the commanded scan to begin as soon as possible rather than wait up to 5 additional system clocks for the start of the next ADC clock period. This is shown in Figure 28-26 for sequential and simultaneous parallel modes of operation. In these modes, both ADCs operate off of the same start signal.

In a parallel scan mode when SIMULT equals 0, both ADCs operate using independent START*n* bits and SYNC*n* signals. As shown in Figure 28-27, the first scan started is re-synchronized to the system clock, but the second scan may wait up to 5 additional system clocks before starting. Also, which converter is synchronized to the system clock depends on which convert first starts to use the ADC. The case shown has ADCA synchronized, but one could easily imagine the case where the ADCA start comes after instead of before the ADCB start. In this case, ADCAs start would be delayed up to 5 additional system clock periods instead of ADCBs.

If there is a known timing relationship between ADCA and ADCB when operating in a non-simultaneous parallel mode, then the application can control which ADC starts first and gets the re-synchronized clock. The application can also control the delay to starting the second ADC scan so that its start signal aligns with the ADC clock, and the start of the second ADC is not delayed.

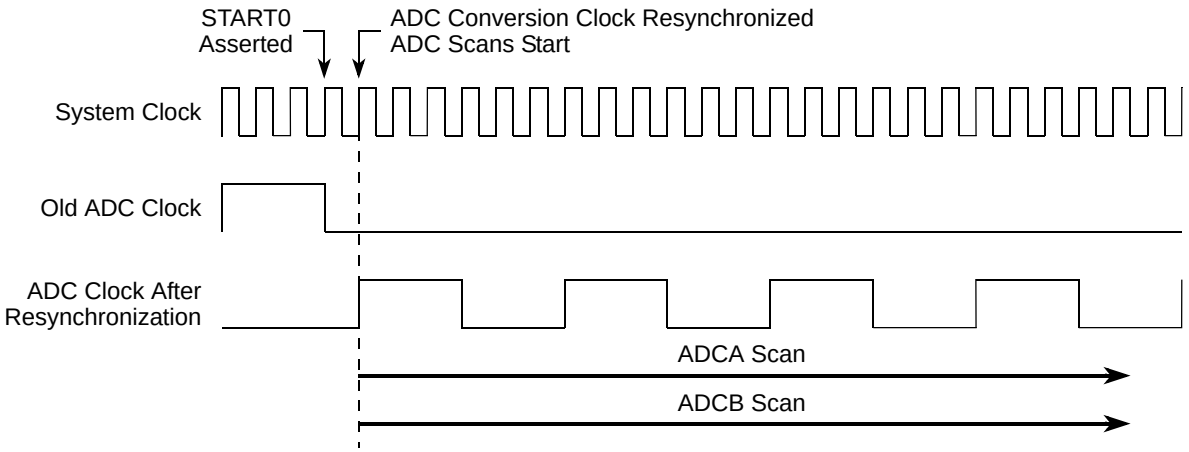


Figure 28-26. ADC Clock Resynchronization for Sequential and Simultaneous Parallel Modes

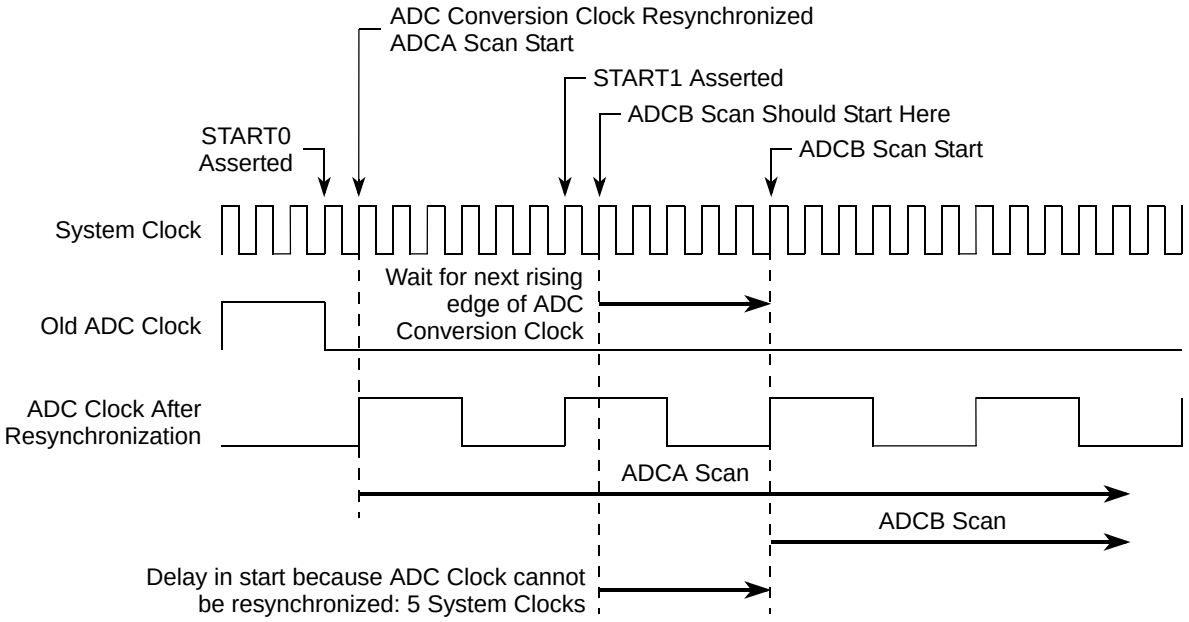


Figure 28-27. ADC Clock Resynchronization for Non-Simultaneous Parallel Modes

28.5.10 Voltage Reference Pins V_{REFH} and V_{REFL}

The voltage difference between V_{REFH} and V_{REFL} provides the reference voltage that all analog inputs are measured against. The reference voltage should be provided from a low noise filtered source capable of providing up to 1mA of reference current.

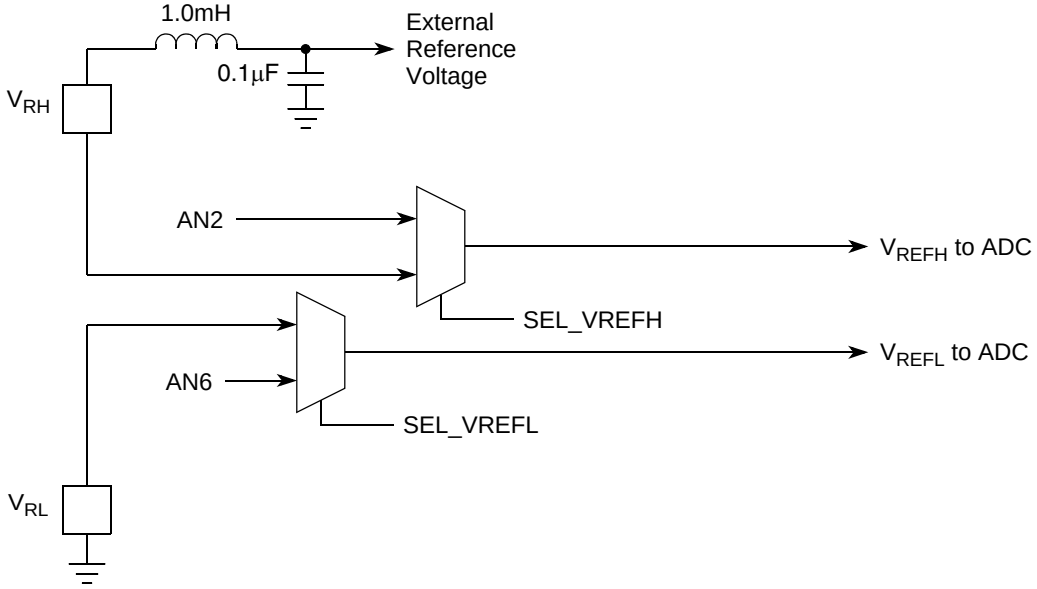


Figure 28-28. ADC Voltage Reference Circuit

When tying V_{REFH} to the same potential as V_{DDA}, relative measurements are being made with respect to the amplitude of V_{DDA}. It is imperative that special precautions be taken to assure the voltage applied to

V_{REFH} is as noise-free as possible. Any noise residing on the V_{REFH} voltage is directly transferred to the digital result.

Figure 28-28 illustrates the internal workings of the ADC voltage reference circuit. V_{REFH} must be noise filtered; a minimum configuration is shown in the figure.

28.5.11 Supply Pins V_{DDA} and V_{SSA}

Dedicated power supply pins are provided for the purposes of reducing noise coupling and to improve accuracy. The power provided to these pins is suggested to come from a low noise filtered source. Uncoupling capacitors ought to be connected between V_{DDA} and V_{SSA}.

Chapter 29
Pulse-Width Modulation (PWM) Module

29.1 Introduction

This chapter describes the configuration and operation of the pulse-width modulation (PWM) module. It includes a block diagram, programming model, and functional description.

29.1.1 Overview

The PWM module, shown in Figure 29-1, generates a synchronous series of pulses having programmable period and duty cycle. With a suitable low-pass filter, the PWM can be used as a digital-to-analog converter.

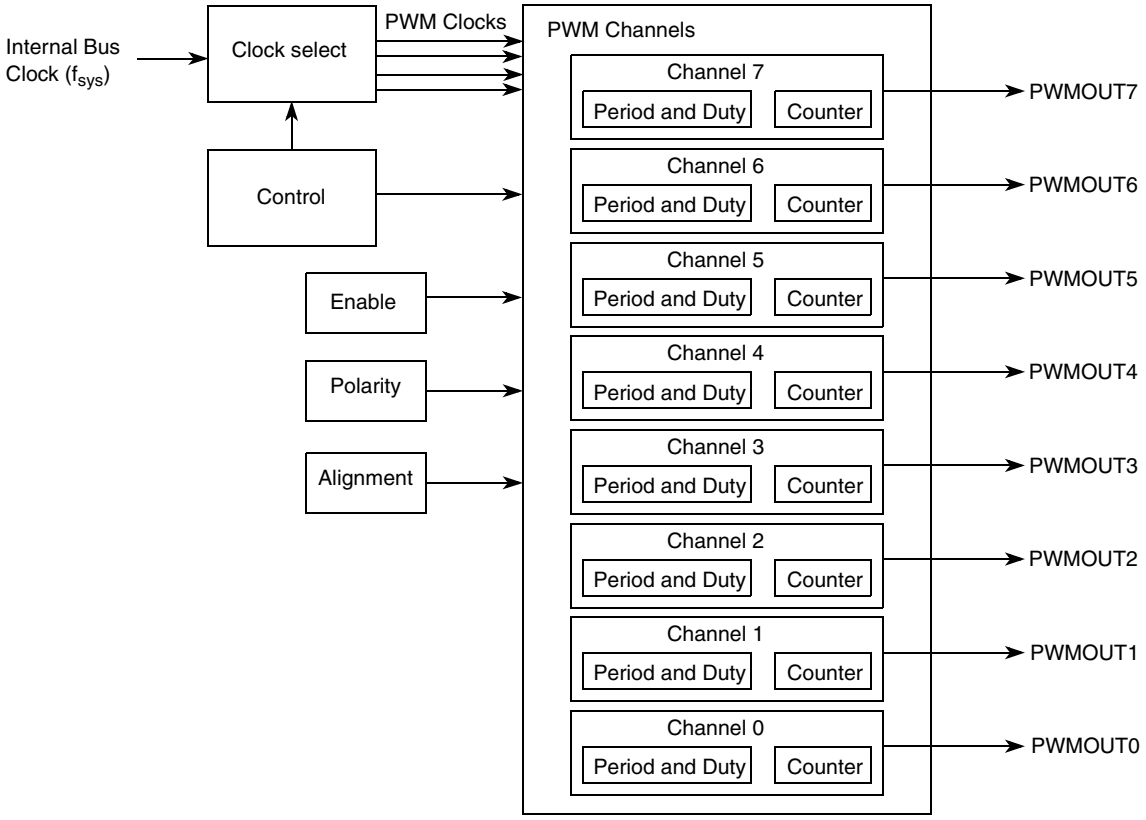


Figure 29-1. PWM Block Diagram

Main features include the following:

- Double-buffered period and duty cycle
- Left- or center-aligned outputs
- Eight independent PWM modules
- Byte-wide registers provide programmable duty cycle and period control
- Four programmable clock sources

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, "General Purpose I/O Module") prior to configuring the PWM module.

29.2 Memory Map/Register Definition

This section describes the registers and control bits in the PWM module. There are eight independent PWM modules, each with its own control and counter registers. The memory map for the PWM is shown below.

Table 29-1. PWM Memory Map

IPSBAR Offset[1,2]	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Read/Write Only Access					
0x1B_0000	PWM Enable Register (PWME)	8	R/W	0x00	29.2.1/29-3
0x1B_0001	PWM Polarity Register (PWMPOL)	8	R/W	0x00	29.2.2/29-4
0x1B_0002	PWM Clock Select Register (PWMCLK)	8	R/W	0x00	29.2.3/29-4
0x1B_0003	PWM Prescale Clock Select Register (PWMPRCLK)	8	R/W	0x00	29.2.4/29-5
0x1B_0004	PWM Center Align Enable Register (PWMCAE)	8	R/W	0x00	29.2.5/29-6
0x1B_0005	PWM Control Register (PWMCTL)	8	R/W	0x00	29.2.6/29-7
0x1B_0008	PWM Scale A Register (PWMSCLA)	8	R/W	0x00	29.2.7/29-8
0x1B_0009	PWM Scale B Register (PWMSCLB)	8	R/W	0x00	29.2.8/29-9
0x1B_000C + *n* *n* = 0–7	PWM Channel *n* Counter Register (PWMCNT*n*)	8	R/W	0x00	29.2.9/29-9
0x1B_0014 + *n* *n* = 0–7	PWM Channel *n* Period Register (PWMPER*n*)	8	R/W	0xFF	29.2.10/29-10
0x1B_001C + *n* *n* = 0–7	PWM Channel *n* Duty Register (PWMDTY*n*)	8	R/W	0xFF	29.2.11/29-11
0x1B_0024	PWM Shutdown Register (PWMSDN)	8	R/W	0x00	29.2.12/29-12

1 Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

2 A 32-bit access to any of these registers results in a bus transfer error.

29.2.1 PWM Enable Register (PWME)

Each PWM channel has an enable bit (PWME*n*) to start its waveform output. While in run mode, if all eight PWM output channels are disabled (PWME[7:0] = 0), the prescaler counter shuts off for power savings. See Section 29.3.2.1, “PWM Enable” for more information.

IPSBAR 0x1B_0000 (PWME) Access: Supervisor
Offset: Read/Write

	7	6	5	4	3	2	1	0
R W	PWME7	PWME6	PWME5	PWME4	PWME3	PWME2	PWME1	PWME0
Reset:	0	0	0	0	0	0	0	0

Figure 29-2. PWM Enable Register (PWME)

Table 29-2. PWME Field Descriptions

Field	Description
7 PWME7	PWM Channel 7 Enable. In normal mode, if enabled, the PWM signal becomes available at PWMOUT7 when its corresponding clock source begins its next cycle. When PWMSDN[SDNEN] is set this channel is an input for emergency shutdown. 0 PWM7 disabled 1 PWM7 enabled
6 PWME6	PWM Channel 6 Output Enable. If enabled, the PWM signal becomes available at PWMOUT6 when its corresponding clock source begins its next cycle. If PWMCTL[CON67] is set, then this bit has no effect and PWMOUT6 is disabled. 0 PWM output disabled 1 PWM output enabled
5 PWME5	PWM Channel 5 Output Enable. If enabled, the PWM signal becomes available at PWMOUT5 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
4 PWME4	PWM Channel 4 Output Enable. If enabled, the PWM signal becomes available at PWMOUT4 when its corresponding clock source begins its next cycle. If PWMCTL[CON45] is set, then this bit has no effect and PWMOUT4 is disabled. 0 PWM output disabled 1 PWM output enabled
3 PWME3	PWM Channel 3 Output Enable. If enabled, the PWM signal becomes available at PWMOUT3 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
2 PWME2	PWM Channel 2 Output Enable. If enabled, the PWM signal becomes available at PWMOUT2 when its corresponding clock source begins its next cycle. If PWMCTL[CON23] is set, then this bit has no effect and PWMOUT2 is disabled. 0 PWM output disabled 1 PWM output enabled, if PWMCTL[CON23]=0

Table 29-2. PWME Field Descriptions (continued)

Field	Description
1 PWME1	PWM Channel 1 Output Enable. If enabled, the PWM signal becomes available at PWMOUT1 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
0 PWME0	PWM Channel 0 Output Enable. If enabled, the PWM signal becomes available at PWMOUT0 when its corresponding clock source begins its next cycle. If PWMCTL[CON01] is set, then this bit has no effect and PWMOUT0 is disabled. 0 PWM output disabled 1 PWM output enabled, if PWMCTL[CON01]=0

29.2.2 PWM Polarity Register (PWMPOL)

The starting polarity of each PWM channel waveform is determined by the associated PWMPOL[PPOL*n*] bit. If the polarity is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

IPSBAR 0x1B_0001 (PWMPOL) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R W	PPOL7	PPOL6	PPOL5	PPOL4	PPOL3	PPOL2	PPOL1	PPOL0
Reset:	0	0	0	0	0	0	0	0

Figure 29-3. PWM Polarity Register (PWMPOL)

Table 29-3. PWMPOL Field Descriptions

Field	Description
7–0 PPOL*n*	PWM Channel *n* Polarity. The even-numbered channels' polarity has no effect when the corresponding PWMCTL[CON*n(n+1)*] bit is set. For example, if PWMCTL[CON01] equals 1, PWMPOL[PPOL0] has no affect. 0 PWM channel *n* output is low at the beginning of the period, then goes high when the duty count is reached 1 PWM channel *n* output is high at the beginning of the period, then goes low when the duty count is reached

29.2.3 PWM Clock Select Register (PWMCLK)

Each PWM channel has the capability of selecting one of two clocks. For channels0, 1, 4, and 5, the clock choices are clock A or SA. For channels2, 3, 6, and 7, the choices are clock B or SB. The clock selection is done with the below PWMCLK[PCLK*n*] control bits. If a clock select is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

IPSBAR 0x1B_0002 (PWMCLK) Access: Supervisor
Offset: Read/Write

	7	6	5	4	3	2	1	0
R W	PCLK7	PCLK6	PCLK5	PCLK4	PCLK3	PCLK2	PCLK1	PCLK0
Reset:	0	0	0	0	0	0	0	0

Figure 29-4. PWM Clock Select Register (PWMCLK)

Table 29-4. PWMCLK Field Descriptions

Field	Description
7–0 PCLK*n*	PWM channel *n* clock select. Selects between one of two clock sources for each PWM channel. See Section 29.2.4, "PWM Prescale Clock Select Register (PWMPRCLK)" and Section 29.2.7, "PWM Scale A Register (PWMSCLA)" for more information on how the different clock rates are generated. The even-numbered channels' clock select has no effect when the corresponding PWMCTL[CON*n(n+1)*] bit is set. For example, if PWMCTL[CON01] equals 1, PWMCLK[PCLK0] has no affect.

	PCLK6 & PCLK7 (PWM6 & PWM7 Clock Source)	PCLK4 & PCLK5 (PWM4 & PWM5 Clock Source)	PCLK2 & PCLK3 (PWM2 & PWM3 Clock Source)	PCLK0 & PCLK1 (PWM0 & PWM1 Clock Source)
0	B	A	B	A
1	SB	SA	SB	SA

29.2.4 PWM Prescale Clock Select Register (PWMPRCLK)

The PWMPRCLK register selects the prescale clock source for clocks A and B independently. If the clock prescale is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

IPSBAR 0x1B_0003 (PWMPRCLK) Access:
Offset: SupervisorRead/Write

	7	6	5	4	3	2	1	0
R	0	PCKB			0	PCKA		
W								
Reset:	0	0	0	0	0	0	0	0

Figure 29-5. PWM Prescale Clock Select Register (PWMPRCLK)

Table 29-5. PWMPRCLK Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6–4
PCKB</td><td>Clock B prescaler select. These three bits control the rate of Clock B, which can be used for PWM channels2, 3, 6 and 7.
<table>
<tr><th>PCKB</th><th>Clock B Rate</th></tr>
<tr><td>000</td><td>Internal bus clock ÷ 2^0</td></tr>
<tr><td>001</td><td>Internal bus clock ÷ 2^1</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>111</td><td>Internal bus clock ÷ 2^7</td></tr>
</table></td></tr>
<tr><td>3</td><td>Reserved, must be cleared.</td></tr>
<tr><td>2–0
PCKA</td><td>Clock A prescaler select. These three bits control the rate of Clock A, which can be used for PWM channels0, 1, 4 and 5.
<table>
<tr><th>PCKA</th><th>Clock A Rate</th></tr>
<tr><td>000</td><td>Internal bus clock ÷ 2^0</td></tr>
<tr><td>001</td><td>Internal bus clock ÷ 2^1</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>111</td><td>Internal bus clock ÷ 2^7</td></tr>
</table></td></tr>
</table>

29.2.5 PWM Center Align Enable Register (PWMCAE)

The PWMCAE register contains eight control bits for the selection of center-aligned outputs or left-aligned outputs for each PWM channel. Write these bits only when the corresponding channel is disabled. See Section 29.3.2.5, “Left-Aligned Outputs” and Section 29.3.2.6, “Center-Aligned Outputs” for a more detailed description of the PWM output modes.

IPSBAR 0x1B_0004 (PWMCAE) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R W	CAE7	CAE6	CAE5	CAE4	CAE3	CAE2	CAE1	CAE0
Reset:	0	0	0	0	0	0	0	0

Figure 29-6. PWM Center Align Enable Register (PWMCAE)

Table 29-6. PWMCAE Field Descriptions

Field	Description
7–0 CAE*n*	Center align enable for channel *n*. The even-numbered channels' center align enable has no effect when the corresponding PWMCTL[CON*n(n+1)*] bit is set. For example, if PWMCTL[CON01] equals 1, PWMCAE[CAE0] has no affect. 0 Channel *n* operates in left-aligned output mode 1 Channel *n* operates in center-aligned output mode

29.2.6 PWM Control Register (PWMCTL)

The PWMCTL register provides various control of the PWM module. Change the CON*n(n+1)* bits only when both corresponding channels are disabled. See Section 29.3.2.7, "PWM 16-Bit Functions" for a more detailed description of the concatenation function.

IPSBAR 0x1B_0005 (PWMCTL) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 29-7. PWM Control Register (PWMCTL)

Table 29-7. PWMCTL Field Descriptions

Field	Description
7 CON67	Concatenates PWM channels 6 and 7 to form one 16-bit PWM channel. 0 Channels 6 and 7 are separate 8-bit PWMs 1 Concatenate PWM 6 and 7. Channel 6 becomes the high order byte and channel 7 the low order byte. PWMOUT7 is the output for this 16-bit PWM signal, and PWMOUT6 is disabled. The channel 7 clock select, polarity, center align enable, and enable bits control this concatenated output.
6 CON45	Concatenates PWM channels 4 and 5 to form one 16-bit PWM channel. 0 Channels 4 and 5 are separate 8-bit PWMs 1 Concatenate PWM 4 and 5. Channel 4 becomes the high order byte and channel 5 the low order byte. PWMOUT5 is the output for this 16-bit PWM signal, and PWMOUT4 is disabled. The channel 5 clock select, polarity, center align enable, and enable bits control this concatenated output.
5 CON23	Concatenates PWM channels 2 and 3 to form one 16-bit PWM channel. 0 Channels 2 and 3 are separate 8-bit PWMs 1 Concatenate PWM 2 and 3. Channel 2 becomes the high order byte and channel 3 the low order byte. PWMOUT3 is the output for this 16-bit PWM signal, and PWMOUT2 is disabled. The channel 3 clock select, polarity, center align enable, and enable bits control this concatenated output.
4 CON01	Concatenates PWM channels 0 and 1 to form one 16-bit PWM channel. 0 Channels 0 and 1 are separate 8-bit PWMs 1 Concatenate PWM 0 and 1. Channel 0 becomes the high order byte and channel 1 the low order byte. PWMOUT1 is the output for this 16-bit PWM signal, and PWMOUT0 is disabled. The channel 1 clock select, polarity, center align enable, and enable bits control this concatenated output.

Table 29-7. PWMCTL Field Descriptions (continued)

Field	Description
3 PSWAI	PWM stops in doze mode. Disables the input clock to the prescaler while in doze mode. 0 Allow the clock to the prescaler while in doze mode 1 Stop the input clock to the prescaler when the core is in doze mode
2 PFRZ	PWM counters stop in debug mode ($\overline{\text{BKPT}}$ asserted). 0 Allow PWM counters to continue while in debug mode 1 Disable PWM input clock to the prescaler when the core is in debug mode. Useful for emulation as it allows the PWM function to be suspended.
1–0	Reserved, must be cleared.

29.2.7 PWM Scale A Register (PWMSCLA)

PWMSCLA is the programmable scale value used in scaling clock A to generate clock SA. Clock SA is generated with the following equation:

$$\text{Clock SA} = \frac{\text{Clock A}}{2 \times \text{PWMSCLA}} \qquad \textit{Eqn. 29-1}$$

Any value written to this register causes the scale counter to load the new scale value (PWMSCLA).

IPSBAR 0x1B_0008 (PWMSCLA) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R W	SCALEA							
Reset:	0	0	0	0	0	0	0	0

Figure 29-8. PWM Scale A Register (PWMSCLA)

Table 29-8. PWMSCLA Field Descriptions

Field	Description
7–0 SCALEA	Part of divisor used to form Clock SA from Clock A. SCALEA / Value: 0x00 — 256 0x01 — 1 0x02 — 2 ... — ... 0xFF — 255

29.2.8 PWM Scale B Register (PWMSCLB)

PWMSCLB is the programmable scale value used in scaling clock B to generate clock SB. Clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}} \qquad \textit{Eqn. 29-2}$$

Any value written to this register causes the scale counter to load the new scale value (PWMSCLB).

IPSBAR 0x1B_0009 (PWMSCLB) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R W	SCALEB							
Reset:	0	0	0	0	0	0	0	0

Figure 29-9. PWM Scale B Register (PWMSCLB)

Table 29-9. PWMSCLB Field Descriptions

Field	Description
7–0 SCALEB	Divisor used to form Clock SB from Clock B. **SCALEB** / **Value** 0x00 / 256 0x01 / 1 0x02 / 2 ... / ... 0xFF / 255

29.2.9 PWM Channel Counter Registers (PWMCNT*n*)

Each channel has a dedicated 8-bit up/down counter that runs at the rate of the selected clock source, PWMCLK[PCLK*n*]. The user can read the counters at any time without affecting the count or the operation of the PWM channel. In left-aligned output mode, the counter counts from 0 to the value in the period register minus 1. In center-aligned output mode, the counter counts from 0 up to the value in the period register and then back down to 0. Therefore, given the same value in the period register, center-aligned mode is twice the period of left-aligned mode.

Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up for center-aligned mode, the immediate load of duty and period registers with values from the buffers, and the output to change according to the polarity bit.

The counter is also cleared at the end of the effective period (see Section 29.3.2.5, “Left-Aligned Outputs” and Section 29.3.2.6, “Center-Aligned Outputs” for more details). When the channel is disabled

(PWME*n*=0), the PWMCNT*n* register does not count. When a channel is enabled (PWME*n*=1), the associated PWM counter starts at the count in the PWMCNT*n* register. For more detailed information on the operation of the counters, refer to Section 29.3.2.4, “PWM Timer Counters.”

IPSBAR 0x1B_000C (PWMCNT0)
Offset: 0x1B_000D (PWMCNT1)
0x1B_000E (PWMCNT2)
0x1B_000F (PWMCNT3)
0x1B_0010 (PWMCNT4)
0x1B_0011 (PWMCNT5)
0x1B_0012 (PWMCNT6)
0x1B_0013 (PWMCNT7)

Access: SupervisorRead/Write

	7	6	5	4	3	2	1	0
R W	COUNT							
Reset:	0	0	0	0	0	0	0	0

Figure 29-10. PWM Counter Registers (PWMCNT*n*)

Table 29-10. PWMCNT*n* Field Descriptions

Field	Description
7–0 COUNT	Current value of the PWM up counter. Resets to zero when written.

29.2.10 PWM Channel Period Registers (PWMPER*n*)

The PWM period registers determine the period of the associated PWM channel. Refer to Section 29.3.2.3, “PWM Period and Duty” for more information.

Calculating the output period depends on the output mode (center-aligned has twice the period as left-aligned mode) as well as PWMPER*n*. See the below equation:

$$\text{PWM}n\text{ period} = \text{Channel clock period} \times (\text{PWMCAE}[\text{CAE}n] + 1) \times \text{PWMPER}n \qquad \textit{Eqn. 29-3}$$

For boundary case programming values (e.g. PWMPER*n* = 0x00), please refer to Section 29.3.2.8, “PWM Boundary Cases”.

IPSBAR 0x1B_0014 (PWMPER0)
Offset: 0x1B_0015 (PWMPER1)
0x1B_0016 (PWMPER2)
0x1B_0017 (PWMPER3)
0x1B_0018 (PWMPER4)
0x1B_0019 (PWMPER5)
0x1B_001A (PWMPER6)
0x1B_001B (PWMPER7)

Access: SupervisorRead/Write

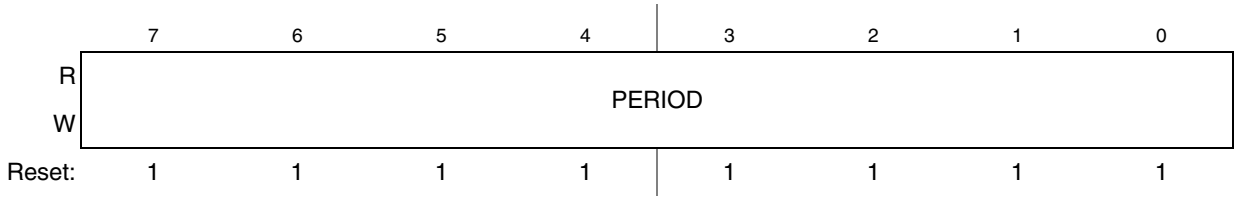


Figure 29-11. PWM Period Registers (PWMPER*n*)

Table 29-11. PWMPER*n* Field Descriptions

Field	Description
7–0 PERIOD	Period counter for the output PWM signal. If PERIOD equals 0x00, the PWM*n* output is always high (PPOL*n*=1) or always low (PPOL*n*=0). See Section 29.3.2.8, “PWM Boundary Cases” for other special cases.

29.2.11 PWM Channel Duty Registers (PWMDTY*n*)

The PWM duty registers determine the duty cycle of the associated PWM channel. To calculate the output duty cycle (high time as a percentage of period) for a particular channel:

$$\text{Duty Cycle} = \left|\left(1 - \text{PWMPOL}[\text{PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n}\right)\right| \times 100\% \qquad \textit{Eqn. 29-4}$$

For boundary case programming values (e.g. PWMDTY*n* = 0x00 or PWMDTY*n* > PWMPER*n*), refer to Section Section 29.3.2.8, “PWM Boundary Cases”.

IPSBAR 0x1B_001C (PWMDTY0)
Offset: 0x1B_001D (PWMDTY1)
0x1B_001E (PWMDTY2)
0x1B_001F (PWMDTY3)
0x1B_0020 (PWMDTY4)
0x1B_0021 (PWMDTY5)
0x1B_0022 (PWMDTY6)
0x1B_0023 (PWMDTY7)

Access: SupervisorRead/Write

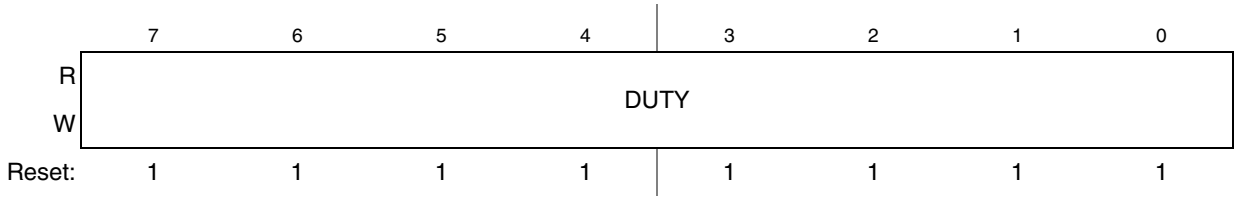


Figure 29-12. PWM Duty Registers (PWMDTY*n*)

Table 29-12. PWMDTY*n* Field Descriptions

Field	Description
7–0 DUTY	Contains the duty value used to determine when a transition occurs on the PWM output signal. When a match occurs with the corresponding PWMCNT*n* register, the PWM output toggles. If DUTY equals 0x00, the PWM*n* output is always low (PPOL*n*=1) or always high (PPOL*n*=0). See Section 29.3.2.8, "PWM Boundary Cases" for other special cases.

29.2.12 PWM Shutdown Register (PWMSDN)

The PWM shutdown register provides emergency shutdown functionality of the PWM module. The PWMSDN[7:1] bits are ignored if PWMSDN[SDNEN] is cleared.

IPSBAR 0x1B_0024 (PWMSDN) Access: SupervisorRead/Write
Offset:

	7	6	5	4	3	2	1	0
R	IF	IE	0	LVL	0	PWM7IN	PWM7IL	SDNEN
W	w1c		RESTART					
Reset:	0	0	0	0	0	0	0	0

Figure 29-13. PWM Shutdown Register (PWMSDN)

Table 29-13. PWMSDN Field Descriptions

Field	Description
7 IF	PWM interrupt flag. Any change in state of PWM7IN is flagged by setting this bit. The flag is cleared by writing a 1 to it. Writing 0 has no effect. 0 No change in PWM7IN input 1 Change in PWM7IN input
6 IE	PWM interrupt enable. An interrupt is triggered to the device's interrupt controller when PWMSDN[IF] is set. 0 Interrupt is disabled 1 Interrupt is enabled
5 RESTART	PWM restart. After setting the RESTART bit, the PWM channels start running after the corresponding counter resets to zero. Also, if emergency shutdown is cleared (after being set), the PWM outputs restart after the corresponding counter resets to zero. This bit is self-clearing, so is always read as zero.
4 LVL	PWM shutdown output level. Describes the behavior of the PWM outputs when PWM7IN input is asserted and PWMSDN[SDNEN] is set. 0 PWM outputs are forced to logic 0 1 PWM outputs are forced to logic 1
3	Reserved, must be cleared.
2 PWM7IN	PWM channel 7 input status. Reflects the current status of the PWMOUT7 pin. Read only.

Table 29-13. PWMSDN Field Descriptions (continued)

Field	Description
1 PWM7IL	PWM channel 7 input polarity. If PWMSDN[SDNEN] is set, this bit sets the active level of the PWM 7 channel 0 PWM 7 input is active low 1 PWN 7 input is active high
0 SDNEN	PWM emergency shutdown enable. If set, the pin associated with PWM channel 7 is forced to input and the emergency shutdown is enabled. 0 Emergency shutdown is disabled 1 Emergency shutdown is enabled

29.3 Functional Description

29.3.1 PWM Clock Select

There are four available clocks—clock A, B, SA (scaled A), and SB (scaled B)—all based on the internal bus clock.

Clock A and B can be programmed to run at 1, 1/2,..., 1/128 times the internal bus clock. Clock SA and SB use clock A and B respectively as an input and divide it further with a reloadable counter. The rates available for clock SA and SB are programmable to run at clock A and B divided by 2, 4,..., or 512. Each PWM channel has the capability of selecting one of two clocks, the prescaled clock (clock A or B) or the scaled clock (clock SA or SB). The block diagram in Figure 29-14 shows the four different clocks and how the scaled clocks are created.

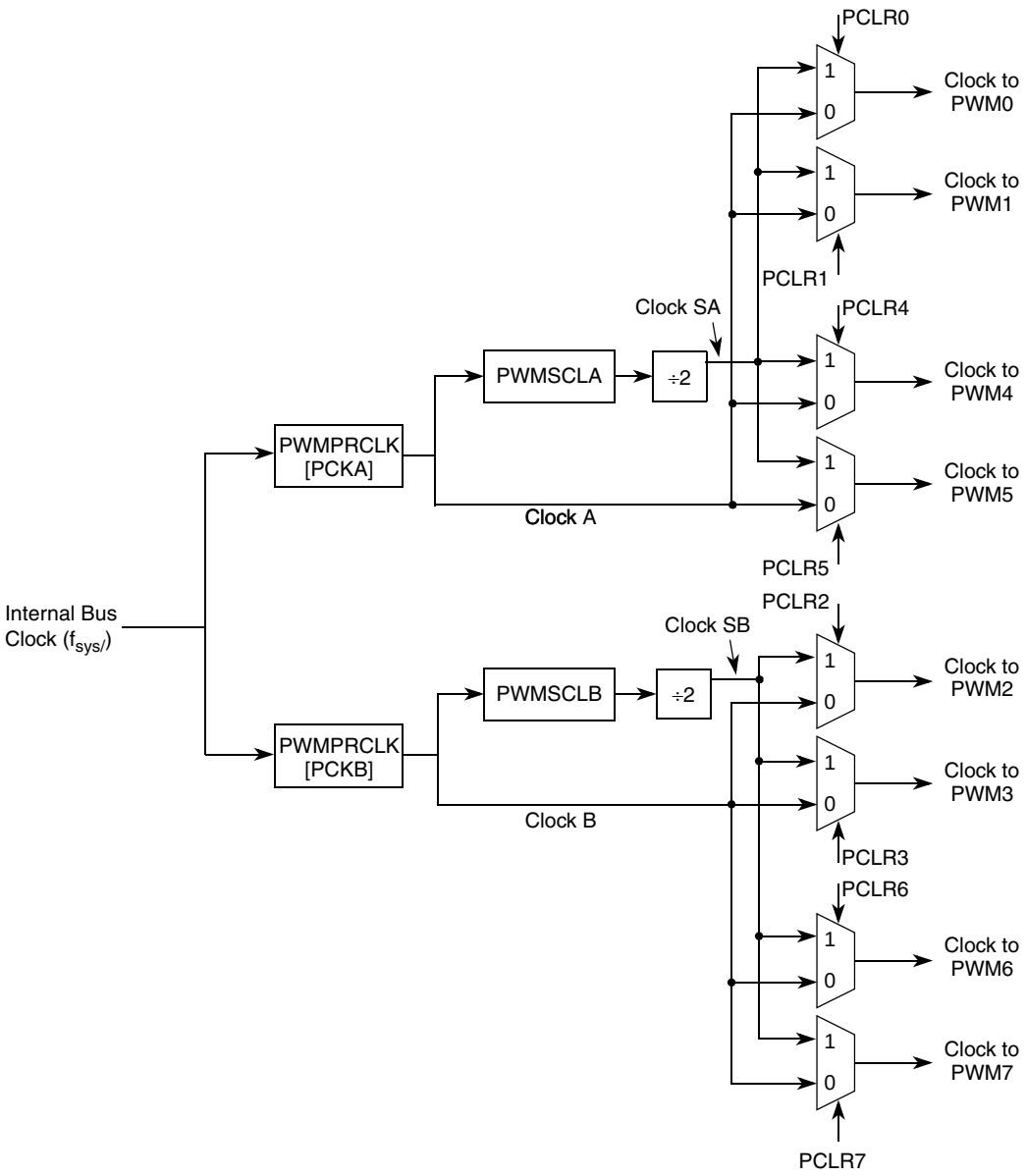


Figure 29-14. PWM Clock Select Block Diagram

29.3.1.1 Prescaled Clock (A or B)

The internal bus clock is the input clock to the PWM prescaler that can be disabled when the device is in debug mode by setting the PWMCTL[PFRZ] bit. This is useful for reducing power consumption and for emulation to freeze the PWM. The input clock is also disabled when all PWM channels are disabled (PWME*n*=0).

Clock A and B are scaled values of the input clock. The value is software selectable for clock A and B and has options of 1, 1/2,..., or 1/128 times the internal bus clock. The value selected for clock A and B is determined by the PWMPRCLK[PCKA*n*] and PWMPRCLK[PCKB*n*] bits.

29.3.1.2 Scaled Clock (SA or SB)

The scaled A (SA) and scaled B (SB) clocks use clock A and B respectively as inputs, divide it further with a user programmable value, then divide this by 2. The rates available for clock SA are programmable to run at clock A divided by 2, 4,..., or 512. Similar rates are available for clock SB.

Clock SA equals clock A divided by two times the value in the PWMSCLA register:

$$\text{Clock SA} = \frac{\text{Clock A}}{2 \times \text{PWMSCLA}} \qquad \textit{Eqn. 29-5}$$

Similarly, clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}} \qquad \textit{Eqn. 29-6}$$

As an example, consider the case in which the user writes 0xFF into the PWMSCLA register. Clock A for this case is selected to be internal bus clock divided by 4. A pulse occurs at a rate of once every 255×4 bus cycles. Passing this through the divide by two circuit produces a clock signal of the internal bus clock divided by 2040. Similarly, a value of 0x01 in the PWMSCLA register when clock A is internal bus clock divided by 4 produces an internal bus clock divided by 8 rate.

Writing to PWMSCLA or PWMSCLB causes the associated 8-bit down counter to be re-loaded. Otherwise, when changing rates, the counter would have to count down to 0x01 before counting at the proper rate. Forcing the associated counter to re-load the scale register value every time PWMSCLA or PWMSCLB is written prevents this.

Writing to the scale registers while channels are operating can cause irregularities in the PWM outputs.

29.3.1.3 Clock Select

Each PWM channel has the capability of selecting one of two clocks. For channels 0, 1, 4, and 5 the clock choices are clock A or SA. For channels 2, 3, 6 and 7, the choices are clock B or SB. The clock selection is done with the PWMCLK[PCLK*x*] control bits.

Changing clock control bits while channels are operating can cause irregularities in the PWM outputs.

29.3.2 PWM Channel Timers

The main part of the PWM module is the actual timers. Each of the timer channels has a counter, a period register, and a duty register (each are 8-bit). The waveform output period is controlled by a match between the period register and the value in the counter. The duty is controlled by a match between the duty register and the counter value and causes the state of the output to change during the period. The starting polarity of the output is also selectable on a per channel basis. Figure 29-15 shows a block diagram for a PWM timer.

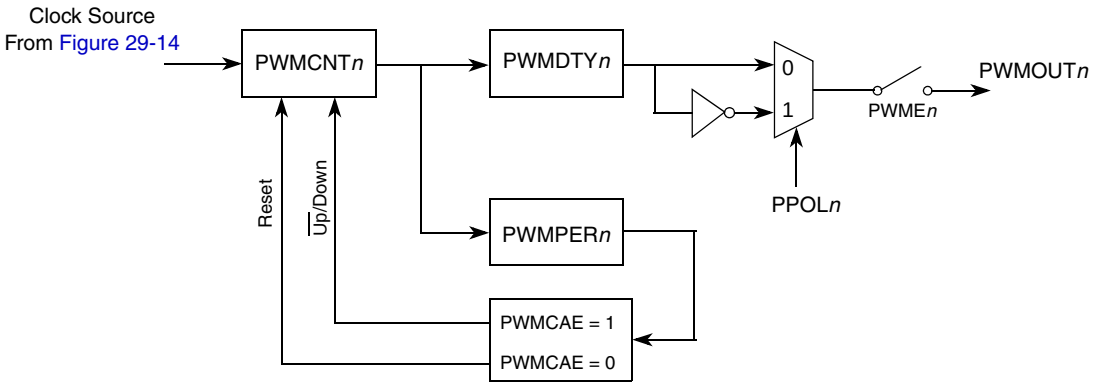


Figure 29-15. PWM Timer Channel Block Diagram

29.3.2.1 PWM Enable

Each PWM channel has an enable bit (PWME*n*) to start its waveform output. When any of the PWME*n* bits are set (PWME*n*=1), the associated PWM output signal is enabled immediately. However, the actual PWM waveform is not available on the associated PWM output until its clock source begins its next cycle; this is due to the synchronization of PWME*n* and the clock source. An exception is when channels are concatenated. Refer to Section 29.3.2.7, “PWM 16-Bit Functions” for more detail.

The first PWM cycle after enabling the channel can be irregular. When the channel is disabled (PWME*n*=0), the counter for the channel does not count.

29.3.2.2 PWM Polarity

Each channel has a polarity bit to allow starting a waveform cycle with a high or low signal. This is shown on the block diagram as a mux select. When one of the bits in the PWMPOL register is set, the associated PWM channel output is high at the beginning of the waveform, then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

29.3.2.3 PWM Period and Duty

Dedicated period and duty registers exist for each channel and are double buffered so that if they change while the channel is enabled, the change does not take effect until one of the following occurs:

- The effective period ends
- The PWMCNT*n* register is written (counter resets to 0x00)
- The channel is disabled, PWME*n* = 0

In this way, the output of the PWM is always the old waveform or the new waveform, not some variation in between. If the channel is not enabled, writes to the period and duty registers go directly to the latches as well as the buffer.

A change in duty or period can be forced into effect immediately by writing the new value to the duty and/or period registers and then writing to the counter. This forces the counter to reset and the new duty

and/or period values to be latched. In addition, because the counter is readable, it is possible to know where the count is with respect to the duty value, and software can be used to make adjustments. When forcing a new period or duty into effect immediately, an irregular PWM cycle can occur.

Depending on the polarity bit, the duty registers contain the count of the high time or the low time.

29.3.2.4 PWM Timer Counters

Each channel has a dedicated 8-bit up/down counter that runs at the rate of the selected clock source (see Figure 29-14 for the available clock sources and rates). The counter compares to two registers, a duty register and a period register, as shown in Figure 29-15. When the PWM counter matches the duty register, the output flip-flop changes state, causing the PWM waveform to also change state. A match between the PWM counter and the period register behaves differently depending on what output mode is selected as shown in Figure 29-15 and described in Section 29.3.2.5, “Left-Aligned Outputs” and Section 29.3.2.6, “Center-Aligned Outputs.”

Each channel counter can be read at anytime without affecting the count or the operation of the PWM channel.

Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up, the immediate load of duty and period registers with values from the buffers, and the output to change according to the polarity bit. When the channel is disabled (PWME*n* = 0), the counter stops. When a channel becomes enabled (PWME*n* = 1), the associated PWM counter continues from the count in the PWMCNT*n* register. This allows the waveform to continue where it left off when the channel is re-enabled. When the channel is disabled, writing 0 to the period register causes the counter to reset on the next selected clock.

NOTE

If the user wants to start a new clean PWM waveform without any history from the old waveform, the user must write to channel counter (PWMCNT*n*) prior to enabling the PWM channel (PWME*n* = 1).

Generally, writes to the counter are done prior to enabling a channel to start from a known state. However, writing a counter can also be done while the PWM channel is enabled (counting). The effect is similar to writing the counter when the channel is disabled, except that the new period is started immediately with the output set according to the polarity bit. Writing to the counter while the channel is enabled can cause an irregular PWM cycle to occur.

The counter is cleared at the end of the effective period (see Section 29.3.2.5, “Left-Aligned Outputs” and Section 29.3.2.6, “Center-Aligned Outputs” for more details).

Table 29-14. PWM Timer Counter Conditions

Counter Clears (0x00)	Counter Counts	Counter Stops
When PWMCNT*n* register written to any value	When PWM channel is enabled (PWME*n* = 1). Counts from last value in PWMCNT*n*.	When PWM channel is disabled (PWME*n* = 0)
Effective period ends		

29.3.2.5 Left-Aligned Outputs

The PWM timer provides the choice of two types of outputs: left- or center-aligned. They are selected with the PWMCAE[CAE*n*] bits. If the CAE*n* bit is cleared, the corresponding PWM output is left-aligned.

In left-aligned output mode, the 8-bit counter is configured as an up counter only. It compares to two registers, a duty register and a period register, as shown in the block diagram in Figure 29-15. When the PWM counter matches the duty register, the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register resets the counter and the output flip-flop, as shown in Figure 29-16, as well as performing a load from the double buffer period and duty register to the associated registers, as described in Figure 29.3.2.3. The counter counts from 0 to the value in the period register minus 1.

NOTE

Changing the PWM output mode from left-aligned to center-aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.

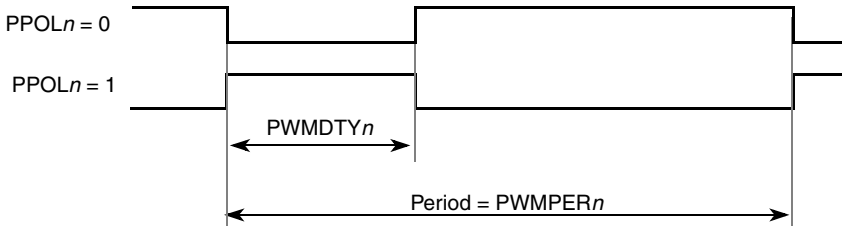


Figure 29-16. PWM Left-Aligned Output Waveform

To calculate the output frequency in left-aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by the value in the period register for that channel.

$$\text{PWM}n\text{ frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{\text{PWMPER}n} \qquad \textit{Eqn. 29-7}$$

The PWM*n* duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left(1 - \text{PWMPOL}[\text{PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n}\right) \times 100\% \qquad \textit{Eqn. 29-8}$$

29.3.2.5.1 Left-Aligned Output Example

As an example of a left-aligned output, consider the following case:

Clock source = internal bus clock, where internal bus clock = 40 MHz (25 ns period)

PPOL*n* = 0, PWMPER*n* = 4, PWMDTY*n* = 1

PWM*n* frequency = 40 MHz ÷ 4 = 10 MHz

PWM*n* period = 100 ns

$$\text{PWMn Duty Cycle} = \left(1 - \frac{1}{4}\right) \times 100\% = 75\%$$

The output waveform generated is below:

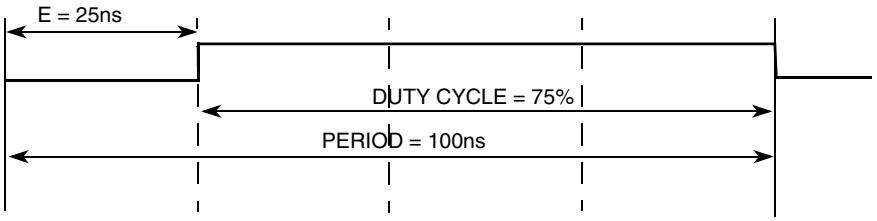


Figure 29-17. PWM Left-Aligned Output Example Waveform

29.3.2.6 Center-Aligned Outputs

For center-aligned output mode selection, set the PWMCAE[CAE*n*] bit and the corresponding PWM output is center-aligned.

The 8-bit counter operates as an up/down counter in this mode and is set to up when the counter is equal to 0x00. The counter compares to two registers, a duty register and a period register, as shown in the block diagram in Figure 29-15. When the PWM counter matches the duty register, the output flip-flop changes state, causing the PWM waveform to also change state. A match between the PWM counter and the period register changes the counter direction from an up-count to a down-count. When the PWM counter decrements and matches the duty register again, the output flip-flop changes state causing the PWM output to also change state. When the PWM counter decrements and reaches zero, the counter direction changes from a down-count back to an up-count, and a load from the double buffer period and duty registers to the associated registers is performed as described in Figure 29.3.2.3. The counter counts from 0 up to the value in the period register and then back down to 0. Thus the effective period is PWMPER*n* × 2.

Changing the PWM output mode from left-aligned output to center-aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.

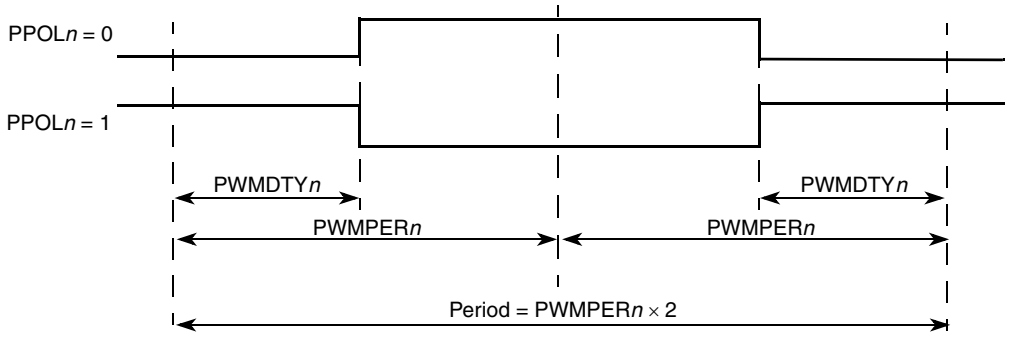


Figure 29-18. PWM Center-Aligned Output Waveform

To calculate the output frequency in center-aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by twice the value in the period register for that channel.

$$\text{PWM}n\text{ frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{2 \times \text{PWMPER}n} \qquad \textit{Eqn. 29-9}$$

The PWM*n* duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left(1 - \text{PWMPOL}[\text{PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n}\right) \times 100\% \qquad \textit{Eqn. 29-10}$$

29.3.2.6.1 Center-Aligned Output Example

As an example of a center-aligned output, consider the following case:

Clock source = internal bus clock, where internal bus clock = 40 MHz (25 ns period)

PPOL*n* = 0, PWMPER*n* = 4, PWMDTY*n* = 1

PWM*n* frequency = 40 MHz / (2×4) = 5 MHz

PWM*n* period = 200 ns

$$\text{PWMn Duty Cycle} = \left(1 - \frac{1}{4}\right) \times 100\% = 75\%$$

Shown below is the generated output waveform.

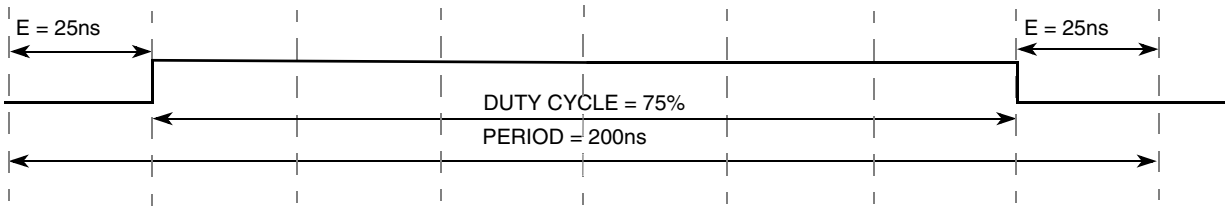


Figure 29-19. PWM Center-Aligned Output Example Waveform

29.3.2.7 PWM 16-Bit Functions

The PWM timer also has the option of generating eight 8-bit channels or four 16-bit channels for greater PWM resolution. This 16-bit channel option is achieved through the concatenation of two 8-bit channels.

The PWMCTL register contains four concatenation control bits, each used to concatenate a pair of PWM channels into one 16-bit channel. Channels 0 and 1 are concatenated with the CON01 bit, channels 2 and 3 are concatenated with the CON23 bit, and so on. Change these bits only when both corresponding channels are disabled.

As shown in Figure 29-20, when channels 2 and 3 are concatenated, channel 2 registers become the high order bytes of the double byte channel. When channels 0 and 1 are concatenated, channel 0 registers become the high order bytes of the double byte channel.

When using the 16-bit concatenated mode, the clock source is determined by the low order 8-bit channel clock select control bits (the odd numbered channel). The resulting PWM is output to the pins of the corresponding low order 8-bit channel, as shown in Figure 29-20. The polarity of the resulting PWM output is controlled by the PPOL*n* bit of the corresponding low order 8-bit channel as well.

After concatenated mode is enabled (PWMCTL[CON*nn*] bits set), enabling/disabling the corresponding 16-bit PWM channel is controlled by the low order PWME*n* bit. In this case, the high order bytes' PWME*n* bits have no effect, and their corresponding PWM output is disabled.

In concatenated mode, writes to the 16-bit counter by using a 16-bit access or writes to the low or high order byte of the counter resets the 16-bit counter. Reads of the 16-bit counter must be made by 16-bit access to maintain data coherency.

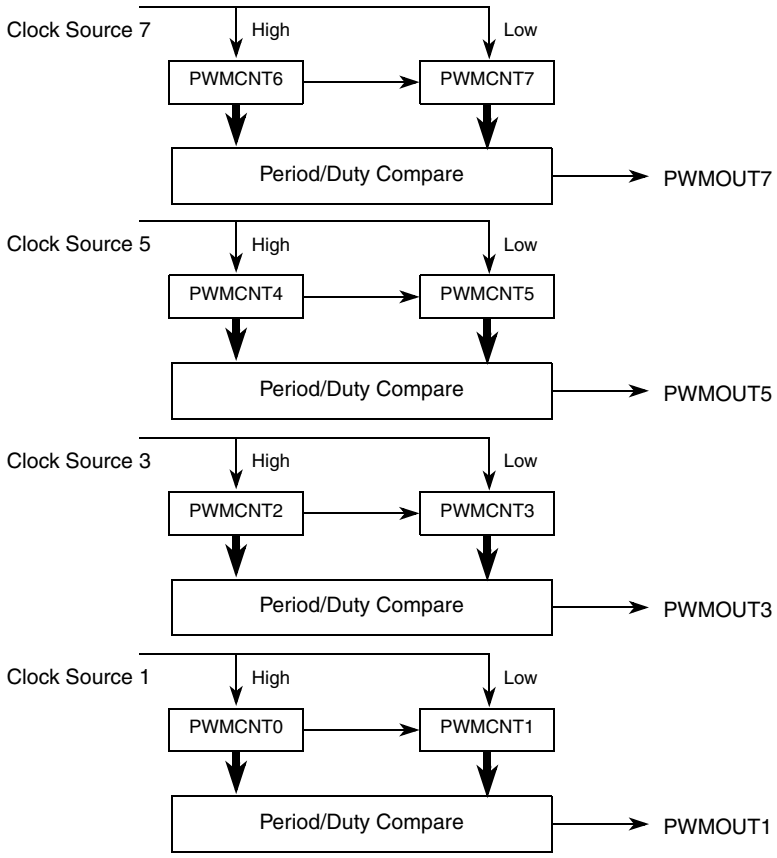


Figure 29-20. PWM 16-Bit Mode

Left- or center-aligned output mode can be used in concatenated mode and is controlled by the low order CAE*n* bit. The high order CAE*n* bit has no effect. The table shown below is used to summarize which channels are used to set the various control bits when in 16-bit mode.

Table 29-15. 16-bit Concatenation Mode Summary

CON*nn*	PWME*n*	PPOL*n*	PCLK*n*	CAE*n*	PWM*n* Output
CON67	PWM7	PPOL7	PCLK7	CAE7	PWMOUT7
CON45	PWM5	PPOL5	PCLK5	CAE5	PWMOUT5
CON23	PWME3	PPOL3	PCLK3	CAE3	PWMOUT3
CON01	PWME1	PPOL1	PCLK1	CAE1	PWMOUT1

29.3.2.8 PWM Boundary Cases

The following table summarizes the boundary conditions for the PWM regardless of the output mode (left- or center-aligned) and 8-bit (normal) or 16-bit (concatenation):

Table 29-16. PWM Boundary Cases

PWMDTY*n*	PWMPER*n*	PPOL*n*	PWM*n* Output
0x00 (indicates no duty)	>0x00	1	Always Low
0x00 (indicates no duty)	>0x00	0	Always High
XX	0x00[1] (indicates no period)	1	Always High
XX	0x00[1] (indicates no period)	0	Always Low
≥ PWMPER*n*	XX	1	Always High
≥ PWMPER*n*	XX	0	Always Low

[1] Counter = 0x00 and does not count.

Chapter 30
FlexCAN

30.1 Introduction

The FlexCAN is a communication controller implementing the controller area network (CAN) protocol, an asynchronous communications protocol used in automotive and industrial control systems. It is a high speed (1 Mbps), short distance, priority-based protocol that can communicate using a variety of mediums (such as fiber optic cable or an unshielded twisted pair of wires). The FlexCAN supports the standard and extended identifier (ID) message formats specified in the CAN protocol specification, revision 2.0, part B.

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. A general working knowledge of the CAN protocol revision 2.0 is assumed in this document. For details, refer to the CAN protocol revision 2.0 specification.

30.1.1 Block Diagram

A block diagram describing the various submodules of the FlexCAN module is shown in Figure 30-1. Each submodule is described in detail in subsequent sections.

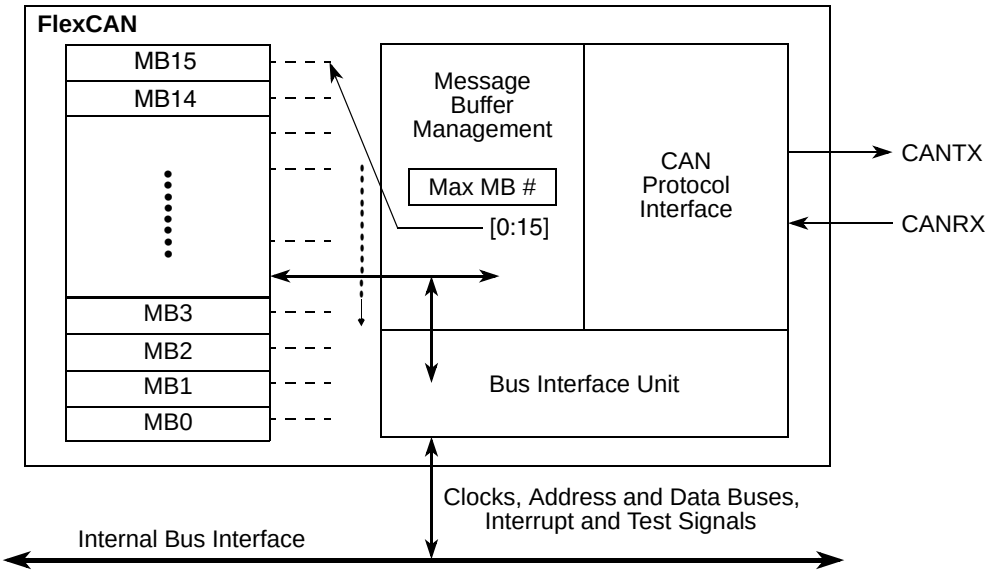


Figure 30-1. FlexCAN Block Diagram

The message buffer architecture is shown in Figure 30-2.

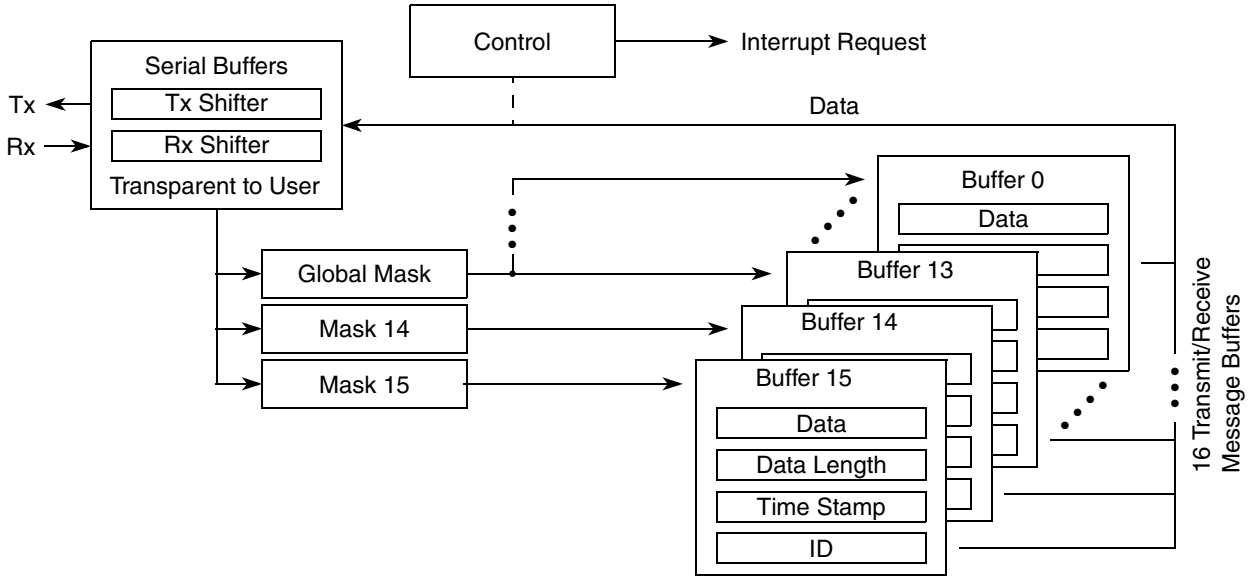


Figure 30-2. FlexCAN Message Buffer Architecture

30.1.1.1 The CAN System

A typical CAN system is shown below in Figure 30-3. Each CAN station is connected physically to the CAN bus through a transceiver. The transceiver provides the transmit drive, waveshaping, and receive/compare functions required for communicating on the CAN bus. It can also provide protection against damage to the FlexCAN caused by a defective CAN bus or defective stations.

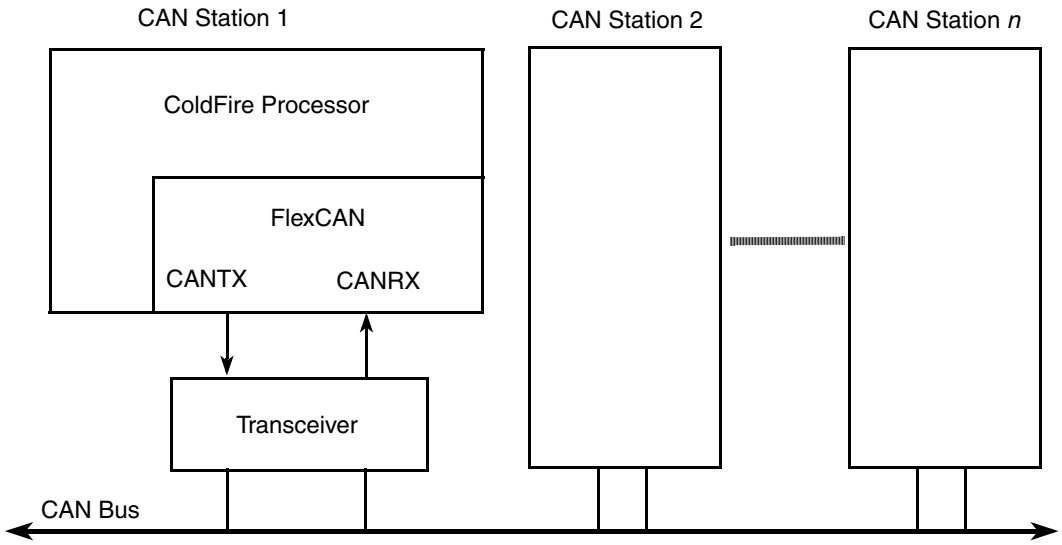


Figure 30-3. Typical CAN System

30.1.2 Features

Following are the main features of the FlexCAN module:

- Full implementation of the CAN protocol specification version 2.0B
  - — Standard data and remote frames (up to 109 bits long)
  - — Extended data and remote frames (up to 127 bits long)
  - — 0–8 bytes data length
  - — Programmable bit rate up to 1 Mbps
  - — Content-related addressing
- Up to 16 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Three programmable mask registers: global (for MBs 0–13), special for MB14, and special for MB15
- Programmable transmission priority 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)
- Open network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

30.1.3 Modes of Operation

30.1.3.1 Normal Mode

In normal mode, the module operates receiving and/or transmitting message frames, errors are managed normally, and all the CAN protocol functions are enabled. User and supervisor modes differ in the access to some restricted control registers.

30.1.3.2 Freeze Mode

Freeze mode is entered by setting:

- CANMCR[FRZ], and
- CANMCR[HALT], or by asserting the $\overline{\text{BKPT}}$ signal.

After entry into freeze mode is requested, the FlexCAN waits until an intermission or idle condition exists on the CAN bus, or until the FlexCAN enters the error passive or bus off state. After one of these conditions exists, the FlexCAN waits for the completion of all internal activity such as arbitration, matching, move-in, and move-out. When this happens, the following events occur:

- The FlexCAN stops transmitting/receiving frames.
- The prescaler is disabled, thus halting all CAN bus communication.
- The FlexCAN ignores its Rx pins and drives its Tx pins as recessive.
- The FlexCAN loses synchronization with the CAN bus and the NOTRDY and FRZACK bits in CANMCR are set.
- The CPU is allowed to read and write the error counter registers (in other modes they are read-only).

After engaging one of the mechanisms to place the FlexCAN in freeze mode, the user must wait for the FRZACK bit to be set before accessing any other registers in the FlexCAN; otherwise, unpredictable operation may occur. In freeze mode, all memory mapped registers are accessible.

To exit freeze mode, the $\overline{\text{BKPT}}$ line must be negated or the HALT bit in CANMCR must be cleared. After freeze mode is exited, the FlexCAN resynchronizes with the CAN bus by waiting for 11 consecutive recessive bits before beginning to participate in CAN bus communication.

30.1.3.3 Module Disabled Mode

This mode disables the FlexCAN module; it is entered by setting CANMCR[MDIS]. If the module is disabled during freeze mode, it shuts down the system clocks, sets the LPMACK bit, and clears the FRZACK bit.

If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in 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 such as arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the system clocks

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 disabled. Exiting from this mode is done by negating the MDIS bit, which resumes the clocks and negate the LPMACK bit.

30.1.3.4 Loop-back Mode

The module enters this mode when the LPB bit in the control register is set. 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. Transmit and receive interrupts are generated.

30.1.3.5 Listen-only Mode

In listen-only mode, transmission is disabled, all error counters are frozen and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station are received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. Because the module does not influence the CAN bus in this mode, the device is capable of functioning like a monitor or for automatic bit-rate detection.

30.2 External Signal Description

Each FlexCAN module has two I/O signals connected to the external MPU pins: CANTX and CANRX. CANTX transmits serial data to the CAN bus transceiver, while CANRX receives serial data from the CAN bus transceiver.

30.3 Memory Map/Register Definition

The FlexCAN module address space is split into 128 bytes starting at the base address, and 256 bytes starting at the base address + 0x80. Out of the lower 128 bytes, only part is occupied by various registers. The upper 256 bytes are fully used for the message buffer structures, as described in Section 30.3.9, “Message Buffer Structure.”

Table 30-1. FlexCAN Memory Map

IPSBAR Offset FlexCAN	Register	Width (bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
Supervisor-only Access Registers							
0x1C_0000	FlexCAN Module Configuration Register (CANMCR)	32	Y	Y	R/W	0xD890_000F	30.3.1/30-6
Supervisor/User Access Registers							
0x1C_0004	FlexCAN Control Register (CANCTRL)	32	Y	N	R/W	0x0000_0000	30.3.2/30-8
0x1C_0008	Free Running Timer (TIMER)	32	Y	Y	R/W	0x0000_0000	30.3.3/30-10
0x1C_0010	Rx Global Mask (RXGMASK)	32	Y	N	R/W	0x1FFF_FFFF	30.3.4/30-11
0x1C_0014	Rx Buffer 14 Mask (RX14MASk)	32	Y	N	R/W	0x1FFF_FFFF	30.3.4/30-11
0x1C_0018	Rx Buffer 15 Mask (RX15MASK)	32	Y	N	R/W	0x1FFF_FFFF	30.3.4/30-11
0x1C_001C	Error Counter Register (ERRCNT)	32	Y	Y	R/W	0x0000_0000	30.3.6/30-13
0x1C_0020	Error and Status Register (ERRSTAT)	32	Y	Y	R/W	0x0000_0000	30.3.6/30-13
0x1C_0028	Interrupt Mask Register (IMASK)	32	Y	Y	R/W	0x0000_0000	30.3.7/30-15
0x1C_0030	Interrupt Flag Register (IFLAG)	32	Y	Y	R/W	0x0000_0000	30.3.8/30-16
0x1C_0080	Message Buffers 0–15 (MB0–15)	2048	N	N	R/W	—	30.3.9/30-16

NOTE

The FlexCAN has no hard-wired protection against invalid bit/field programming within its registers. Specifically, no protection is provided if the programming does not meet CAN protocol requirements.

Programming the FlexCAN control registers is typically done during system initialization, prior to the FlexCAN becoming synchronized with the CAN bus. The configuration registers can be changed after synchronization by halting the FlexCAN module. This is done when the user sets the CANMCR[HALT] bit. The FlexCAN responds by setting the CANMCR[NOTRDY] bit.

30.3.1 FlexCAN Configuration Register (CANMCR)

CANMCR defines global system configurations, such as the module operation mode 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.

IPSBAR Offset: 0x1C_0000 (CANMCR) — Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	0	HALT	NOT RDY	0	SOFT RST	FRZ ACK	SUPV	0	0	LPM ACK	0	0	0	0
W																
Reset	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	0

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

Figure 30-4. FlexCAN Configuration Register (CANMCR)

Table 30-2. CANMCR Field Descriptions

Field	Description
31 MDIS	Module disable. This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the FlexCAN clocks that drive the CAN interface and Message Buffer sub-module. This is the only bit in CANMCR not affected by soft reset. See Section 30.1.3.3, “Module Disabled Mode,” for more information. 0 Enable the FlexCAN module, clocks enabled 1 Disable the FlexCAN module, clocks disabled
30 FRZ	Freeze mode enable. When set, the FlexCAN can enter freeze mode when the $\overline{\text{BKPT}}$ line is asserted or the HALT bit is set. Clearing this bit causes the FlexCAN to exit freeze mode. Refer to Section 30.1.3.2, “Freeze Mode,” for more information. 0 FlexCAN ignores the $\overline{\text{BKPT}}$ signal and the CANMCR[HALT] bit. 1 FlexCAN module enabled to enter debug mode.
29	Reserved, must be cleared.

Table 30-2. CANMCR Field Descriptions (continued)

Field	Description
28 HALT	Halt FlexCAN. Setting this bit puts the FlexCAN module into freeze mode. It has the same effect as assertion of the BKPT signal. This bit is set after reset and should be cleared after initializing the message buffers and control registers. FlexCAN message buffer receive and transmit functions are inactive until this bit is cleared. While in freeze mode, the CPU has write access to the error counter register (ERRCNT) that is otherwise read-only. 0 The FlexCAN operates normally 1 FlexCAN enters freeze mode if FRZ equals 1
27 NOTRDY	FlexCAN not ready. This bit indicates that the FlexCAN is in disable or freeze mode. This bit is read-only and it is cleared after the FlexCAN exits these modes. 0 FlexCAN is in normal mode, listen-only mode, or loop-back mode. 1 FlexCAN is in disable or freeze mode.
26	Reserved, must be cleared.
25 SOFTRST	Soft reset. When set, the FlexCAN resets its internal state machines (sequencer, error counters, error flags, and timer) and the host interface registers (CANMCR [except the MDIS bit], TIMER, ERRCNT, ERRSTAT, IMASK, and IFLAG). The configuration registers that control the interface with the CAN bus are not changed (CANCTRL, RXGMASK, RX14MASK, RX15MASK). Message buffers are also not changed. This allows SOFTRST to be used as a debug while the system is running. Because 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 SOFTRST bit remains set while reset is pending and is automatically cleared when reset completes. The user should poll this bit to know when the soft reset has completed. 0 Soft reset cycle completed 1 Soft reset cycle initiated
24 FRZACK	Freeze acknowledge. Indicates that the FlexCAN module has entered freeze mode. The user should poll this bit after freeze mode has been requested, to know when the module has actually entered freeze mode. When freeze mode is exited, this bit is cleared after the FlexCAN prescaler is enabled. This is a read-only bit. 0 The FlexCAN has exited freeze mode and the prescaler is enabled. 1 The FlexCAN has entered freeze mode, and the prescaler is disabled.
23 SUPV	Supervisor/user data space. Places the FlexCAN registers in supervisor or user data space. 0 Registers with access controlled by the SUPV bit are accessible in user or supervisor privilege mode. 1 Registers with access controlled by the SUPV bit are restricted to supervisor mode.
22–21	Reserved, must be cleared.
20 LPMACK	Low power mode acknowledge. Indicates that FlexCAN is disabled. Disabled mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPMACK bit to know when the FlexCAN has actually entered low power mode. See Section 30.1.3.3, “Module Disabled Mode,” and Chapter 8, “Power Management,” for more information. This bit is read-only. 0 FlexCAN not disabled. 1 FlexCAN is in disabled mode.
19–4	Reserved, must be cleared.
3–0 MAXMB	Maximum number of message buffers. Defines the maximum number of message buffers that take part in the matching and arbitration process. The reset value (0xF) is equivalent to16 message buffer (MB) configuration. This field should be changed only while the module is in freeze mode. **Note:** Maximum MBs in Use = MAXMB + 1

30.3.2 FlexCAN Control Register (CANCTRL)

CANCTRL 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. 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 disabled or in freeze mode. Exceptions are the BOFFMSK, ERRMSK, and BOFFREC bits, which can be accessed at any time.

IPSBAR Offset: 0x1C_0004 (CANCTRL) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BOFF MSK	ERR MSK	CLK_ SRC	LPB	0	0	0	0	SMP	BOFF REC	TSYN	LBUF	LOM	PROPSEG		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-5. FlexCAN Control Register (CANCTRL)

Table 30-3. CANCTRL Field Descriptions

Field	Description
31–24 PRESDIV	Prescaler division factor. Defines the ratio between the clock source frequency (set by CLK_SRC bit) and the serial clock (S clock) frequency. The S clock period defines the time quantum of the CAN protocol. For the reset value, the S clock frequency is equal to the clock source frequency. The maximum value of this register is 0xFF, that gives a minimum S clock frequency equal to the clock source frequency divided by 256. For more information refer to Section 30.3.18, "Protocol Timing." $\text{S clock frequency} = \frac{f_{sys} \text{ or EXTAL}}{\text{PRESDIV} + 1}$ ***Eqn. 30-1***
23–22 RJW	Resynchronization jump width. Defines the maximum number of time quanta (one time quantum is equal to the S clock period) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. Resync jump width = (RJW + 1) time quanta ***Eqn. 30-2***
21–19 PSEG1	Phase buffer segment 1. 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) time quanta ***Eqn. 30-3***
18–16 PSEG2	Phase buffer segment 2. 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) time quanta ***Eqn. 30-4***
15 BOFFMSK	Bus off interrupt mask. 0 Bus off interrupt disabled 1 Bus off interrupt enabled

Table 30-3. CANCTRL Field Descriptions (continued)

Field	Description
14 ERRMSK	Error interrupt mask. 0 Error interrupt disabled 1 Error interrupt enabled
13 CLK_SRC	Clock source. Selects the clock source for the CAN interface to be fed to the prescalar. This bit should only be changed while the module is disabled. 0 Clock source is EXTAL 1 Clock source is the internal bus clock, f_{sys}
12 LPB	Loop back. 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. Transmit and receive interrupts are generated. 0 Loop back disabled 1 Loop back enabled
11–8	Reserved, must be cleared.
7 SMP	Sampling mode. Determines whether the FlexCAN module samples each received bit one time or three times to determine its value. 0 One sample, taken at the end of phase buffer segment 1, is used to determine the value of the received bit. 1 Three samples are used to determine the value of the received bit. The samples are taken at the normal sample point and at the two preceding periods of the S-clock; a majority rule is used.
6 BOFFREC	Bus off recovery mode. Defines how FlexCAN recovers from bus off state. If this bit is cleared, automatic recovering from bus off state occurs according to the *CAN Specification 2.0B*. If the bit is set, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is cleared by the user. If the bit is cleared before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been set. If the bit is cleared after 128 sequences of 11 recessive bits occurred, FlexCAN re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After clearing, the BOFFREC bit can be set again during bus off, but it is only effective the next time the module enters bus off. If BOFFREC was cleared when the module entered bus off, setting it during bus off is not effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0B 1 Automatic recovering from bus off state disabled
5 TSYN	Timer synchronize mode. Enables the mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This provides the means to synchronize multiple FlexCAN stations with a special SYNC message (global network time). 0 Timer synchronization disabled. 1 Timer synchronization enabled. **Note:** There can be a bit clock skew of four to five counts between different FlexCAN modules that are using this on the same network.
4 LBUF	Lowest buffer transmitted first. Defines the ordering mechanism for message buffer transmission. 0 Message buffer with lowest ID is transmitted first 1 Lowest numbered buffer is transmitted first

Table 30-3. CANCTRL Field Descriptions (continued)

Field	Description
3 LOM	Listen-only mode. 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. Only messages acknowledged by another CAN station is received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 FlexCAN module is in normal active operation; listen-only mode is deactivated 1 FlexCAN module is in listen-only mode operation
2–0 PROPSEG	Propagation segment. 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 ***Eqn. 30-5*** **Note:** A time-quantum equals 1 S clock period.

30.3.3 FlexCAN Free Running Timer Register (TIMER)

This register represents a 16-bit free running counter that can be read and written to 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 received or transmitted bit. 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 (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP 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, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

IPSBAR 0x1C_0008 (TIMER) Access: User read/write
Offset:

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

Figure 30-6. FlexCAN Timer Register (TIMER)

Table 30-4. TIMER Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 TIMER	Free running timer. Captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

30.3.4 Rx Mask Registers (RXGMASK, RX14MASK, RX15MASK)

These registers are used as acceptance masks for received frame IDs. Three masks are defined: a global mask (RXGMASK) used for Rx buffers 0–13 and two separate masks for buffers 14 (RX14MASK) and 15 (RX15MASK). The meaning of each mask bit is the following:

MI*n* bit = 0: The corresponding incoming ID bit is don't care.

MI*n* bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

These masks are used for standard and extended ID formats. The value of the mask registers should not be changed while in normal operation (only while in freeze mode), as locked frames that matched a message buffer (MB) through a mask may be transferred into the MB (upon release) but may no longer match.

Table 30-5. Mask Examples for Normal/Extended Messages

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2-ID	1 1 1 1 1 1 1 1 0 0 0	0		
MB3-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4-ID	0 0 0 0 0 0 0 1 1 1 1	0		
MB5-ID	0 0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx_Global_Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
Rx_Msg in[1]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB3[1]
Rx_Msg in[2]	1 1 1 1 1 1 1 1 0 0 1	0		MB2[2]
Rx_Msg in[3]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	[3]
Rx_Msg in[4]	0 1 1 1 1 1 1 1 0 0 0	0		[4]
Rx_Msg in[5]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[5]
RX14MASK	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
Rx_Msg in[6]	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	[6]
Rx_Msg in[7]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[7]

[1] Match for Extended Format (MB3).

[2] Match for Normal Format. (MB2).

[3] Mismatch for MB3 because of ID0.

4 Mismatch for MB2 because of ID28.
5 Mismatch for MB3 because of ID28, Match for MB14 (Uses RX14MASK).
6 Mismatch for MB14 because of ID27 (Uses RX14MASK).
7 Match for MB14 (Uses RX14MASK).

IPSBAR 0x1C_0010 (RXGMASK) Access: User read/write
Offsets: 0x1C_0014 (RX14MASK)
0x1C_0018 (RX15MASK)

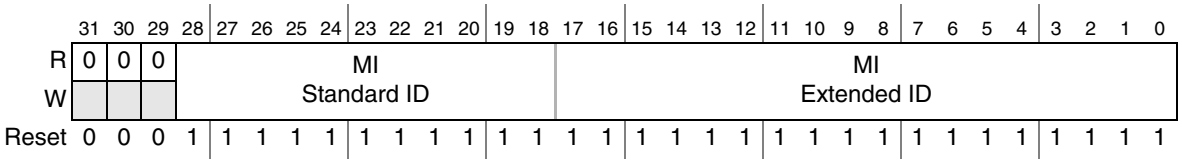


Figure 30-7. FlexCAN Rx Mask Registers (RXGMASK, RX14MASK, RX15MASK)

Table 30-6. RX*xx*MASK Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28–18 MI28–18	Standard ID mask bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–0	Extended ID mask bits. These bits are used to mask comparison only in Extended Format.

30.3.5 FlexCAN Error Counter Register (ERRCNT)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: transmit error counter (TXECTR) and receive error counter (RXECTR). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only, except in freeze mode, where they can be written by the CPU.

Writing to the ERRCNT register while in freeze mode is an indirect operation. The data is first written to an auxiliary register, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

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 TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the error and status register (ERRSTAT) is updated to reflect error-passive state.
- If the FlexCAN state is error-passive, and TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the ERRSTAT[FLTCONF] field is updated to reflect error-active state.
- If the value of TXECTR increases to be greater than 255, the ERRSTAT[FLTCONF] field is updated to reflect bus off state, and an interrupt may be issued. The value of TXECTR is then reset to zero.

- If FlexCAN is in bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the ERRSTAT[FLTCONF] field is updated to be error-active, and both error counters are reset to zero. At any instance of a dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ERRSTAT[ACKERR] bit). After the transition to error-passive state, the TXECTR does not increment anymore by acknowledge errors. Therefore, the device never goes to the bus off state.
- If the RXECTR 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.

IPSBAR Offset: 0x1C_001C (ERRCNT) Access: User read/write

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

Figure 30-8. FlexCAN Error Counter Register (ERRCNT)

Table 30-7. ERRCNT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 RXECTR	Receive error counter. Indicates current number of receive errors.
7–0 TXECTR	Transmit error counter. Indicates current number of transmit errors.

30.3.6 FlexCAN Error and Status Register (ERRSTAT)

ERRSTAT reflects various error conditions, some general status of the device, and is the source of three interrupts to the CPU. The reported error conditions (bits 15:10) are those occurred since the last time the CPU read this register. The read action clears bits 15-10. Bits 9–3 are status bits.

Most bits in this register are read only, except for BOFFINT and ERRINT, which are interrupt flags that can be cleared by writing 1 to them. Writing 0 has no effect. Refer to Section 30.4.1, “Interrupts.”

IPSBAR 0x1C_0020 (ERRSTAT) Access: User read/write
Offset:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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	0
W														w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-9. FlexCAN Error and Status Register (ERRSTAT)

Table 30-8. ERRSTAT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15 BIT1ERR	Bit1 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as recessive was received as dominant **Note:** The transmit bit error field is not modified during the arbitration field or the ACK slot bit time of a message, or by a transmitter that detects dominant bits while sending a passive error frame.
14 BIT0ERR	Bit0 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as dominant was received as recessive
13 ACKERR	Acknowledge error. Indicates whether an acknowledgment has been correctly received for a transmitted message. 0 No ACK error was detected since the last read of this register. 1 An ACK error was detected since the last read of this register.
12 CRCERR	Cyclic redundancy check error. Indicates whether or not a CRC error has been detected by the receiver. 0 No CRC error was detected since the last read of this register. 1 A CRC error was detected since the last read of this register.
11 FRMERR	Message form error. Indicates that a form error has been detected by the receiver node, i.e. a fixed-form bit field contains at least one illegal bit. 0 No form error was detected since the last read of this register. 1 A form error was detected since the last read of this register.
10 STFERR	Bit stuff error. 0 No bit stuffing error was detected since the last read of this register. 1 A bit stuffing error was detected since the last read of this register.
9 TXWRN	Transmit error status flag. Reflects the status of the FlexCAN transmit error counter. 0 Transmit error counter < 96 1 TXErrCounter ≥ 96
8 RXWRN	Receiver error status flag. Reflects the status of the FlexCAN receive error counter. 0 Receive error counter < 96 1 RxErrCounter ≥ 96

Table 30-8. ERRSTAT Field Descriptions (continued)

Field	Description
7 IDLE	Idle status. Indicates when there is activity on the CAN bus. 0 The CAN bus is not idle. 1 The CAN bus is idle.
6 TXRX	Transmit/receive status. Indicates when the FlexCAN module is transmitting or receiving a message. TXRX has no meaning when IDLE equals 1. 0 The FlexCAN is receiving a message if IDLE equals 0. 1 The FlexCAN is transmitting a message if IDLE equals 0.
5–4 FLTCONF	Fault confinement state. Indicates the confinement state of the FlexCAN module, as shown below. If the CANCTRL[LOM] bit is set, FLTCONF indicates error-passive. Because the CANCTRL register is not affected by soft reset, the FLTCONF field is not affected by soft reset if the LOM bit is set. 00 Error active 01 Error passive 1x Bus off
3	Reserved, must be cleared.
2 BOFFINT	Bus off interrupt. Used to request an interrupt when the FlexCAN enters the bus off state. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No bus off interrupt requested. 1 This bit is set when the FlexCAN state changes to bus off. If the CANCTRL[BOFFMSK] bit is set an interrupt request is generated. This interrupt is not requested after reset.
1 ERRINT	Error interrupt. Indicates that at least one of the ERRSTAT[15:10] bits is set. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No error interrupt request. 1 At least one of the error bits is set. If the CANCTRL[ERRMSK] bit is set, an interrupt request is generated.
0	Reserved, must be cleared.

30.3.7 Interrupt Mask Register (IMASK)

IMASK contains one interrupt mask bit per buffer. It enables the CPU to determine which buffer generates an interrupt after a successful transmission/reception (when the corresponding IFLAG bit is set).

IPSBAR Offset: 0x1C_0028 (IMASK) Access: User read/write

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

Figure 30-10. FlexCAN Interrupt Mask Register (IMASK)

Table 30-9. IMASK Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BUF*n*M	Buffer interrupt mask. Enables the respective FlexCAN message buffer (MB0 to MB15) interrupt. These bits allow the CPU to designate which buffers generate interrupts after successful transmission/reception. 0 The interrupt for the corresponding buffer is disabled. 1 The interrupt for the corresponding buffer is enabled. **Note:** Setting or clearing an IMASK bit can assert or negate an interrupt request, if the corresponding IFLAG bit it is set.

30.3.8 Interrupt Flag Register (IFLAG)

IFLAG contains one interrupt flag bit per buffer. Each successful transmission/reception sets the corresponding IFLAG bit and, if the corresponding IMASK bit is set, generates an interrupt.

The interrupt flag is cleared by writing a 1, while writing 0 has no effect.

IPSBAR Offset: 0x1C_0030 (IFLAG) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BUF*n*I															
W																	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-11. FlexCAN Interrupt Flags Register (IFLAG)

Table 30-10. IFLAG Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BUF*n*I	Buffer interrupt flag. Indicates a successful transmission/reception for the corresponding message buffer. If the corresponding IMASK bit is set, an interrupt request is generated. The user must write a 1 to clear an interrupt flag; writing 0 has no effect. 0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.

30.3.9 Message Buffer Structure

The message buffer memory map starts at an offset of 0x80 from the FlexCAN's base address (0x1C_0000). The 256-byte message buffer space is fully used by the16 message buffer structures.

Each message buffer consists of a control and status field that configures the message buffer, an identifier field for frame identification, and up to 8 bytes of data.

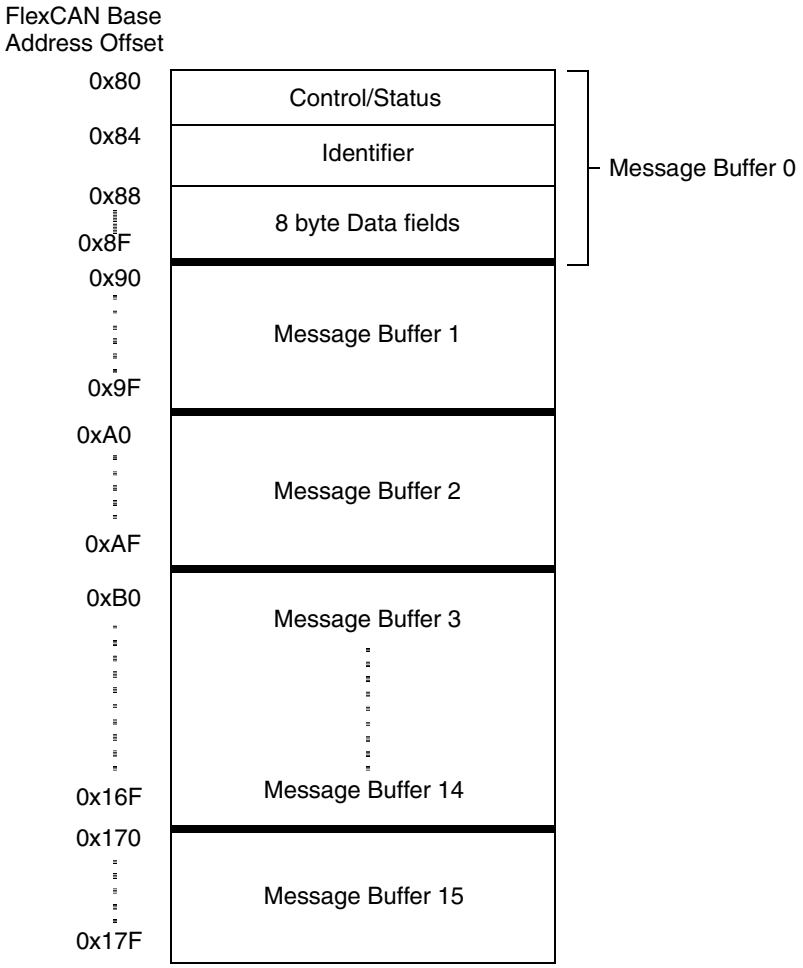


Figure 30-12. FlexCAN Message Buffer Memory Map

The message buffer structure used by the FlexCAN module is shown in Figure 30-13. Standard and extended frames used in the *CAN Specification Version 2.0, Part B* are represented. A standard frame is represented by the 11-bit standard identifier, and an extended frame is represented by the combined 29-bits of the standard identifier (11 bits) and the extended identifier (18 bits).

	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
0x0		CODE		SRR IDE RTR	LENGTH	TIME STAMP		
0x4		Standard ID[28:18]		Extended ID[17:0]				
0x8	Data Byte 0		Data Byte 1		Data Byte 2		Data Byte 3	
0xC	Data Byte 4		Data Byte 5		Data Byte 6		Data Byte 7	

Figure 30-13. Message Buffer Structure for Extended and Standard Frames

Table 30-11. Message Buffer Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 CODE	Message buffer code. 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 30-12 and Table 30-13. See Section 30.3.10, “Functional Overview,” for additional information.
23	Reserved, must be cleared.
22 SRR	Substitute remote request. Fixed recessive bit, used only in extended format. It must be set by the user for transmission (Tx Buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as 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
21 IDE	ID extended bit. Identifies whether the frame format is standard or extended. 0 Standard frame format 1 Extended frame format
20 RTR	Remote transmission request. 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
19–16 LENGTH	Length of data in bytes. Indicates the length (in bytes) of the Rx or Tx data; data is located in offset 0x8 through 0xF of the MB space (see Figure 30-13). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. DLC is defined by the *CAN Specification* and refers to the data length of the actual frame before it is copied into the message buffer. In transmission, this field is written by the CPU and is used as the DLC field value of the frame to be transmitted. When RTR is set, the frame to be transmitted is a remote frame and is transmitted without the DATA field, regardless of the LENGTH field.
15–-0 TIME STAMP	Free-running counter time stamp. Stores the value of the free-running timer which is captured when the beginning of the identifier (ID) field appears on the CAN bus.
31–29	Reserved, must be cleared.
28–0 ID	Standard frame identifier: In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in receive and transmit cases. The 18 least significant bits are ignored.
	Extended frame identifier: In extended frame format, all bits (the 11 bits of the standard frame identifier and the 18 bits of the extended frame identifier) are used for frame identification in receive and transmit cases.
31–24, 23–16, 15–8, 7–0 DATA	Data field. Up to eight 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 provides the data to be transmitted within the frame.

Table 30-12. 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 control & status (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 remains FULL.
		0110	If the MB is FULL and a new frame should be written into this MB before the CPU had time to read it, the MB is overwritten, and the code is automatically updated to OVERRUN.
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB, the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN.
0XY1[1]	BUSY: Flexcan is updating the contents of the MB with a new receive frame. The CPU should not try to access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

[1] For transmit message buffers (see Table 30-13), the BUSY bit should be ignored upon read.

Table 30-13. Message Buffer Code for Tx Buffers

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
X	1000	—	INACTIVE: Message buffer not ready for transmit and participates in the arbitration process.
0	1100	1000	Data frame to be transmitted once, unconditionally. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Remote frame to be transmitted unconditionally once, and message buffer becomes an Rx message buffer with the same ID for data frames.

Table 30-13. Message Buffer Code for Tx Buffers (continued)

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
0	1010	1010	Transmit a data frame when a remote request frame with the same ID is received. This message buffer participates simultaneously in 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 message buffer 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 automatically written to the message buffer as a result of match to a remote request frame. The data frame is transmitted unconditionally once, and then the code automatically returns to 1010. The CPU can also write this code with the same effect.

30.3.10 Functional Overview

The FlexCAN module is flexible in that each one of its 16 message buffers (MBs) can be assigned as a transmit buffer or a receive buffer. Each MB, which is up to 8 bytes long, is also assigned an interrupt flag bit that indicates successful completion of transmission or reception.

An arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID or the MB ordering. 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. Data coherency mechanisms are implemented to guarantee data integrity during MB manipulation by the CPU.

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 30-12). Similarly, a Tx MB with a 1000 code is inactive (refer to Table 30-13). An MB not programmed with 0000 or 1000 is temporarily deactivated (does not participate in the current arbitration/matching run) when the CPU writes to the C/S field of that MB.

30.3.11 Transmit Process

The CPU prepares or changes an MB for transmission by writing the following:

1. Control/status word to hold Tx MB inactive (CODE = 1000)
2. ID word
3. Data bytes
4. Control/status word (active CODE, LENGTH)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or ID matching processes, giving time for the CPU to program the rest of the MB (see Section 30.3.15.1, “Message Buffer Deactivation”). After the MB is activated in the fourth step, it participates in 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 (TIMER) is written into the message buffer’s time stamp field, the code field in the control and status word is updated, a status flag is set in the IFLAG register, and an interrupt is generated if allowed by the corresponding IMASK register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see Table 30-13).

30.3.12 Arbitration Process

The arbitration process is an algorithm executed by the message buffer management (MBM) that scans the entire MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers are scanned to find the lowest ID or the lowest MB number, depending on the CANCTRL[LBUF] bit.

NOTE

If CANCTRL[LBUF] is cleared, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

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

After 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.

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to 8 data bytes, even if the data length code (DLC) value is bigger.

30.3.13 Receive Process

The CPU prepares or changes an MB for frame reception by writing the following:

1. Control/status word to hold Rx MB inactive (CODE = 0000)

2. ID word
3. Control/status word to mark the Rx MB as active and empty (CODE = 0100)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was a pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or matching process, giving time for the CPU to program the rest of the MB. After the MB is activated in the third step, it is able to receive CAN frames that match the programmed ID. At the end of a successful reception:

- The value of the free running timer (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 30-12), and
- A status flag is set in the IFLAG register and an interrupt is generated if allowed by the corresponding IMASK bit.

The CPU should read a receive frame from its MB by reading the following:

1. Control/status word (mandatory—activates internal lock for this buffer)
2. ID (optional—needed only if a mask was used)
3. Data field words
4. Free-running timer (Releases internal lock —optional)

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. 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.

The CPU should synchronize to frame reception by an IFLAG bit for the specific MB (see Section 30.3.8, “Interrupt Flag Register (IFLAG)”), and not by the control/status word CODE field for that MB. Polling the CODE field does not work because after a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the CODE field does not return to EMPTY. It remains FULL, as explained in Table 30-12. If the CPU tries to workaround 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 poll by directly reading the C/S word of the MBs. Instead, read the IFLAG register.

The received identifier 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.

30.3.13.1 Self-Received Frames

Self-received frames are frames that are sent by the FlexCAN and received by itself. The FlexCAN sends a frame externally through the physical layer onto the CAN bus. If the ID of the frame matches the ID of the FlexCAN MB, the frame is received by the FlexCAN. Such a frame is a self-received frame. FlexCAN

does not receive frames transmitted by itself if another device on the CAN bus has an ID that matches the FlexCAN Rx MB ID.

30.3.14 Matching Process

The matching process is an algorithm that scans the entire MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. Only MBs programmed to receive participate in the matching process for received frames.

While the ID, DLC and data fields are retrieved from the CAN bus, they are stored temporarily in the serial message buffer. The matching process takes place during the CRC field. If a matching ID is found in one of the MBs, the contents of the SMB are transferred to the matched MB during the sixth 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.

An MB with a matching ID is free to receive a new frame if the MB is not locked (see Section 30.3.15.2, “Locking and Releasing Message Buffers”). The CODE field is EMPTY, FULL, or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB).

Matching to a range of IDs is possible by using ID acceptance masks. FlexCAN supports a masking scheme with three mask registers (RXGMASK, RX14MASK, and RX15MASK). 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.

30.3.15 Message Buffer Managing

To maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 30.3.11, “Transmit Process” and Section 30.3.13, “Receive Process.” Any form of CPU accessing a MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

30.3.15.1 Message Buffer Deactivation

If the CPU wants to change the function of an active MB, the recommended procedure is to put the module into freeze mode and then change the CODE field of that MB. This is a safe procedure because the FlexCAN waits for pending CAN bus and MB moving activities to finish before entering freeze mode. Nevertheless, a mechanism is 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 C/S word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. This mechanism is called MB deactivation. It 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, that MB is deactivated.

Even with the coherence mechanism described above, writing to the C/S 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 continues looking for another matching MB within the ones it has not scanned yet. If it can not find one, the message is 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 is lost even if the second matching MB was free to receive.
- If a Tx MB containing the lowest ID is deactivated after the FlexCAN has scanned it, the FlexCAN looks for another winner within the MBs that it has not yet scanned. Therefore, it may transmit an MB that may not have the lowest ID at the time because a lower ID might be present 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.

30.3.15.2 Locking and Releasing Message Buffers

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 which have a code different than INACTIVE (0000) or EMPTY1 (0100). Also, Tx MBs can not be locked.

Suppose, for example, that FlexCAN has already received and stored a message into one of the MBs. Suppose now that the CPU decides to read that MB at the same time another message with the same ID is arriving. When the CPU reads the control and status word, the MB is locked. The new message arrives and the matching algorithm finds out that the matching MB is not free to receive It remains in the SMB waiting for the MB to be unlocked, and only then, is it 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 is no indication of lost messages 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 set. 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 cleared.

If the BUSY bit is set or if the MB is empty, then reading the control and status word does not lock the MB.

NOTE

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 is not transferred to the MB anymore.

30.3.16 CAN Protocol Related Frames

30.3.16.1 Remote Frames

The remote frame is a message frame transmitted to request a data frame. The FlexCAN can be configured to transmit a data frame automatically in response to a remote frame, or to transmit a remote frame and then wait for the responding data frame to be received.

When transmitting a remote frame, the user initializes a message buffer as a transmit message buffer with the RTR bit set. After this remote frame is transmitted successfully, the transmit message buffer automatically becomes a receive message buffer, with the same ID as the remote frame that was transmitted.

When a remote frame is received by the FlexCAN, the remote frame ID is compared to the IDs of all transmit message buffers programmed with a CODE of 1010. If there is an exact matching ID, the data frame in that message buffer is transmitted. If the RTR bit in the matching transmit message buffer is set, the FlexCAN transmits a remote frame as a response.

A received remote frame is not stored in a receive message buffer. It is only used to trigger the automatic transmission of a frame in response. The mask registers are not used in remote frame ID matching. All ID bits (except RTR) of the incoming received frame must match for the remote frame to trigger a response transmission. The matching message buffer immediately enters the internal arbitration process, but is considered as a normal Tx MB, with no higher priority. The data length of this frame is independent of the data length code (DLC) field in the remote frame that initiated its transmission.

30.3.16.2 Overload Frames

Overload frame transmissions are not initiated by the FlexCAN unless certain conditions are detected on the CAN bus. These conditions include detection of a dominant bit in the following:

- First or second bit of intermission
- Seventh (last) bit of the end-of-frame (EOF) field in receive frames
- Eighth (last) bit of the error frame delimiter or overload frame delimiter

30.3.17 Time Stamp

The value of TIMER is sampled at the beginning of the identifier field on the CAN bus. For a message being received, the time stamp is stored in the TIMESTAMP entry of the receive message buffer at the time the message is written into that buffer. For a message being transmitted, the TIMESTAMP entry is written into the transmit message buffer after the transmission has completed successfully.

The free-running timer can optionally be reset upon the reception of a frame into message buffer 0. This allows network time synchronization to be performed. See the CANCTRL[TSYN] bit.

30.3.18 Protocol Timing

The FlexCAN module CANCTRL register configures the bit timing parameters required by the CAN protocol. The CLK_SRC, PRESDIV, RJW, PSEG1, PSEG2, and the PROPSEG fields allow the user to configure the bit timing parameters.

The CANCTRL[CLK_SRC] bit defines whether the module uses the internal bus clock or the output of the crystal oscillator via the EXTAL pin. The crystal oscillator clock should be selected when a tight tolerance (up to 0.1%) is required for the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks. The value of this bit should not be changed, unless the module is in disable mode (CANMCR[MDIS] bit is set)

The PRESDIV field controls a prescaler that generates the serial clock (S-clock), whose period defines the time quantum used to compose the CAN waveform. A time quantum is the atomic unit of time managed by the CAN engine.

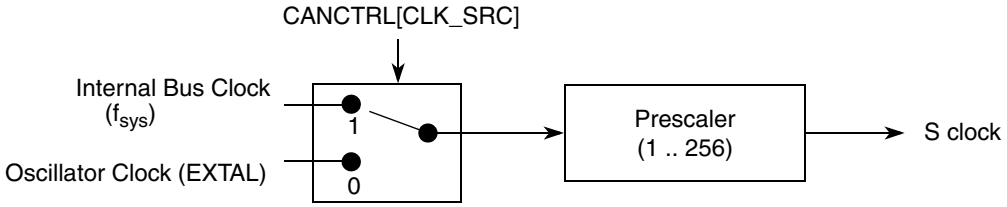


Figure 30-14. CAN Engine Clocking Scheme

$$f_{Tq} = \frac{f_{sys} \text{ or EXTAL}}{(\text{PRESDIV} + 1)} \qquad \textit{Eqn. 30-6}$$

A bit time is subdivided into three segments[1] (see Figure 30-15 and Table 30-14):

- SYNC_SEG: Has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: 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 CANCTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: Represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CANCTRL register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})} \qquad \textit{Eqn. 30-7}$$

1. 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.

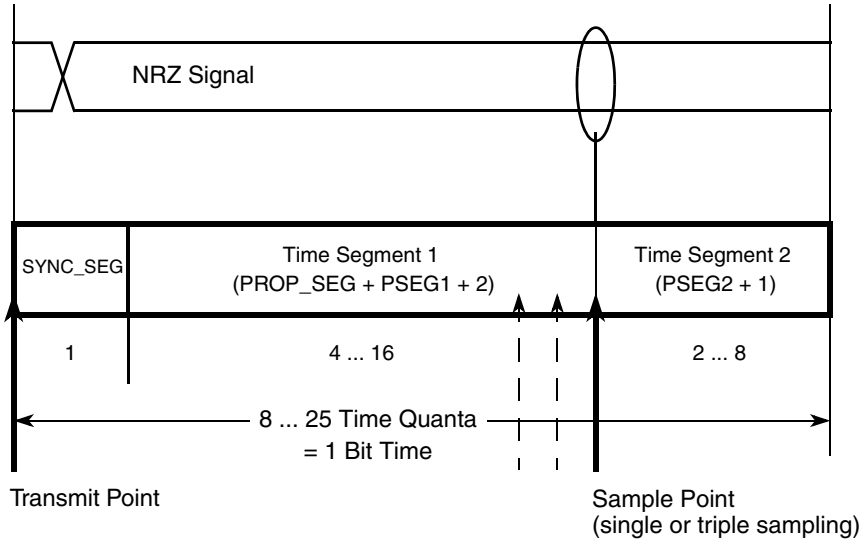


Figure 30-15. Segments within the Bit Time

Table 30-14. 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 30-15 gives an overview of the CAN compliant segment settings and the related parameter values.

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

Table 30-15. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4

Table 30-15. CAN Standard Compliant Bit Time Segment Settings (continued)

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

30.4 Initialization/Application Information

Initialization of the FlexCAN includes the initial configuration of the message buffers and configuration of the CAN communication parameters following a reset, as well as any reconfiguration that may be required during operation. The FlexCAN module may be reset in three ways:

- Device level hard reset—resets all memory mapped registers asynchronously
- Device level soft reset—resets some of the memory mapped registers synchronously (refer to Table 30-1 to see which registers are affected by soft reset)
- CANMCR[SOFT_RST] bit—has the same effect as the device 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 CANMCR[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, CANCTRL[CLK_SRC], should be selected while the module is in disable mode. After the clock source is selected and the module is enabled (CANMCR[MDIS] bit cleared), the FlexCAN automatically enters freeze mode. In freeze mode, the FlexCAN is un-synchronized to the CAN bus, the CANMCR register's HALT and FRZ bits are set, the internal state machines are disabled, and the CANMCR register's FRZ_ACK and NOT_RDY bits are set. The CANTX pin is in recessive state and the FlexCAN does not initiate any transmission or reception of CAN frames. The message buffers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, the FlexCAN must be in freeze mode (see Section 30.1.3.2, "Freeze Mode"). The following is a generic initialization sequence applicable to the FlexCAN module:

1. Initialize all operation modes in the CANCTRL register.
   a) Initialize the bit timing parameters PROPSEG, PSEGS1, PSEG2, and RJW.
   b) Select the S-clock rate by programming the PRESDIV field.
   c) Select the internal arbitration mode via the LBUF bit.
2. Initialize message buffers.
   a) The control/status word of all message buffers must be written as an active or inactive message buffer
   b) All other entries in each message buffer should be initialized as required
3. Initialize RXGMASK, RX14MASK, and RX15MASK registers for acceptance mask as needed.

4. Initialize FlexCAN interrupt handler.
   a) Initialize the interrupt controller registers for any needed interrupts. See Chapter 15, “Interrupt Controller Module,” for more information.
   b) Set the required mask bits in the IMASK register (for all message buffer interrupts) and the CANCTRL (for bus off and error interrupts).
5. Clear the CANMCR[HALT] bit. At this point, the FlexCAN attempts to synchronize with the CAN bus.

30.4.1 Interrupts

There are 18 interrupt sources for the FlexCAN module. An interrupt for each of the 16 MBs. The other interrupt sources (bus off and error) act in the same manner, and are located in the ERRSTAT register. The bus off and error interrupt mask bits are located in the CANCTRL register.

Chapter 31
Debug Module

31.1 Introduction

This chapter describes the revision B+ enhanced hardware debug module.

31.1.1 Block Diagram

The debug module is shown in Figure 31-1.

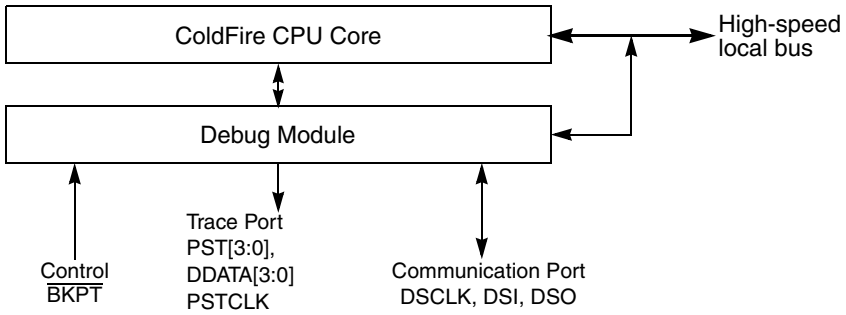


Figure 31-1. Processor/Debug Module Interface

31.1.2 Overview

Debug support is divided into three areas:

- Real-time trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The ColdFire solution implements an 8-bit parallel output bus that reports processor execution status and data to an external emulator system. See Section 31.4.4, “Real-Time Trace Support”.
- Background debug mode (BDM)—Provides low-level debugging in the ColdFire processor complex. In BDM, the processor complex is halted and a variety of commands can be sent to the processor to access memory, registers, and peripherals. The external emulator uses a three-pin, serial, full-duplex channel. See Section 31.4.1, “Background Debug Mode (BDM),” and Section 31.3, “Memory Map/Register Definition”.
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. Debug interrupts let real-time systems execute a unique service routine that can quickly save the contents of key registers and variables and return the system to normal operation. External development systems can access saved data, because the hardware supports concurrent operation of the processor and BDM-initiated commands. In addition, the option allows interrupts to occur. See Section 31.4.2, “Real-Time Debug Support”.

The first version 2 ColdFire core devices implemented the original debug architecture, now called revision A. Based on feedback from customers and third-party developers, enhancements have been added to succeeding generations of ColdFire cores. For revision A, CSR[HRL] is 0. See Section 31.3.2, “Configuration/Status Register (CSR)”.

Revision B (and B+) of the debug architecture offers more flexibility for configuring the hardware breakpoint trigger registers and removing the restrictions involving concurrent BDM processing while hardware breakpoint registers are active. Revision B+ adds three additional PC breakpoint registers. For revision B, CSR[HRL] is 1, and for revision B+, CSR[HRL] is 0x9.

The following table summarizes the various debug revisions.

Table 31-1. Debug Revision Summary

Revision	CSR[HRL]		Enhancements
A	0000	—	Initial debug revision
B	0001	—	BDM command execution does not affect hardware breakpoint logic Added BDM address attribute register (BAAR) $\overline{\text{BKPT}}$ configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor's current PC
B+	1001	—	3 additional PC breakpoint registers PBR1–3

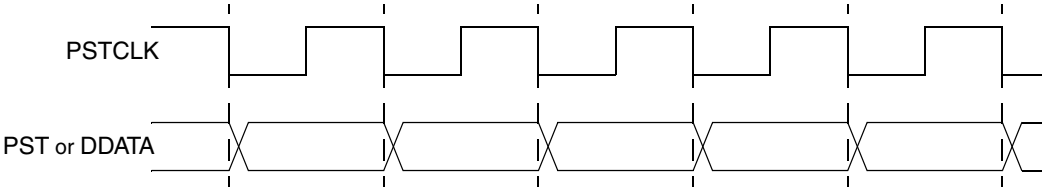
31.2 Signal Descriptions

Table 31-2 describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor core's clock signal. The standard 26-pin debug connector is shown in Section 31.4.6, "Freescale-Recommended BDM Pinout".

Table 31-2. Debug Module Signals

Signal	Description
Development Serial Clock (DSCLK)	Internally synchronized input. (The logic level on DSCLK is validated if it has the same value on two consecutive rising bus clock edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is 1/5 the processor status clock (PSTCLK). At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.
Development Serial Input (DSI)	Internally synchronized input that provides data input for the serial communication port to the debug module after the DSCLK has been seen as high (logic 1).
Development Serial Output (DSO)	Provides serial output communication for debug module responses. DSO is registered internally. The output is delayed from the validation of DSCLK high.
Breakpoint ($\overline{\text{BKPT}}$)	Input requests a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status signals (PST[3:0]) as the value 0xF. If CSR[BKD] is set (disabling normal $\overline{\text{BKPT}}$ functionality), asserting $\overline{\text{BKPT}}$ generates a debug interrupt exception in the processor.

Table 31-2. Debug Module Signals (continued)

Signal	Description
Processor Status Clock (PSTCLK)	Delayed version of the processor clock. Its rising edge appears in the center of valid PST and DDATA output. PSTCLK indicates when the development system should sample PST and DDATA values. The following figure shows PSTCLK timing with respect to PSTD and DATA.  If real-time trace is not used, setting CSR[PCD] keeps PSTCLK, PST and DDATA outputs from toggling without disabling triggers. Non-quiescent operation can be reenabled by clearing CSR[PCD], although the external development systems must resynchronize with the PST and DDATA outputs. PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing. Table 31-24 describes PST values.
Debug Data (DDATA[3:0])	These output signals display the register breakpoint status as a default, or optionally, captured address and operand values. The capturing of data values is controlled by the setting of the CSR. Additionally, execution of the WDDATA instruction by the processor captures operands that are displayed on DDATA. These signals are updated each processor cycle. These signals are not implemented on packages containing fewer than 100 pins.
Processor Status (PST[3:0])	These output signals report the processor status. Table 31-24 shows the encoding of these signals. These outputs indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer. The PST value is updated each processor cycle. These signals are not implemented on packages containing fewer than 100 pins.
All Processor Status Outputs (ALLPST)	ALLPST is a logical AND of the four PST signals and is provided on all packages. PST[3:0] and DDATA[3:0] are not available on the low cost (less than 100 pin) packages. When asserted, reflects that the core is halted.

31.3 Memory Map/Register Definition

In addition to the existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contain a number of registers to support the required functionality. These registers are also accessible from the processor's supervisor programming model by executing the WDEBUG instruction (write only). Therefore, the breakpoint hardware in debug module can be read or written by the external development system using the debug serial interface or written by the operating system running on the processor core. Software guarantees that accesses to these resources are serialized and logically consistent. Hardware provides a locking mechanism in CSR to allow external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued if the ColdFire processor is using the WDEBUG instruction to access debug module registers, or the resulting behavior is undefined. The DSCLK must be quiescent during operation of the WDEBUG command.

These registers, shown in Table 31-3, are treated as 32-bit quantities, regardless of the number of implemented bits. These registers are also accessed through the BDM port by the commands, WDMREG and RDMREG, described in Section 31.4.1.5, "BDM Command Set". These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 31-3.

Table 31-3. Debug Module Memory Map

DRc[4–0]	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W See Note	0x0090_0000	31.3.2/31-5
0x05	BDM address attribute register (BAAR)	32[1]	W	0x05	31.3.3/31-8
0x06	Address attribute trigger register (AATR)	32[1]	W	0x0005	31.3.4/31-9
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	31.3.5/31-10
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined	31.3.6/31-13
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined	31.3.6/31-13
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined	31.3.7/31-15
0x0D	Address breakpoint low register (ABLR)	32	W	Undefined	31.3.7/31-15
0x0E	Data breakpoint register (DBR)	32	W	Undefined	31.3.8/31-16
0x0F	Data breakpoint mask register (DBMR)	32	W	Undefined	31.3.8/31-16
0x18	PC breakpoint register 1 (PBR1)	32	W	See Section	31.3.6/31-13
0x1A	PC breakpoint register 2 (PBR2)	32	W	See Section	31.3.6/31-13
0x1B	PC breakpoint register 3 (PBR3)	32	W	See Section	31.3.6/31-13

[1] Each debug register is accessed as a 32-bit register; reserved fields are not used (don't care).

NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. These control registers are write-only from the programming model and they can be written through the BDM port using the WDMREG command. In addition, the configuration/status register (CSR) can be read through the BDM port using the RDMREG command.

The ColdFire debug architecture supports a number of hardware breakpoint registers, that can be configured into single- or double-level triggers based on the PC or operand address ranges with an optional inclusion of specific data values.

31.3.1 Shared Debug Resources

The debug module revision A implementation provides a common hardware structure for BDM and breakpoint functionality. Certain hardware structures are used for BDM and breakpoint purposes as shown in Table 31-4.

Table 31-4. Shared BDM/Breakpoint Hardware

Register	BDM Function	Breakpoint Function
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for all BDM write commands	Data for data breakpoint

Therefore, loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, if an operand address breakpoint is loaded into the debug module, a BDM command to access memory overwrites an address breakpoint in ABHR. If a data breakpoint is configured, a BDM write command overwrites the data breakpoint in DBR.

Revision B added hardware registers to eliminate these shared functions. The BAAR is used to specify bus attributes for BDM memory commands and has the same format as the LSB of the AATR. The registers containing the BDM memory address and the BDM data are not program visible.

31.3.2 Configuration/Status Register (CSR)

The CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is write-only from the programming model. It can be read from and written to through the BDM port. CSR is accessible in supervisor mode as debug control register 0x00 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.

DRc[4:0]: 0x00 (CSR)

Access: Supervisor write-only
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BSTAT				FOF	TRG	HALT	BKPT	HRL				0	BKD	PCD	IPW
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAP	TRC	EMU	DDC		UHE	BTB		0	NPL	IPI	SSM	0	0	FDBG	DBGH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-2. Configuration/Status Register (CSR)

Table 31-5. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint Status. Provides read-only status (from the BDM port only) information concerning hardware breakpoints. BSTAT is cleared by a TDR write or by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpoint 0110 Level-2 breakpoint triggered Else Reserved
27 FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM. FOF is cleared when CSR is read (from the BDM port only).
26 TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command or reading CSR (from the BDM port only) clear TRG.
25 HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clear HALT.
24 BKPT	Breakpoint assert. If BKPT is set, $\overline{\text{BKPT}}$ was asserted, forcing the processor into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clear BKPT.
23–20 HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator could use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ (This is the value used for this device) 1011 Revision D+ 1111 Revision D+PSTB
19	Reserved, must be cleared.
18 BKD	Breakpoint disable. Disables the normal $\overline{\text{BKPT}}$ input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17 PCD	PST/DDATA Disable. Disables the PST/DDATA output signal. PSTCLK is unaffected, it remains under the control of the SYNCR[DISCLK] bit. 0 Normal operation 1 Disables the generation of the PSTDDATA output signals, and forces these signals to remain quiescent **Note:** When PCD is set, do not execute a wddata instruction or perform any debug captures. Doing so, hangs the device.
16 IPW	Inhibit processor writes. Setting IPW inhibits processor-initiated writes to the debug module's programming model registers. Only commands from the external development system can modify IPW.

Table 31-5. CSR Field Descriptions (continued)

Field	Description
15 MAP	Force processor references in emulator mode. 0 All emulator-mode references are mapped into supervisor code and data spaces. 1 The processor maps all references while in emulator mode to a special address space, TT equals 10, TM equals 101 or 110. The internal SRAM and caches are disabled.
14 TRC	Force emulation mode on trace exception. 0 The processor enters supervisor mode 1 The processor enters emulator mode when a trace exception occurs
13 EMU	Force emulation mode. 0 Do not force emulator mode 1 The processor begins executing in emulator mode. See Section 31.4.2.2, “Emulator Mode”.
12–11 DDC	Debug data control. Controls operand data capture for DDATA, which displays the number of bytes defined by the operand reference size before the actual data; byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple PSTCLK clock cycles). See Table 31-24. 00 No operand data is displayed. 01 Capture all write data. 10 Capture all read data. 11 Capture all read and write data.
10 UHE	User halt enable. Selects the CPU privilege level required to execute the HALT instruction. 0 HALT is a supervisor-only instruction. 1 HALT is a supervisor/user instruction.
9–8 BTB	Branch target bytes. Defines the number of bytes of branch target address DDATA displays. 00 0 bytes 01 Lower 2 bytes of the target address 10 Lower 3 bytes of the target address 11 Entire 4-byte target address See Section 31.4.4.1, “Begin Execution of Taken Branch (PST = 0x5)”.
7	Reserved, must be cleared.
6 NPL	Non-pipelined mode. Determines whether the core operates in pipelined mode or not. 0 Pipelined mode 1 Non-pipelined mode. The processor effectively executes one instruction at a time with no overlap. This adds at least 5 cycles to the execution time of each instruction. Given an average execution latency of 1.6 cycles/instruction, throughput in non-pipeline mode would be 6.6 cycles/instruction, approximately 25% or less of pipelined performance. Regardless of the NPL state, a triggered PC breakpoint is always reported before the triggering instruction executes. In normal pipeline operation, occurrence of an address and/or data breakpoint trigger is imprecise. In non-pipeline mode, triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise. An address or data breakpoint should always occur before the next instruction begins execution. Therefore, the occurrence of the address/data breakpoints should be guaranteed.
5 IPI	Ignore pending interrupts. 0 Core services any pending interrupt requests that were signalled while in single-step mode. 1 Core ignores any pending interrupt requests signalled while in single-instruction-step mode.
4 SSM	Single-Step Mode. Setting SSM puts the processor in single-step mode. 0 Normal mode. 1 Single-step mode. The processor halts after execution of each instruction. While halted, any BDM command can be executed. On receipt of the GO command, the processor executes the next instruction and halts again. This process continues until SSM is cleared.

Table 31-5. CSR Field Descriptions (continued)

Field	Description
3–2	Reserved, must be cleared.
1 FDBG	Force the debug mode core output signal (to the on-chip peripherals). The debug mode output is logically defined as: Debug mode output = CSR[FDBG] | (CSR[DBGH] and Core is halted) 0 Debug mode output is not forced asserted. 1 Debug mode output core output signal is asserted.
0 DBGH	Disable debug signal assertion during core halt. The debug mode output (to the on-chip peripherals) is logically defined as: Debug mode output = CSR[FDBG] | (CSR[DBGH] and Core is halted) 0 Debug mode output is asserted when the core is halted. 1 Debug mode output is not asserted when the core is halted.

31.3.3 BDM Address Attribute Register (BAAR)

The BAAR register defines the address space for memory-referencing BDM commands. BAAR[R, SZ] are loaded directly from the BDM command, while the low-order 5 bits can be programmed from the external development system. To maintain compatibility with revision A, BAAR is loaded any time the AATR is written. The BAAR is initialized to a value of 0x05, setting supervisor data as the default address space.

DRc[4:0]: 0x05 (BAAR)

Access: Supervisor write-only
BDM write-only

	7	6	5	4	3	2	1	0
R								
W	R	SZ		TT		TM		
Reset:	0	0	0	0	0	1	0	1

Figure 31-3. BDM Address Attribute Register (BAAR)

Table 31-6. BAAR Field Descriptions

Field	Description
7 R	Read/Write. 0 Write 1 Read
6–5 SZ	Size. 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	Transfer Type. See the TT definition in the AATR description, Section 31.3.4, “Address Attribute Trigger Register (AATR)”.
2–0 TM	Transfer Modifier. See the TM definition in the AATR description, Section 31.3.4, “Address Attribute Trigger Register (AATR)”.

31.3.4 Address Attribute Trigger Register (AATR)

The AATR defines address attributes and a mask to be matched in the trigger. The register value is compared with address attribute signals from the processor's local high-speed bus, as defined by the setting of the trigger definition register (TDR). AATR is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x06 (AATR)

Access: Supervisor write-only
BDM write-only

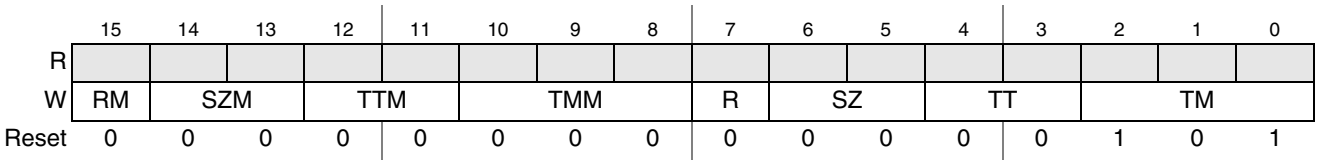


Figure 31-4. Address Attribute Trigger Register (AATR)

Table 31-7. AATR Field Descriptions

Field	Description
15 RM	Read/write Mask. Setting RM masks R in address comparisons.
14–13 SZM	Size Mask. Setting an SZM bit masks the corresponding SZ bit in address comparisons.
12–11 TTM	Transfer Type Mask. Setting a TTM bit masks the corresponding TT bit in address comparisons.
10–8 TMM	Transfer Modifier Mask. Setting a TMM bit masks the corresponding TM bit in address comparisons.
7 R	Read/Write. R is compared with the R/$\overline{W}$ signal of the processor's local bus.
6–5 SZ	Size. Compared to the processor's local bus size signals. 00 Longword 01 Byte 10 Word 11 Reserved

Table 31-7. AATR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4–3
TT</td><td>Transfer Type. Compared with the local bus transfer type signals.
00 Normal processor access
01 Reserved
10 Emulator mode access
11 Acknowledge/CPU space access
These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding indicates an external or DMA access (for backward compatibility). These bits affect the TM bits.</td></tr>
<tr><td>2–0
TM</td><td>Transfer Modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type. These bits also define the TM encoding for BDM memory commands (for backward compatibility).
<table>
<tr><th>TM</th><th>TT=00
(normal mode)</th><th>TT=10
(emulator mode)</th><th>TT=11
(acknowledge/CPU space transfers)</th></tr>
<tr><td>000</td><td>Reserved</td><td>Reserved</td><td>CPU space access</td></tr>
<tr><td>001</td><td>User data access</td><td>Reserved</td><td>Interrupt ack level 1</td></tr>
<tr><td>010</td><td>User code access</td><td>Reserved</td><td>Interrupt ack level 2</td></tr>
<tr><td>011</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 3</td></tr>
<tr><td>100</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 4</td></tr>
<tr><td>101</td><td>Supervisor data access</td><td>Emulator mode access</td><td>Interrupt ack level 5</td></tr>
<tr><td>110</td><td>Supervisor code access</td><td>Emulator code access</td><td>Interrupt ack level 6</td></tr>
<tr><td>111</td><td>Reserved</td><td>Reserved</td><td>Interrupt ack level 7</td></tr>
</table>
</td></tr>
</table>

31.3.5 Trigger Definition Register (TDR)

The TDR configures the operation of the hardware breakpoint logic corresponding with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as a one- or two-level trigger. TDR[31–16] bits define second-level trigger, and bits 15–0 define first-level trigger.

NOTE

The debug module has no hardware interlocks to prevent spurious breakpoint triggers while the breakpoint registers are being loaded. Disable TDR (by clearing TDR[29,13]) before defining triggers.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT]. TDR is accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x07 (TDR)

Access: Supervisor write-only
BDM write-only

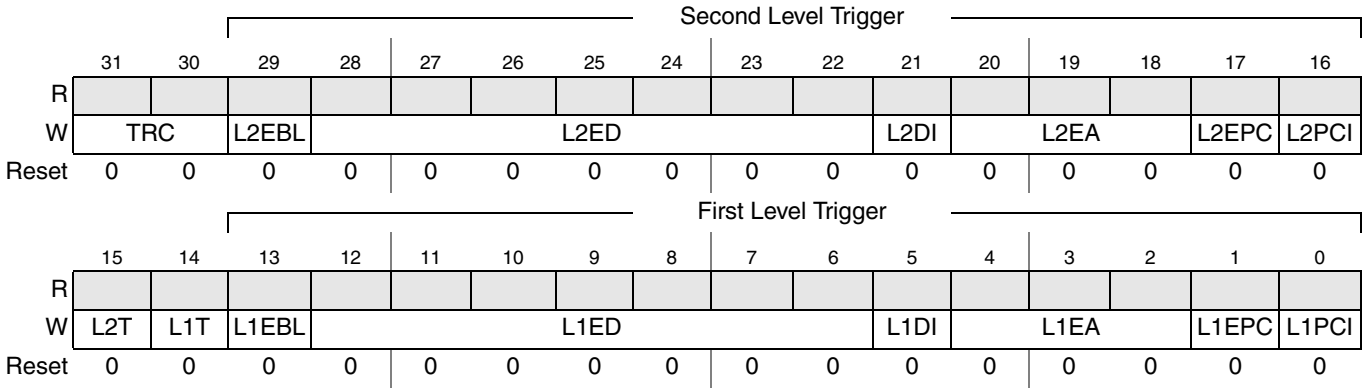


Figure 31-5. Trigger Definition Register (TDR)

Table 31-8. TDR Field Descriptions

Field	Description
31–30 TRC	Trigger Response Control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on DDATA. 00 Display on DDATA only 01 Processor halt 10 Debug interrupt 11 Reserved
29 L2EBL	Enable Level 2 Breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 2 breakpoints 1 Enables all level 2 breakpoint triggers
28–22 L2ED	Enable Level 2 Data Breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all ED bits disables data breakpoints.
21 L2DI	Level 2 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents. 0 No inversion 1 Invert data breakpoint comparators.

TDR Bit	Description
28	Data longword. Entire processor's local data bus.
27	Lower data word.
26	Upper data word.
25	Lower lower data byte. Low-order byte of the low-order word.
24	Lower middle data byte. High-order byte of the low-order word.
23	Upper middle data byte. Low-order byte of the high-order word.
22	Upper upper data byte. High-order byte of the high-order word.

Table 31-8. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20–18
L2EA</td><td>Enable Level 2 Address Breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>17
L2EPC</td><td>Enable Level 2 PC Breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint where the trigger is defined by the logical summation of:
(PBR0 and $\overline{\text{PBMR}}$) | PBR1 | PBR2 | PBR3 Eqn. 31-1</td></tr>
<tr><td>16
L2PCI</td><td>Level 2 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBRn and PBMR.
1 The PC breakpoint is defined outside the region defined by PBRn and PBMR.</td></tr>
<tr><td>15
L2T</td><td>Level 2 Trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers.
0 Level 2 trigger = PC_condition & Address_range & Data_condition
1 Level 2 trigger = PC_condition | (Address_range & Data_condition)
Note: Debug Rev A only had the AND condition available for the triggers.</td></tr>
<tr><td>14
L1T</td><td>Level 1 Trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers.
0 Level 1 trigger = PC_condition & Address_range & Data_condition
1 Level 1 trigger = PC_condition | (Address_range & Data_condition)
Note: Debug Rev A only had the AND condition available for the triggers.</td></tr>
<tr><td>13
L1EBL</td><td>Enable Level 1 Breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
</table>

Table 31-8. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12–6
L1ED</td><td>Enable Level 1 Data Breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor's local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>5
L1DI</td><td>Level 1 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>4–2
L1EA</td><td>Enable Level 1 Address Breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>1
L1EPC</td><td>Enable Level 1 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>0
L1PCI</td><td>Level 1 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBRn and PBMR.
1 The PC breakpoint is defined outside the region defined by PBRn and PBMR.</td></tr>
</table>

31.3.6 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBR*n* registers define an instruction address for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR is configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR. Breakpoint registers, PBR1–3, have no masking associated with them. The

contents of the breakpoint registers are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command using values shown in Section 31.4.1.5, "BDM Command Set".

DRc[4:0]: 0x08 (PBR0)

Access: Supervisor write-only
BDM write-only

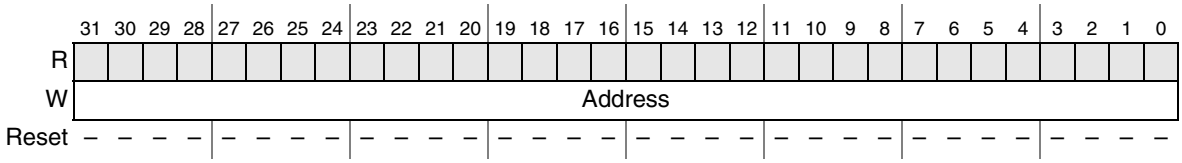


Figure 31-6. PC Breakpoint Register (PBR0)

Table 31-9. PBR0 Field Descriptions

Field	Description
31–0 Address	PC Breakpoint Address. The address to be compared with the PC as a breakpoint trigger. **Note:** PBR0[0] should always be loaded with a 0.

DRc[4:0]: 0x18 (PBR1)
0x1A (PBR2)
0x1B (PBR3)

Access: Supervisor write-only
BDM write-only

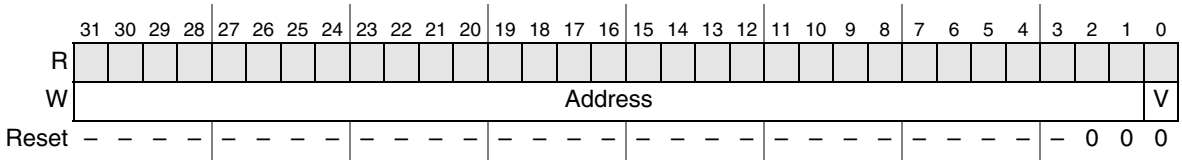


Figure 31-7. PC Breakpoint Register *n* (PBR*n*)

Table 31-10. PBR*n* Field Descriptions

Field	Description
31–1 Address	PC Breakpoint Address. The 31-bit address to be compared with the PC as a breakpoint trigger.
0 V	Valid Bit. This bit must be set for the PC breakpoint to occur at the address specified in the Address field. 0 PBR is disabled. 1 PBR is enabled.

Figure 31-8 shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command. PBMR only masks PBR0.

DRc[4:0]: 0x09 (PBMR)

Access: Supervisor write-only
BDM write-only

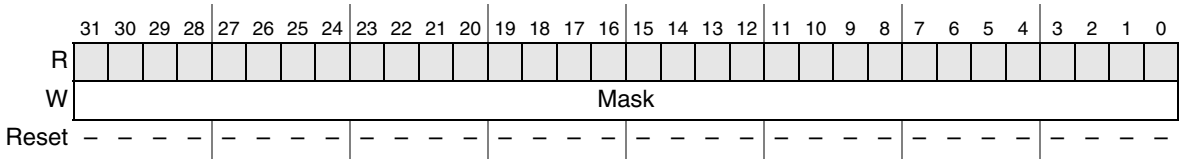


Figure 31-8. PC Breakpoint Mask Register (PBMR)

Table 31-11. PBMR Field Descriptions

Field	Description
31–0 Mask	PC Breakpoint Mask. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

31.3.7 Address Breakpoint Registers (ABLR, ABHR)

The ABLR and ABHR define regions in the processor's data address space that can act as part of the trigger. These register values are compared with the address for each transfer on the processor's high-speed local bus. The trigger definition register (TDR) identifies the trigger as one of three cases:

- Identically the value in ABLR
- Inside the range bound by ABLR and ABHR inclusive
- Outside that same range

ABLR and ABHR are accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command.

DRc[4:0]: 0x0C (ABHR)
0x0D (ABLR)

Access: Supervisor write-only
BDM write-only

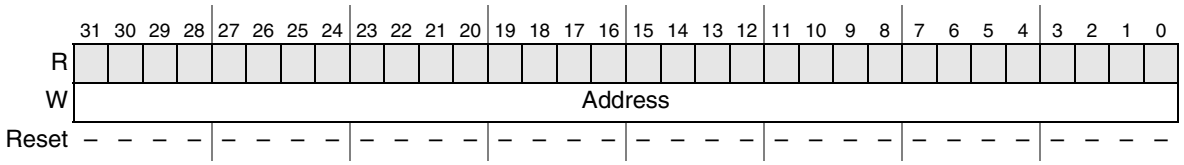


Figure 31-9. Address Breakpoint Registers (ABLR, ABHR,)

Table 31-12. ABLR Field Description

Field	Description
31–0 Address	Low Address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific single addresses are programmed into ABLR.

Table 31-13. ABHR Field Description

Field	Description
31–0 Address	High Address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

31.3.8 Data Breakpoint and Mask Registers (DBR, DBMR)

The data breakpoint register (DBR), specify data patterns used as part of the trigger into debug mode. DBR bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR and DBMR are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x0E (DBR)

Access: Supervisor write-only
BDM write-only

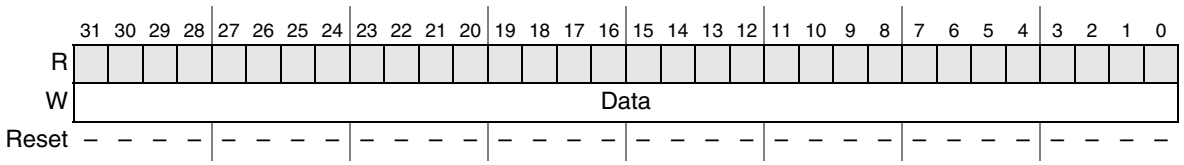


Figure 31-10. Data Breakpoint Registers (DBR)

Table 31-14. DBR Field Descriptions

Field	Description
31–0 Data	Data Breakpoint Value. Contains the value to be compared with the data value from the processor's local bus as a breakpoint trigger.

DRc[4:0]: 0x0F (DBMR)

Access: Supervisor write-only
BDM write-only

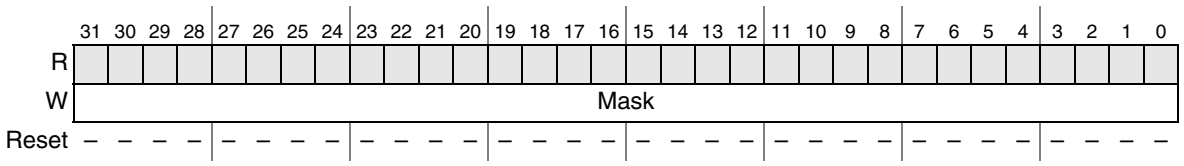


Figure 31-11. Data Breakpoint Mask Registers (DBMR)

Table 31-15. DBMR Field Descriptions

Field	Description
31–0 Mask	Data Breakpoint Mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBMR bit allows the corresponding DBR bit to be compared to the appropriate bit of the processor's local data bus. Setting a DBMR bit causes that bit to be ignored.

The DBR supports aligned and misaligned references. Table 31-16 shows relationships between processor address, access size, and location within the 32-bit data bus.

Table 31-16. Address, Access Size, and Operand Data Location

Address[1:0]	Access Size	Operand Location
00	Byte	D[31:24]
01	Byte	D[23:16]
10	Byte	D[15:8]
11	Byte	D[7:0]
0x	Word	D[31:16]
1x	Word	D[15:0]
xx	Longword	D[31:0]

31.4 Functional Description

31.4.1 Background Debug Mode (BDM)

The ColdFire family implements a low-level system debugger in the microprocessor in a dedicated hardware module. Communication with the development system is managed through a dedicated, high-speed serial command interface. Although some BDM operations, such as CPU register accesses, require the CPU to be halted, other BDM commands, such as memory accesses, can be executed while the processor is running.

BDM is useful because:

- In-circuit emulation is not needed, so physical and electrical characteristics of the system are not affected.
- BDM is always available for debugging the system and provides a communication link for upgrading firmware in existing systems.
- Provides high-speed cache downloading (500 Kbytes/sec), especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This allows quick hardware debugging with the same tool set used for firmware development.

31.4.1.1 CPU Halt

Although most BDM operations can occur in parallel with CPU operations, unrestricted BDM operation requires the CPU to be halted. The sources that can cause the CPU to halt are listed below in order of priority:

1. A catastrophic fault-on-fault condition automatically halts the processor.
2. A hardware breakpoint trigger can generate a pending halt condition similar to the assertion of $\overline{\text{BKPT}}$. This type of halt is always first marked as pending in the processor, which samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point. See Section 31.4.2.1, “Theory of Operation”.

3. The execution of a HALT instruction immediately suspends execution. Attempting to execute HALT in user mode while CSR[UHE] is cleared generates a privilege violation exception. If CSR[UHE] is set, HALT can be executed in user mode. After HALT executes, the processor can be restarted by serial shifting a GO command into the debug module. Execution continues at the instruction after HALT.
4. The assertion of the $\overline{\text{BKPT}}$ input is treated as a pseudo-interrupt; asserting $\overline{\text{BKPT}}$ creates a pending halt postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction; if a pending halt is detected, the processor suspends execution and enters the halted state.

The are two special cases involving the assertion of $\overline{\text{BKPT}}$:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input is asserted within eight cycles after $\overline{\text{RESET}}$ is negated, the processor enters the halt state, signaling halt status (0xF) on the PST outputs. While the processor is in this state, all resources accessible through the debug module can be referenced. This is the only chance to force the processor into emulation mode through CSR[EMU].
- After system initialization, the processor's response to the GO command depends on the set of BDM commands performed while it is halted for a breakpoint. Specifically, if the PC register was loaded, the GO command causes the processor to exit halted state and pass control to the instruction address in the PC, bypassing normal reset exception processing. If the PC was not loaded, the GO command causes the processor to exit halted state and continue reset exception processing.
- The ColdFire architecture also manages a special case of $\overline{\text{BKPT}}$ asserted while the processor is stopped by execution of the STOP instruction. For this case, the processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction, which follows the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions.

31.4.1.2 BDM Serial Interface

When the CPU is halted and PST reflects the halt status, the development system can send unrestricted commands to the debug module. The debug module implements a synchronous serial protocol using two inputs (DSCLK and DSI) and one output (DSO), where DSO is specified as a delay relative to the rising edge of the processor clock. See Table 31-2. The development system serves as the serial communication channel master and must generate DSCLK.

The serial channel operates at a frequency from DC to 1/5 of the PSTCLK frequency. The channel uses full-duplex mode, where data is sent and received simultaneously by master and slave devices. The transmission consists of 17-bit packets composed of a status/control bit and a 16-bit data word. As shown in Figure 31-12, all state transitions are enabled on a rising edge of the PSTCLK clock when DSCLK is high; DSI is sampled and DSO is driven.

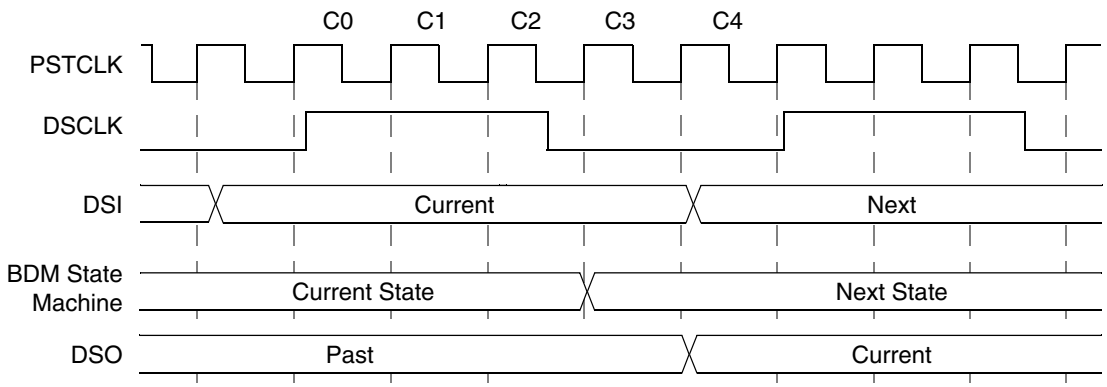


Figure 31-12. Maximum BDM Serial Interface Timing

DSCLK and DSI are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled, along with DSI, on the rising edge of PSTCLK. DSO is delayed from the DSCLK-enabled PSTCLK rising edge (registered after a BDM state machine state change). All events in the debug module's serial state machine are based on the PSTCLK rising edge. DSCLK must also be sampled low (on a positive edge of PSTCLK) between each bit exchange. The msb is sent first. Because DSO changes state based on an internally recognized rising edge of DSCLK, DSO cannot be used to indicate the start of a serial transfer. The development system must count clock cycles in a given transfer. C0–C4 are described as:

- C0: Set the state of the DSI bit
- C1: First synchronization cycle for DSI (DSCLK is high)
- C2: Second synchronization cycle for DSI (DSCLK is high)
- C3: BDM state machine changes state depending upon DSI and whether the entire input data transfer has been transmitted
- C4: DSO changes to next value

NOTE

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

31.4.1.3 Receive Packet Format

The basic receive packet consists of 16 data bits and 1 status bit

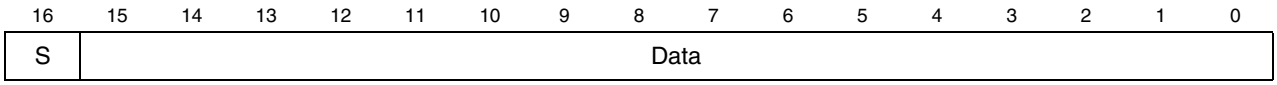


Figure 31-13. Receive BDM Packet

Table 31-17. Receive BDM Packet Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16
S</td><td>Status. Indicates the status of CPU-generated messages listed below. The not-ready response can be ignored unless a memory-referencing cycle is in progress. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.
<table>
<tr><th>S</th><th>Data</th><th>Message</th></tr>
<tr><td>0</td><td>xxxx</td><td>Valid data transfer</td></tr>
<tr><td>0</td><td>FFFF</td><td>Status OK</td></tr>
<tr><td>1</td><td>0000</td><td>Not ready with response; come again</td></tr>
<tr><td>1</td><td>0001</td><td>Error-Terminated bus cycle; data invalid</td></tr>
<tr><td>1</td><td>FFFF</td><td>Illegal Command</td></tr>
</table></td></tr>
<tr><td>15–0
Data</td><td>Data. Contains the message to be sent from the debug module to the development system. The response message is always a single word, with the data field encoded as shown above.</td></tr>
</table>

31.4.1.3.1 Transmit Packet Format

The basic transmit packet consists of 16 data bits and 1 reserved bit.

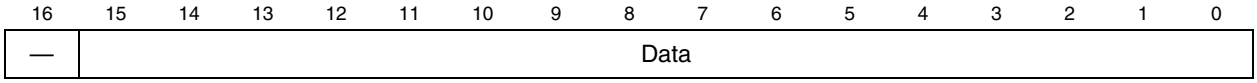


Figure 31-14. Transmit BDM Packet

Table 31-18. Transmit BDM Packet Field Description

Field	Description
16	Reserved, must be cleared.
15–0 Data	Data bits 15–0. Contains the data to be sent from the development system to the debug module.

31.4.1.3.2 BDM Command Format

All ColdFire family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words.

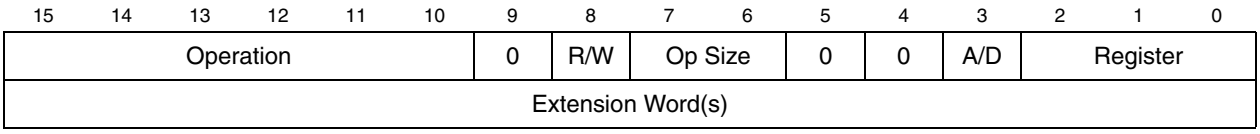


Figure 31-15. BDM Command Format

Table 31-19. BDM Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–10
Operation</td><td>Specifies the command. These values are listed in Table 31-20.</td></tr>
<tr><td>9</td><td>Reserved, must be cleared.</td></tr>
<tr><td>8
R/W</td><td>Direction of operand transfer.
0 Data is written to the CPU or to memory from the development system.
1 The transfer is from the CPU to the development system.</td></tr>
<tr><td>7–6
Op Size</td><td>Operand Data Size for Sized Operations. Addresses are expressed as 32-bit absolute values. A command performing a byte-sized memory read leaves the upper 8 bits of the response data undefined. Referenced data is returned in the lower 8 bits of the response.
<table>
<tr><th></th><th>Operand Size</th><th>Bit Values</th></tr>
<tr><td>00</td><td>Byte</td><td>8 bits</td></tr>
<tr><td>01</td><td>Word</td><td>16 bits</td></tr>
<tr><td>10</td><td>Longword</td><td>32 bits</td></tr>
<tr><td>11</td><td>Reserved</td><td>—</td></tr>
</table></td></tr>
<tr><td>5–4</td><td>Reserved, must be cleared.</td></tr>
<tr><td>3
A/D</td><td>Address/Data. Determines whether the register field specifies a data or address register.
0 Data register.
1 Address register.</td></tr>
<tr><td>2–0
Register</td><td>Contains the register number in commands that operate on processor registers. See Table 31-21.</td></tr>
</table>

31.4.1.3.3 Extension Words as Required

Some commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Longword accesses are forcibly longword-aligned and word accesses are forcibly word-aligned. Immediate data can be 1 or 2 words long. Byte and word data each requires a single extension word, while longword data requires two extension words.

Operands and addresses are transferred most-significant word first. In the following descriptions of the BDM command set, the optional set of extension words is defined as address, data, or operand data.

31.4.1.4 Command Sequence Diagrams

The command sequence diagram in Figure 31-16 shows serial bus traffic for commands. Each bubble represents a 17-bit bus transfer. The top half of each bubble indicates the data the development system sends to the debug module; the bottom half indicates the debug module's response to the previous development system commands. Command and result transactions overlap to minimize latency.

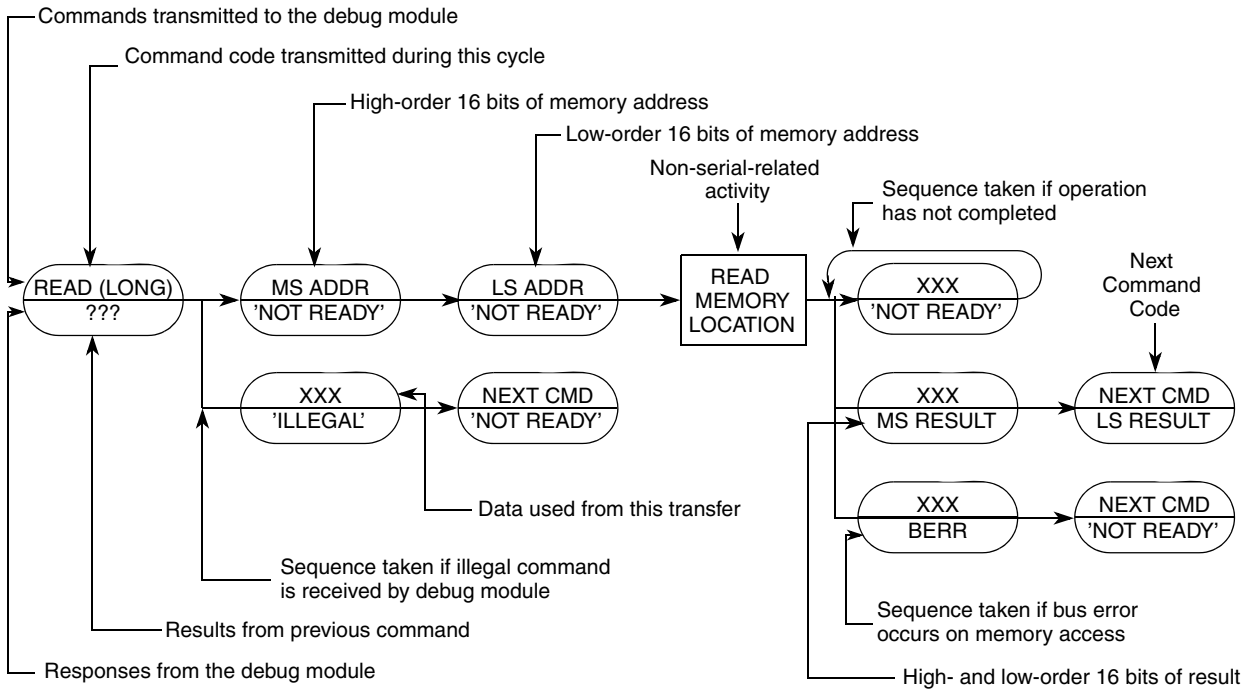


Figure 31-16. Command Sequence Diagram

The sequence is as follows:

- In cycle 1, the development system command is issued (READ in this example). The debug module responds with the low-order results of the previous command or a command complete status of the previous command, if no results are required.
- In cycle 2, the development system supplies the high-order 16 address bits. The debug module returns a not-ready response unless the received command is decoded as unimplemented, which is indicated by the illegal command encoding. If this occurs, the development system should retransmit the command.

NOTE

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

- In cycle 3, the development system supplies the low-order 16 address bits. The debug module always returns a not-ready response.
- At the completion of cycle 3, the debug module initiates a memory read operation. Any serial transfers that begin during a memory access return a not-ready response.
- Results are returned in the two serial transfer cycles after the memory access completes. For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined and the referenced data is returned in the lower 8 bits. The next command's opcode is sent to the debug module during the final transfer. If a bus error terminates a memory or register access, error status (S = 1, DATA = 0x0001) returns instead of result data.

31.4.1.5 BDM Command Set

Table 31-20 summarizes the BDM command set. Subsequent sections contain detailed descriptions of each command. Issuing a BDM command when the processor is accessing debug module registers using the WDEBUG instruction causes undefined behavior. See Table 31-21 for register address encodings.

Table 31-20. BDM Command Summary

Command	Mnemonic	Description	CPU State[1]	Section/Page	Command (Hex)
Read A/D register	RAREG/ RDREG	Read the selected address or data register and return the results through the serial interface.	Halted	31.4.1.5.1/31-24	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/ WDREG	Write the data operand to the specified address or data register.	Halted	31.4.1.5.2/31-24	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	31.4.1.5.3/31-25	0x1900—byte 0x1940—word 0x1980—lword
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	31.4.1.5.4/31-26	0x1800—byte 0x1840—word 0x1880—lword
Dump memory block	DUMP	Used with READ to dump large blocks of memory. An initial READ executes to set up the starting address of the block and to retrieve the first result. A DUMP command retrieves subsequent operands.	Steal	31.4.1.5.5/31-28	0x1D00—byte 0x1D40—word 0x1D80—lword
Fill memory block	FILL	Used with WRITE to fill large blocks of memory. An initial WRITE executes to set up the starting address of the block and to supply the first operand. A FILL command writes subsequent operands.	Steal	31.4.1.5.6/31-30	0x1C00—byte 0x1C40—word 0x1C80—lword
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	31.4.1.5.7/31-31	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	31.4.1.5.8/31-32	0x0000
Output the current PC	SYNC_PC	Capture the current PC and display it on the PST/DDATA outputs.	Parallel	31.4.1.5.9/31-32	0x0001
Read control register	RCREG	Read the system control register.	Halted	31.4.1.5.10/31-33	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	31.4.1.5.13/31-35	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	31.4.1.5.14/31-36	0x2D {0x4[2] DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	31.4.1.5.15/31-37	0x2C {0x4[2] DRc[4:0]}

[1] General command effect and/or requirements on CPU operation:
- Halted: The CPU must be halted to perform this command.
- Steal: Command generates bus cycles that can be interleaved with bus accesses.
- Parallel: Command is executed in parallel with CPU activity.

[2] 0x4 is a three-bit field.

Freescale reserves unassigned command opcodes. All unused command formats within any revision level perform a NOP and return the illegal command response.

The following sections describe the commands summarized in Table 31-20.

NOTE

The BDM status bit (S) is 0 for normally completed commands. S is set for illegal commands, not-ready responses, and transfers with bus-errors. Section 31.4.1.2, "BDM Serial Interface," describes the receive packet format.

31.4.1.5.1 Read A/D Register (RAREG/RDREG)

Read the selected address or data register and return the 32-bit result. A bus error response is returned if the CPU core is not halted.

Command/Result Formats:

	15 14 13 12	11 10 9 8	7 6 5 4	3	2 1 0
Command	0x2	0x1	0x8	A/D	Register
Result	D[31:16]				
	D[15:0]				

Figure 31-17. RAREG/RDREG Command Format

Command Sequence:

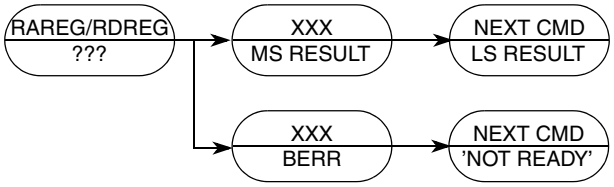


Figure 31-18. RAREG/RDREG Command Sequence

Operand Data: None

Result Data: The contents of the selected register are returned as a longword value, most-significant word first.

31.4.1.5.2 Write A/D Register (WAREG/WDREG)

The operand longword data is written to the specified address or data register. A write alters all 32 register bits. A bus error response is returned if the CPU core is not halted.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0x0				0x8				A/D	Register		
D[31:16]															
D[15:0]															

Figure 31-19. WAREG/WDREG Command Format

Command Sequence:

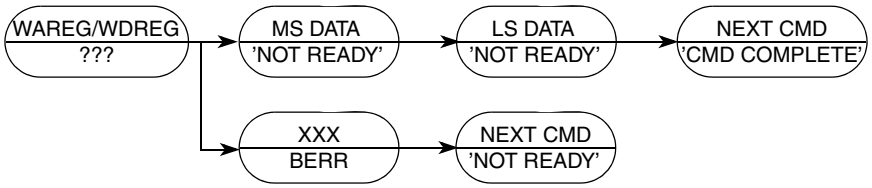


Figure 31-20. WAREG/WDREG Command Sequence

Operand Data: Longword data is written into the specified address or data register. The data is supplied most-significant word first.

Result Data: Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete.

31.4.1.5.3 Read Memory Location (READ)

Read data at the longword address. Address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command/Result Formats:

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	Command	0x1				0x9				0x0				0x0			
		A[31:16]															
		A[15:0]															
	Result	X	X	X	X	X	X	X	X	D[7:0]							
Word	Command	0x1				0x9				0x4				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[15:0]															
Longword	Command	0x1				0x9				0x8				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[31:16]															
		D[15:0]															

Figure 31-21. READ Command/Result Formats

Command Sequence:

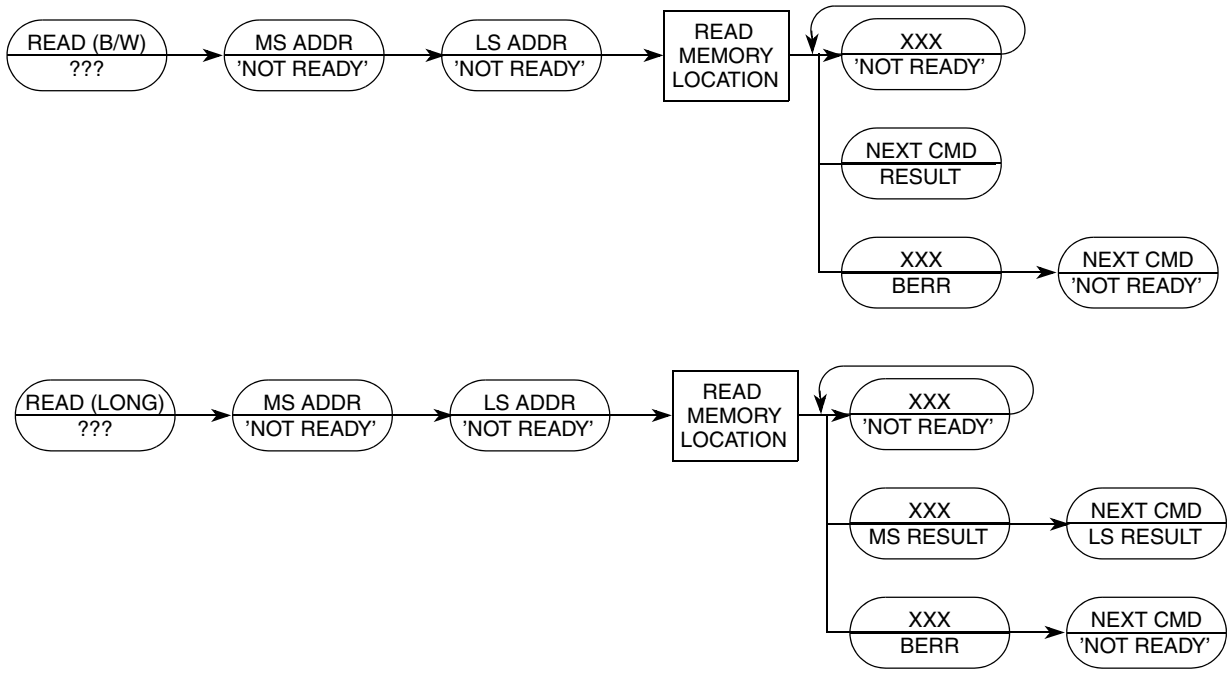


Figure 31-22. READ Command Sequence

Operand Data: The only operand is the longword address of the requested location.

Result Data: Word results return 16 bits of data; longword results return 32. Bytes are returned in the LSB of a word result; the upper byte is undefined. 0x0001 (S = 1) is returned if a bus error occurs.

31.4.1.5.4 Write Memory Location (WRITE)

Write data to the memory location specified by the longword address. BAAR[TT,TM] defines address space. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0x8				0x0				0x0			
	A[31:16]															
	A[15:0]															
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0x8				0x4				0x0			
	A[31:16]															
	A[15:0]															
	D[15:0]															
Longword	0x1				0x8				0x8				0x0			
	A[31:16]															
	A[15:0]															
	D[31:16]															
	D[15:0]															

Figure 31-23. WRITE Command Format

Command Sequence:

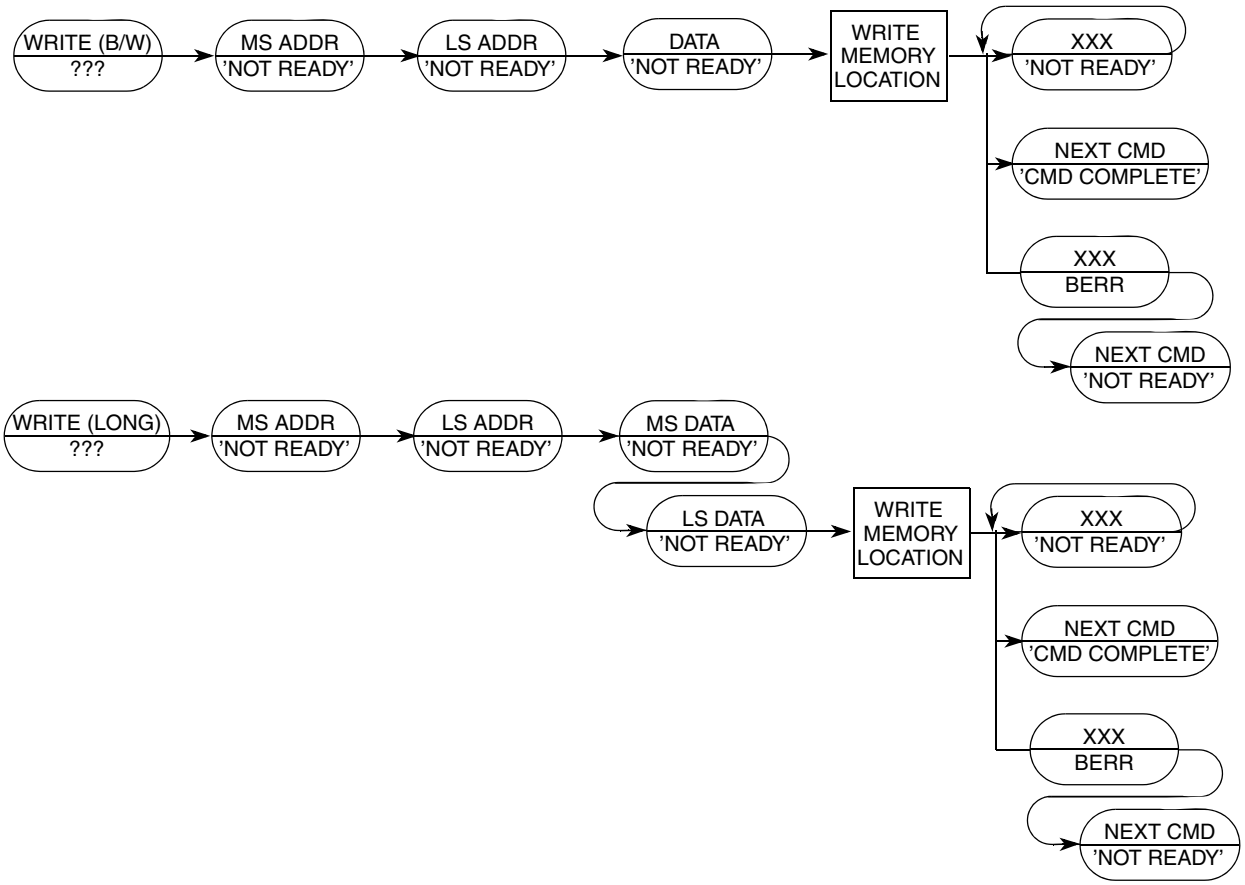


Figure 31-24. WRITE Command Sequence

Operand Data: This two-operand instruction requires a longword absolute address that specifies a location the data operand is written. Byte data is sent as a 16-bit word, justified in the LSB; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

Result Data: Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

31.4.1.5.5 Dump Memory Block (DUMP)

DUMP is used with the READ command to access large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. If an initial READ is not executed before the first DUMP, an illegal command response is returned. The DUMP command retrieves subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register. Subsequent DUMP commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register.

NOTE

DUMP does not check for a valid address; it is a valid command only when preceded by NOP, READ, or another DUMP command. Otherwise, an illegal command response is returned. NOP can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a DUMP command is processed, allowing the operand size to be dynamically altered.

Command/Result Formats:

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	Command	0x1				0xD				0x0				0x0			
	Result	X	X	X	X	X	X	X	X	D[7:0]							
Word	Command	0x1				0xD				0x4				0x0			
	Result	D[15:0]															
Longword	Command	0x1				0xD				0x8				0x0			
	Result	D[31:16]															
		D[15:0]															

Figure 31-25. DUMP Command/Result Formats

Command Sequence:

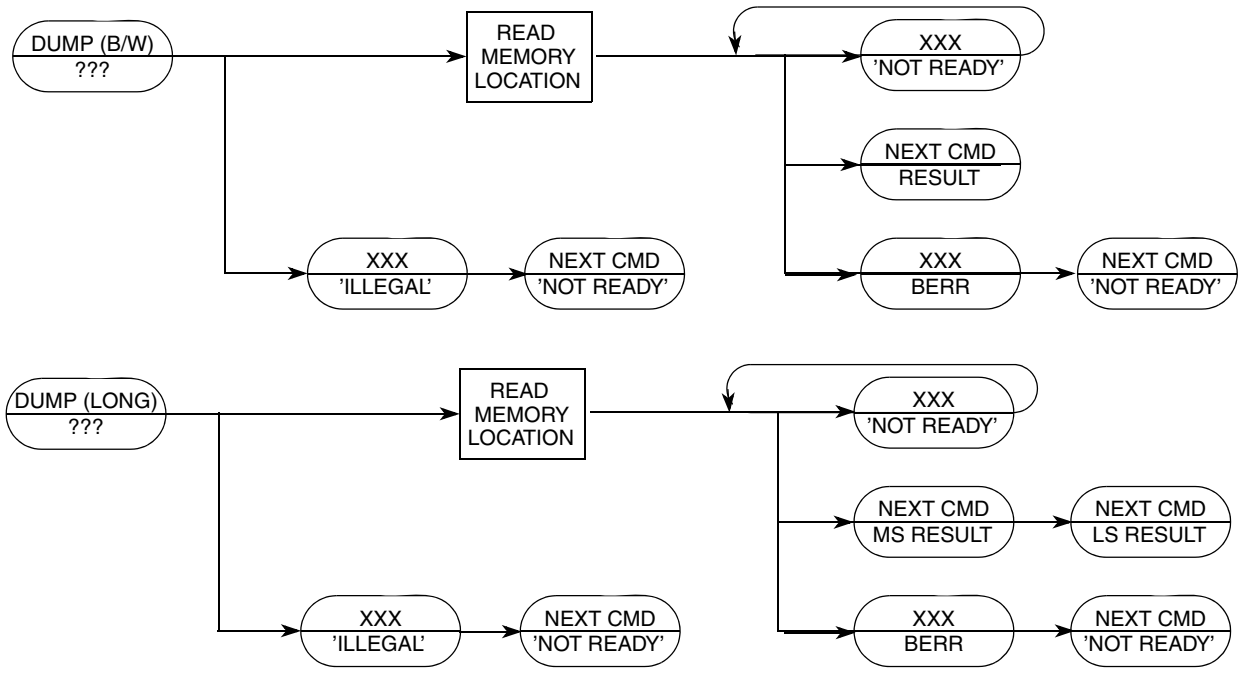


Figure 31-26. DUMP Command Sequence

Operand Data: None

Result Data: Requested data is returned as a word or longword. Byte data is returned in the least-significant byte of a word result. Word results return 16 bits of significant data; longword results return 32 bits. A value of 0x0001 (with S set) is returned if a bus error occurs.

31.4.1.5.6 Fill Memory Block (FILL)

A FILL command is used with the WRITE command to access large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. The FILL command writes subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register after the memory write. Subsequent FILL commands use this address, perform the write, increment it by the current operand size, and store the updated address in the temporary register.

If an initial WRITE is not executed preceding the first FILL command, the illegal command response is returned.

NOTE

The FILL command does not check for a valid address: FILL is a valid command only when preceded by another FILL, a NOP, or a WRITE command. Otherwise, an illegal command response is returned. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a FILL command is processed, allowing the operand size to be altered dynamically.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0xC				0x0				0x0			
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0xC				0x4				0x0			
	D[15:0]															
Longword	0x1				0xC				0x8				0x0			
	D[31:16]															
	D[15:0]															

Figure 31-27. FILL Command Format

Command Sequence:

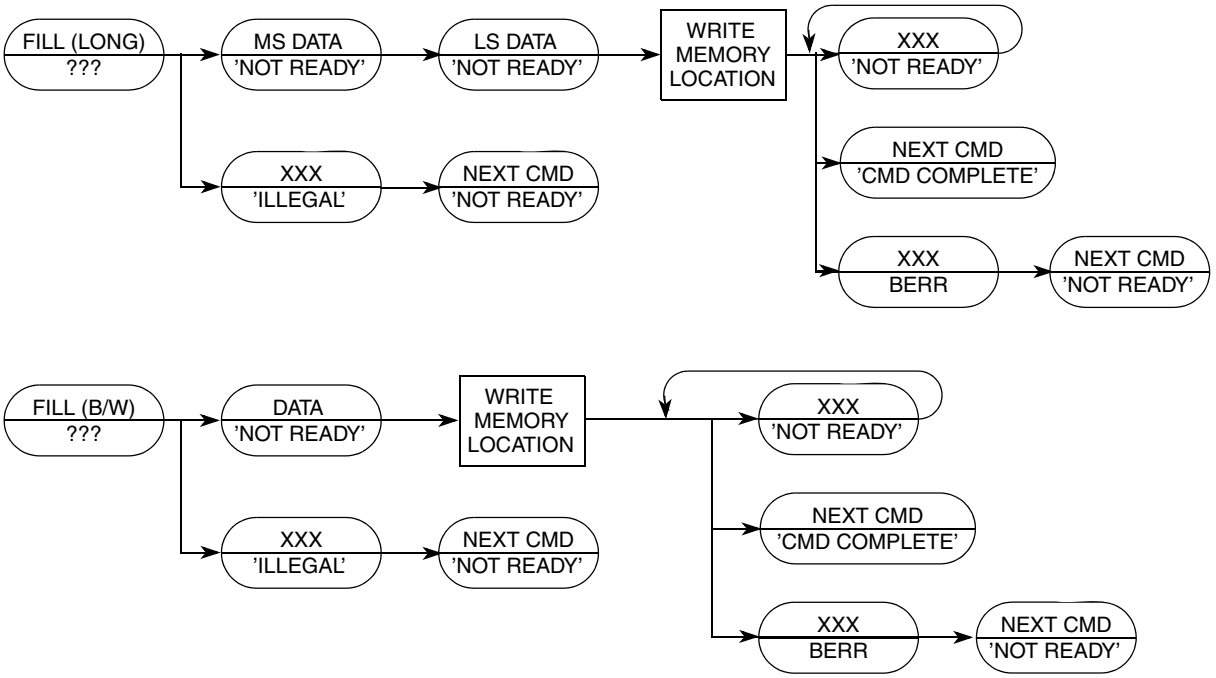


Figure 31-28. FILL Command Sequence

Operand Data: A single operand is data to be written to the memory location. Byte data is sent as a 16-bit word, justified in the least-significant byte; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

Result Data: Command complete status (0xFFFF) is returned when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

31.4.1.5.7 Resume Execution (GO)

The pipeline is flushed and refilled before normal instruction execution resumes. Prefetching begins at the current address in the PC and at the current privilege level. If any register (such as the PC or SR) is altered by a BDM command while the processor is halted, the updated value is used when prefetching resumes. If a GO command issues and the CPU is not halted, the command is ignored.

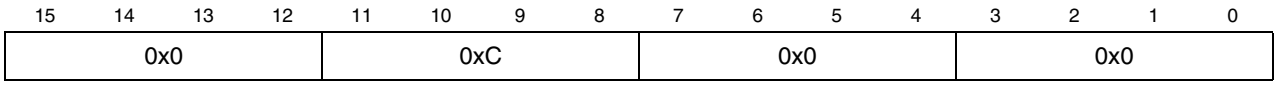


Figure 31-29. GO Command Format

Command Sequence:

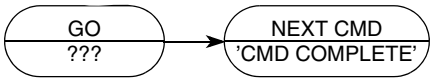


Figure 31-30. GO Command Sequence

Operand Data: None

Result Data: The command-complete response (0xFFFF) is returned during the next shift operation.

31.4.1.5.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

Command Formats:

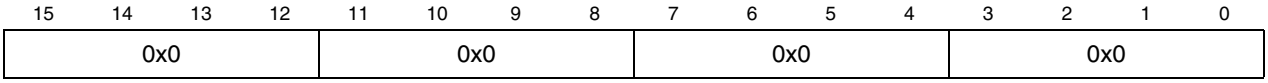


Figure 31-31. NOP Command Format

Command Sequence:

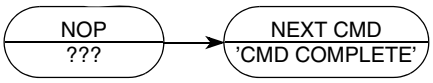


Figure 31-32. NOP Command Sequence

Operand Data: None

Result Data: The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

31.4.1.5.9 Synchronize PC to the PST/DDATA Lines (SYNC_PC)

The SYNC_PC command captures the current PC and displays it on the PST/DDATA outputs. After the debug module receives the command, it sends a signal to the ColdFire processor that the current PC must be displayed. The processor then forces an instruction fetch at the next PC with the address being captured in the DDATA logic under control of the CSR[BTB] bits. The specific sequence of PST and DDATA values is defined below:

1. Debug signals a SYNC_PC command is pending.
2. CPU completes the current instruction.
3. CPU forces an instruction fetch to the next PC, generates a PST equaling 0x5 value indicating a taken branch and signals the capture of DDATA.
4. The instruction address corresponding to the PC is captured.
5. The PST marker (0x9–0xB) is generated and displayed as defined by the CSR[BTB] bit followed by the captured PC address.

The SYNC_PC command can be used to dynamically access the PC for performance monitoring. The execution of this command is considerably less obtrusive to the real-time operation of an application than a HALT-CPU/READ-PC/RESUME command sequence.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0x0				0x0				0x1			

Figure 31-33. SYNC_PC Command Format

Command Sequence:

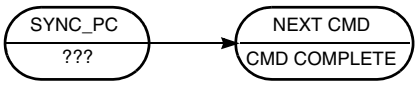


Figure 31-34. SYNC_PC Command Sequence

Operand Data: None

Result Data: Command complete status (0xFFFF) is returned when the register write is complete.

31.4.1.5.10 Read Control Register (RCREG)

Read the selected control register and return the 32-bit result. Accesses to the processor/memory control registers are always 32 bits wide, regardless of register width. The second and third words of the command form a 32-bit address, which the debug module uses to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same the processor's MOVEC instruction uses.

Command/Result Formats:

	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Command	0x2	0x9	0x8	0x0
	0x0	0x0	0x0	0x0
	0x0	Rc		
Result	D[31:16]			
	D[15:0]			

Figure 31-35. RCREG Command/Result Formats

Command Sequence:

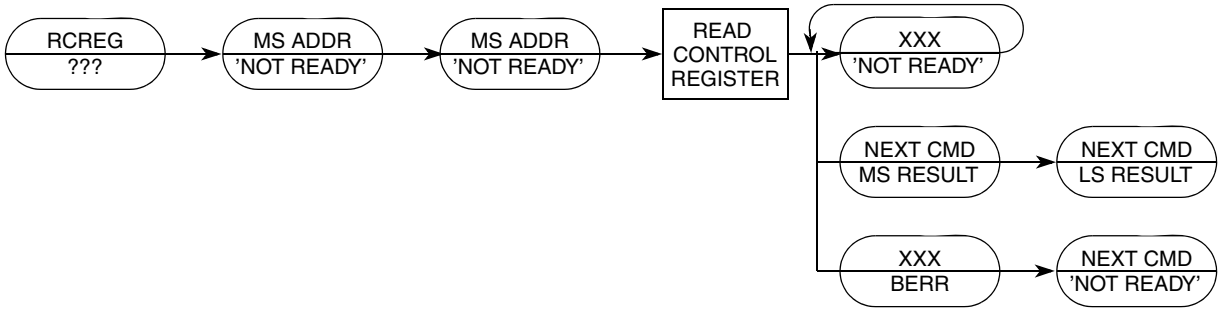


Figure 31-36. RCREG Command Sequence

Operand Data: The only operand is the 32-bit Rc control register select field.

Result Data: Control register contents are returned as a longword, most-significant word first. The implemented portion of registers smaller than 32 bits is guaranteed correct; other bits are undefined.

Rc encoding: See Table 31-21.

Table 31-21. Control Register Map

Rc	Register Definition
0x009	RGPIO Base Address Register (RGPIOBAR)[1]
0x(0,1)80 – 0x(0,1)87	Data Registers 0–7 (0 = load, 1 = store)
0x(0,1)88 – 0x(0,1)8F	Address Registers 0–7 (0 = load, 1 = store) (A7 is user stack pointer)
0x800	Other Stack Pointer (OTHER_A7)
0x801	Vector Base Register (VBR)
0x804	MAC Status Register (MACSR)
0x805	MAC Mask Register (MASK)
0x806	MAC Accumulator 0 (ACC0)
0x807	MAC Accumulator 0,1 Extension Bytes (ACCEXT01)
0x808	MAC Accumulator 2,3 Extension Bytes (ACCEXT23)
0x809	MAC Accumulator 1 (ACC1)
0x80A	MAC Accumulator 2 (ACC2)
0x80B	MAC Accumulator 3 (ACC3)
0x80E	Status Register (SR)
0x80F	Program Register (PC)
0xC04	Flash Base Address Register (FLASHBAR)
0xC05	RAM Base Address Register (RAMBAR)

[1] If an RGPIO module is available on this device.

31.4.1.5.11 BDM Accesses of the Stack Pointer Registers (A7: SSP and USP)

The ColdFire core supports two unique stack pointer (A7) registers: the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two programmable-visible 32-bit registers does not uniquely identify one as the SSP and the other as the USP. Rather, the hardware uses one 32-bit register as the currently-active A7; the other is named the OTHER_A7. Therefore, the contents of the two hardware registers is a function of the operating mode of the processor:

```
if SR[S] = 1
        then    A7 = Supervisor Stack Pointer
                OTHER_A7 = User Stack Pointer
        else    A7 = User Stack Pointer
                OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports reads and writes to A7 and OTHER_A7 directly. It is the responsibility of the external development system to determine the mapping of A7 and OTHER_A7 to the two program-visible definitions (supervisor and user stack pointers), based on the SR[S] bit.

31.4.1.5.12 BDM Accesses of the EMAC Registers

The presence of rounding logic in the output datapath of the EMAC requires special care for BDM-initiated reads and writes of its programming model. In particular, any result rounding modes must be disabled during the read/write process so the exact bit-wise EMAC register contents are accessed.

For example, a BDM read of an accumulator (ACC*x*) must be preceded by two commands accessing the MAC status register, as shown in the following sequence:

```
BdmReadACCx (
        rcreg    macsr;            // read current macsr contents and  save
        wcreg    #0,macsr;         // disable all rounding modes
        rcreg    ACCx;             // read the desired accumulator
        wcreg    #saved_data,macsr;// restore the original macsr
)
```

Likewise, to write an accumulator register, the following BDM sequence is needed:

```
BdmWriteACCx (
        rcreg    macsr;            // read current macsr contents and  save
        wcreg    #0,macsr;         // disable all rounding modes
        wcreg    #data,ACCx;       // write the desired accumulator
        wcreg    #saved_data,macsr;// restore the original macsr
)
```

Additionally, writes to the accumulator extension registers must be performed after the corresponding accumulators are updated because a write to any accumulator alters the corresponding extension register contents.

For more information on saving and restoring the complete EMAC programming model, see Section 4.3.1.2, “Saving and Restoring the EMAC Programming Model.”

31.4.1.5.13 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits. See the RCREG instruction description for the Rc encoding and for additional notes on writes to the A7 stack pointers and the EMAC programming model.

Command/Result Formats:

	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Command	0x2	0x8	0x8	0x0
	0x0	0x0	0x0	0x0
	0x0	Rc		
Result	D[31:16]			
	D[15:0]			

Figure 31-37. WCREG Command/Result Formats

Command Sequence:

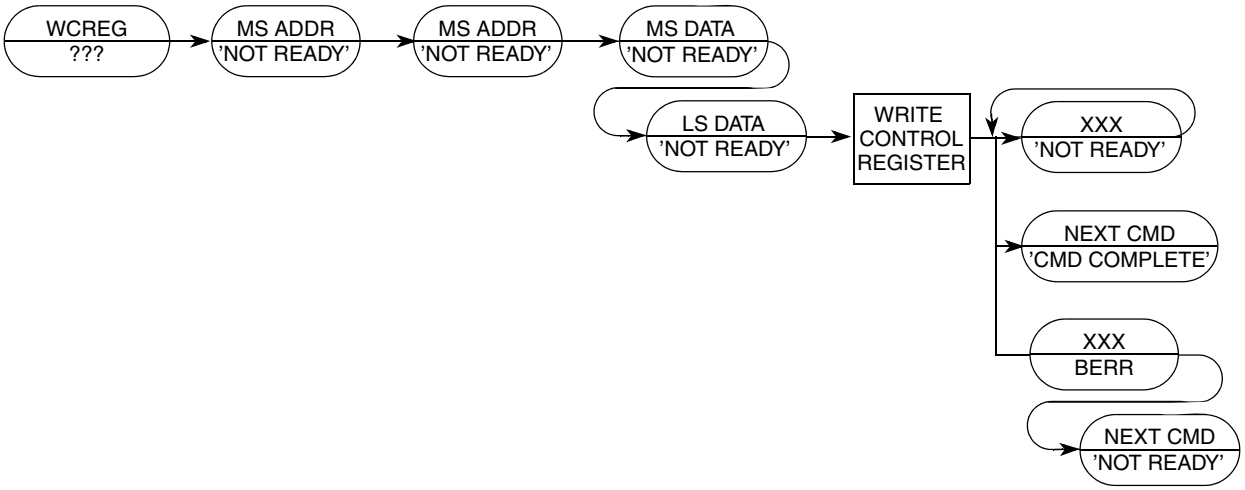


Figure 31-38. WCREG Command Sequence

Operand Data: This instruction requires two longword operands. The first selects the register to the operand data writes to; the second contains the data.

Result Data: Successful write operations return 0xFFFF. Bus errors on the write cycle are indicated by the setting of bit 16 in the status message and by a data pattern of 0x0001.

31.4.1.5.14 Read Debug Module Register (RDMREG)

Read the selected debug module register and return the 32-bit result. The only valid register selection for the RDMREG command is CSR (DRc=0x00).

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0xD				1	0	DRc					
Result	D[31:16]															
	D[15:0]															

Figure 31-39. RDMREG Command/Result Formats

Table 31-22 shows the definition of DRc encoding.

Table 31-22. Definition of DRc Encoding—Read

DRc[5:0]	Debug Register Definition	Mnemonic
0x00	Configuration/Status	CSR

Command Sequence:

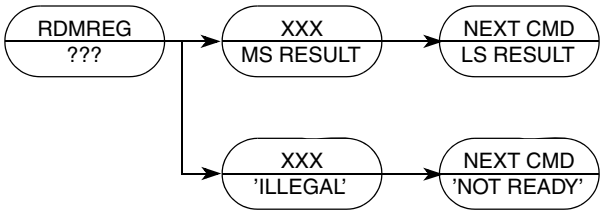


Figure 31-40. RDMREG Command Sequence

Operand Data: None

Result Data: The contents of the selected debug register are returned as a longword value. The data is returned most-significant word first.

31.4.1.5.15 Write Debug Module Register (WDMREG)

The operand (longword) data is written to the specified debug module register. All 32 bits of the register are altered by the write. DSCLK must be inactive while the debug module register writes from the CPU accesses are performed using the WDEBUG instruction.

Command Format:

Figure 31-41. WDMREG BDM Command Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0xC				1	0	DRc					
D[31:16]															
D[15:0]															

Table 31-3 shows the definition of the DRc write encoding.

Command Sequence:

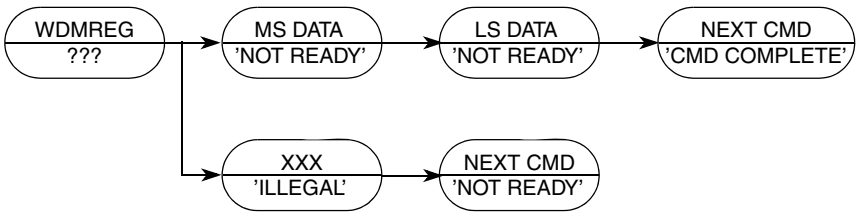


Figure 31-42. WDMREG Command Sequence

Operand Data: Longword data is written into the specified debug register. The data is supplied most-significant word first.

Result Data: Command complete status (0xFFFF) is returned when register write is complete.

31.4.2 Real-Time Debug Support

The ColdFire family provides support debugging real-time applications. For these types of embedded systems, the processor must continue to operate during debug. The foundation of this area of debug support is that while the processor cannot be halted to allow debugging, the system can generally tolerate the small intrusions of the BDM inserting instructions into the pipeline with minimal effect on real-time operation.

The debug module provides four types of breakpoints: PC with mask, PC without mask, operand address range, and data with mask. These breakpoints can be configured into one- or two-level triggers with the exact trigger response also programmable. The debug module programming model can be written from the external development system using the debug serial interface or from the processor's supervisor programming model using the WDEBUG instruction. Only CSR is readable using the external development system.

31.4.2.1 Theory of Operation

Breakpoint hardware can be configured through TDR[TCR] to respond to triggers by displaying DDATA, initiating a processor halt, or generating a debug interrupt. As shown in Table 31-23, when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the DDATA output port when it is not displaying captured processor status, operands, or branch addresses.

Table 31-23. DDATA[3:0]/CSR[BSTAT] Breakpoint Response

DDATA[3:0][1]	CSR[BSTAT][1]	Breakpoint Status
0000	0000	No breakpoints enabled
0010	0001	Waiting for level-1 breakpoint
0100	0010	Level-1 breakpoint triggered
1010	0101	Waiting for level-2 breakpoint
1100	0110	Level-2 breakpoint triggered

1 Encodings not shown are reserved for future use.

The breakpoint status is also posted in the CSR. CSR[BSTAT] is cleared by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and a level-2 breakpoint is not enabled. Status is also cleared by writing to either TDR to disable trigger options.

BDM instructions use the appropriate registers to load and configure breakpoints. As the system operates, a breakpoint trigger generates the response defined in TDR.

PC breakpoints are treated in a precise manner—exception recognition and processing are initiated before the excepting instruction executes. All other breakpoint events are recognized on the processor's local bus, but are made pending to the processor and sampled like other interrupt conditions. As a result, these interrupts are imprecise.

In systems that tolerate the processor being halted, a BDM-entry can be used. With TDR[TRC] equals 01, a breakpoint trigger causes the core to halt (PST = 0xF).

If the processor core cannot be halted, the debug interrupt can be used. With this configuration, TDR[TRC] equals 10, breakpoint trigger becomes a debug interrupt to the processor, which is treated

higher than the nonmaskable level-7 interrupt request. As with all interrupts, it is made pending until the processor reaches a sample point, which occurs once per instruction. Again, the hardware forces the PC breakpoint to occur before the targeted instruction executes and is precise. This is possible because the PC breakpoint is enabled when interrupt sampling occurs. For address and data breakpoints, reporting is considered imprecise, because several instructions may execute after the triggering address or data is detected.

As soon as the debug interrupt is recognized, the processor aborts execution and initiates exception processing. This event is signaled externally by the assertion of a unique PST value (PST = 0xD) for multiple cycles. The core enters emulator mode when exception processing begins. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector, 12, from the vector table.Refer to the *ColdFire Programmer's Reference Manual*. for more information.

Execution continues at the instruction address in the vector corresponding to the debug interrupt. All interrupts are ignored while the processor is in emulator mode. The debug interrupt handler can use supervisor instructions to save the necessary context, such as the state of all program-visible registers into a reserved memory area.

When debug interrupt operations complete, the RTE instruction executes and the processor exits emulator mode. After the debug interrupt handler completes execution, the external development system can use BDM commands to read the reserved memory locations.

In revision B/B+, the hardware inhibits generation of another debug interrupt during the first instruction after the RTE exits emulator mode. This behavior is consistent with the logic involving trace mode where the first instruction executes before another trace exception is generated. Thus, all hardware breakpoints are disabled until the first instruction after the RTE completes execution, regardless of the programmed trigger response.

31.4.2.2 Emulator Mode

Emulator mode facilitates non-intrusive emulator functionality. This mode can be entered in three different ways:

- Setting CSR[EMU] forces the processor into emulator mode. EMU is examined only if $\overline{\text{RSTI}}$ is negated and the processor begins reset exception processing. It can be set while the processor is halted before reset exception processing begins. See Section 31.4.1.1, "CPU Halt".
- A debug interrupt always puts the processor in emulation mode when debug interrupt exception processing begins.
- Setting CSR[TRC] forces the processor into emulation mode when trace exception processing begins.

While operating in emulation mode, the processor exhibits the following properties:

- All interrupts are ignored, including level-7 interrupts.
- If CSR[MAP] is set, all caching of memory and the SRAM module are disabled. All memory accesses are forced into a specially mapped address space signaled by TT equals 0x2, TM equals 0x5, or 0x6. This includes stack frame writes and vector fetch for the exception that forced entry into this mode.

The RTE instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry (0xD) and exit (0x7).

31.4.3 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of the processor and most BDM commands. BDM commands may be executed while the processor is running, except these following operations that access processor/memory registers:

- Read/write address and data registers
- Read/write control registers

For BDM commands that access memory, the debug module requests the processor's local bus. The processor responds by stalling the instruction fetch pipeline and waiting for current bus activity to complete before freeing the local bus for the debug module to perform its access. After the debug module bus cycle, the processor reclaims the bus.

NOTE

Breakpoint registers must be carefully configured in a development system if the processor is executing. The debug module contains no hardware interlocks, so TDR should be disabled while breakpoint registers are loaded, after which TDR can be written to define the exact trigger. This prevents spurious breakpoint triggers.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers (DSCLK must be inactive).

NOTE

The debug module requires the use of the internal bus to perform BDM commands. For this processor core, if the processor is executing a tight loop contained within a single aligned longword, the processor may never grant the internal bus to the debug module, for example:

```
          align4
label1:   nop
          bra.b label1
or
          align4
label2:   bra.w label2
```

The processor grants the internal bus if these loops are forced across two longwords.

31.4.4 Real-Time Trace Support

Real-time trace, which defines the dynamic execution path and is also known as instruction trace, is a fundamental debug function. The ColdFire solution is to include a parallel output port providing encoded processor status and data to an external development system. This port is partitioned into two 4-bit nibbles: one nibble allows the processor to transmit processor status, (PST), and the other allows operand data to

be displayed (debug data, DDATA). The processor status may not be related to the current bus transfer, due to the decoupling FIFOs.

External development systems can use PST outputs with an external image of the program to completely track the dynamic execution path. This tracking is complicated by any change in flow, where branch target address calculation is based on the contents of a program-visible register (variant addressing). DDATA outputs can display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in Section 31.4.4.1, “Begin Execution of Taken Branch (PST = 0x5)”. Two 32-bit storage elements form a FIFO buffer connecting the processor’s high-speed local bus to the external development system through PST[3:0] and DDATA[3:0]. The buffer captures branch target addresses and certain data values for eventual display on the DDATA port, one nibble at a time starting with the least significant bit (lsb).

Execution speed is affected only when both storage elements contain valid data to be dumped to the DDATA port. The core stalls until one FIFO entry is available.

Table 31-24 shows the encoding of these signals.

Table 31-24. Processor Status Encoding

PST[3:0]	Definition
0x0	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more clock cycles, subsequent clock cycles are indicated by driving PST outputs with this encoding.
0x1	Begin execution of one instruction. For most instructions, this encoding signals the first processor clock cycle of an instruction’s execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings.
0x2	Reserved
0x3	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x4	Begin execution of PULSE and WDDATA instructions. PULSE defines logic analyzer triggers for debug and/or performance analysis. WDDATA lets the core write any operand (byte, word, or longword) directly to the DDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. Transfer length depends on the WDDATA operand size.
0x5	Begin execution of taken branch or SYNC_PC command issued. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See Section 31.4.4.1, “Begin Execution of Taken Branch (PST = 0x5)”. Also indicates that the SYNC_PC command has been issued.
0x6	Reserved
0x7	Begin execution of return from exception (RTE) instruction.
0x8–0xB	Indicates the number of bytes to be displayed on the DDATA port on subsequent clock cycles. The value is driven onto the PST port one PSTCLK cycle before the data is displayed on DDATA. 0x8 Begin 1-byte transfer on DDATA. 0x9 Begin 2-byte transfer on DDATA. 0xA Begin 3-byte transfer on DDATA. 0xB Begin 4-byte transfer on DDATA.

Table 31-24. Processor Status Encoding (continued)

PST[3:0]	Definition
0xC	Normal exception processing. Exceptions that enter emulation mode (debug interrupt or optionally trace) generate a different encoding, as described below. Because the 0xC encoding defines a multiple-cycle mode, PST outputs are driven with 0xC until exception processing completes.
0xD	Emulator mode exception processing. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PST outputs are driven with 0xD until exception processing completes.
0xE	Processor is stopped. Appears in multiple-cycle format when the processor executes a STOP instruction. The ColdFire processor remains stopped until an interrupt occurs, thus PST outputs display 0xE until the stopped mode is exited.
0xF	Processor is halted. Because this encoding defines a multiple-cycle mode, the PST outputs display 0xF until the processor is restarted or reset. See Section 31.4.1.1, “CPU Halt”.

31.4.4.1 Begin Execution of Taken Branch (PST = 0x5)

PST is 0x5 when a taken branch is executed. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, which is indicated by the PST marker value immediately preceding the DDATA nibble that begins the data output.

Multiple byte DDATA values are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches that use a variant addressing mode (RTE and RTS instructions, JMP and JSR instructions using address register indirect or indexed addressing modes, and all exception vectors).

The simplest example of a branch instruction using a variant address is the compiled code for a C language case statement. Typically, the evaluation of this statement uses the variable of an expression as an index into a table of offsets, where each offset points to a unique case within the structure. For such change-of-flow operations, the ColdFire processor uses the debug pins to output the following sequence of information on two successive processor clock cycles:

1. Use PST (0x5) to identify that a taken branch is executed.
2. Signal the target address to be displayed sequentially on the DDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.Using the PSTB, o
3. The new target address is optionally available on subsequent cycles using the DDATA port. The number of bytes of displayed on this port is configurable (2, 3, or 4 bytes, where the DDATA encoding is 0x9, 0xA, and 0xB, respectively).

Another example of a variant branch instruction would be a JMP (A0) instruction. Figure 31-43 shows the PST and DDATA outputs that indicate a JMP (A0) execution, assuming the CSR was programmed to display the lower 2 bytes of an address.

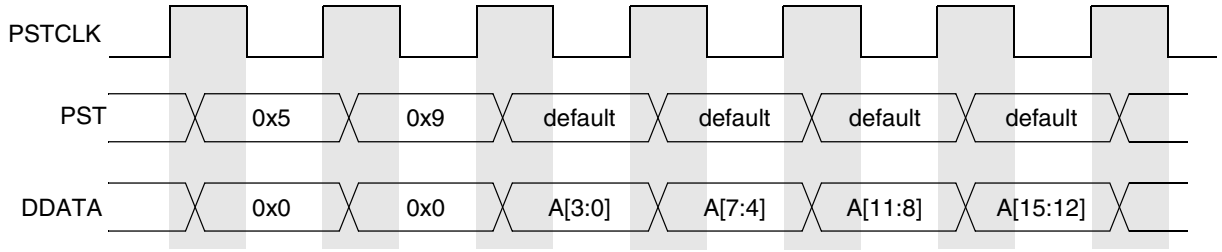


Figure 31-43. Example JMP Instruction Output on PST/DDATA

PST of 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Therefore, the subsequent 4 nibbles of DDATA display the lower two bytes of address register A0 in least-to-most-significant nibble order. The PST output after the JMP instruction completes depends on the target instruction. The PST can continue with the next instruction before the address has completely displayed on DDATA because of the DDATA FIFO. If the FIFO is full and the next instruction has captured values to display on DDATA, the pipeline stalls (PST = 0x0) until space is available in the FIFO.

31.4.5 Processor Status, Debug Data Definition

This section specifies the ColdFire processor and debug module's generation of the processor status (PST) and debug data (DDATA) output on an instruction basis. In general, the PST/DDATA output for an instruction is defined as follows:

PST = 0x1, {PST = [0x89B], DDATA = operand}

where the {...} definition is optional operand information defined by the setting of the CSR.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x8, 0x9, or 0xB} identifies the size and presence of valid data to follow on the DDATA output {1, 2, or 4 bytes}. Additionally, for certain change-of-flow branch instructions, CSR[BTB] provides the capability to display the target instruction address on the DDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}. Addresses use the markers x0D, x0E, or 0xF to store 2, 3, or 4 bytes of address packets with address shifted right by 1 bit.

31.4.5.1 User Instruction Set

Table 31-25 shows the PST/DDATA specification for user-mode instructions. Rn represents any {Dn, An} register. In this definition, the y suffix generally denotes the source, and x denotes the destination operand. For a given instruction, the optional operand data is displayed only for those effective addresses referencing memory. The DD nomenclature refers to the DDATA outputs.

Table 31-25. PST/DDATA Specification for User-Mode Instructions

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
add.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
adda.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}

Table 31-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
addi.l	#<data>,Dx	PST = 0x1
addq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addx.l	Dy,Dx	PST = 0x1
and.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
and.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
andi.l	#<data>,Dx	PST = 0x1
asl.l	{Dy,#<data>},Dx	PST = 0x1
asr.l	{Dy,#<data>},Dx	PST = 0x1
bcc.{b,w}		if taken, then PST = 0x5, else PST = 0x1
bchg.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bchg.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bitrev.l	Dx	PST = 0x1
bra.{b,w}		PST = 0x5
bset.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bset.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bsr.{b,w}		PST = 0x5, {PST = 0xB, DD = destination operand}
btst.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
btst.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
byterev.l	Dx	PST = 0x1
clr.b	<ea>x	PST = 0x1, {PST = 0x8, DD = destination operand}
clr.l	<ea>x	PST = 0x1, {PST = 0xB, DD = destination operand}
clr.w	<ea>x	PST = 0x1, {PST = 0x9, DD = destination operand}
cmp.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
cmpa.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}
cmpi.l	#<data>,Dx	PST = 0x1
divs.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divs.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
divu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
eor.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
eori.l	#<data>,Dx	PST = 0x1

Table 31-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
ext.l	Dx	PST = 0x1
ext.w	Dx	PST = 0x1
extb.l	Dx	PST = 0x1
illegal		PST = 0x1[1]
jmp	<ea>y	PST = 0x5, {PST = [0x9AB], DD = target address}[2]
jsr	<ea>y	PST = 0x5, {PST = [0x9AB], DD = target address}, {PST = 0xB, DD = destination operand}[2]
lea.l	<ea>y,Ax	PST = 0x1
link.w	Ay,#<displacement>	PST = 0x1, {PST = 0xB, DD = destination operand}
lsl.l	{Dy,#<data>},Dx	PST = 0x1
lsr.l	{Dy,#<data>},Dx	PST = 0x1
move.b	<ea>y,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x1, {PST = 0x9, DD = source}, {PST = 0x9, DD = destination}
move.w	CCR,Dx	PST = 0x1
move.w	{Dy,#<data>},CCR	PST = 0x1
movea.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source}
movea.w	<ea>y,Ax	PST = 0x1, {PST = 0x9, DD = source}
movem.l	#list,<ea>x	PST = 0x1, {PST = 0xB, DD = destination},... [3]
movem.l	<ea>y,#list	PST = 0x1, {PST = 0xB, DD = source},... [3]
moveq.l	#<data>,Dx	PST = 0x1
muls.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
neg.l	Dx	PST = 0x1
negx.l	Dx	PST = 0x1
nop		PST = 0x1
not.l	Dx	PST = 0x1
or.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
or.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
ori.l	#<data>,Dx	PST = 0x1
pea.l	<ea>y	PST = 0x1, {PST = 0xB, DD = destination operand}

Table 31-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
pulse		PST = 0x4
rems.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = source operand}
remu.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = source operand}
rts		PST = 0x1, {PST = 0xB, DD = source operand}, PST = 0x5, {PST = 0x[9AB], DD = target address}
rts (not predicted)		PSTDDATA = 0x1, {0xB, source operand}, 0x5, {0x[9AB], target address}
rts (predicted)[4]		PSTDDATA = 0x1, {0xB, source operand}, 0x5
scc.b	Dx	PST = 0x1
sub.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
suba.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}
subi.l	#<data>,Dx	PST = 0x1
subq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subx.l	Dy,Dx	PST = 0x1
swap.w	Dx	PST = 0x1
tpf		PST = 0x1
tpf.l	#<data>	PST = 0x1
tpf.w	#<data>	PST = 0x1
trap	#<data>	PST = 0x1[1]
tst.b	<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
tst.l	<ea>y	PST = 0x1, {PST = 0xB, DD = source operand}
tst.w	<ea>y	PST = 0x1, {PST = 0x9, DD = source operand}
unlk	Ax	PST = 0x1, {PST = 0xB, DD = destination operand}
wddata.b	<ea>y	PST = 0x4, {PST = 0x8, DD = source operand}
wddata.l	<ea>y	PST = 0x4, {PST = 0xB, DD = source operand}
wddata.w	<ea>y	PST = 0x4, {PST = 0x9, DD = source operand}

[1] During normal exception processing, the PST output is driven to a 0xC indicating the exception processing state. The exception stack write operands, as well as the vector read and target address of the exception handler may also be displayed.

```
Exception Processing:
     PST = 0xC,
     {PST = 0xB,DD = destination},        // stack frame
     {PST = 0xB,DD = destination},        // stack frame
     {PST = 0xB,DD = source},             // vector read
     PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The PST/DDATA specification for the reset exception is shown below:

```
Exception Processing:
     PST = 0xC,
     PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The initial references at address 0 and 4 are never captured nor displayed because these accesses are treated as instruction fetches.

For all types of exception processing, the PST = 0xC value is driven at all times, unless the PST output is needed for one of the optional marker values or for the taken branch indicator (0x5).

[2] For JMP and JSR instructions, the optional target instruction address is displayed only for those effective address fields defining variant addressing modes. This includes the following <ea>x values: (An), (d16,An), (d8,An,Xi), (d8,PC,Xi).

[3] For move multiple instructions (MOVEM), the processor automatically generates line-sized transfers if the operand address reaches a 0-modulo-16 boundary and there are four or more registers to be transferred. For these line-sized transfers, the operand data is never captured nor displayed, regardless of the CSR value.
The automatic line-sized burst transfers are provided to maximize performance during these sequential memory access operations.

[4] For a predicted RTS instruction, the source operand is displayed if CSR[12], CSR[9], or CSR[8] is set.

Table 31-26 shows the PST/DDATA specification for multiply-accumulate instructions.

Table 31-26. PST/DDATA Values for User-Mode Multiply-Accumulate Instructions

Instruction	Operand Syntax	PST/DDATA
mac.l	Ry,Rx,ACCx	PST = 0x1
mac.l	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}
mac.w	Ry,Rx,ACCx	PST = 0x1
mac.w	Ry,Rx,ea,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}
move.l	{Ry,#<data>},ACC*x*	PST = 0x1
move.l	{Ry,#<data>},MACSR	PST = 0x1
move.l	{Ry,#<data>},MASK	PST = 0x1
move.l	{Ry,#<data>},ACCext01	PST = 0x1
move.l	{Ry,#<data>},ACCext23	PST = 0x1
move.l	ACCext01,Rx	PST = 0x1
move.l	ACCext23,Rx	PST = 0x1
move.l	ACCy,ACCx	PST = 0x1
move.l	ACCy,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1

Table 31-26. PST/DDATA Values for User-Mode Multiply-Accumulate Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1
msac.l	Ry,Rx,ACCx	PST = 0x1
msac.l	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}
msac.w	Ry,Rx,ACCx	PST = 0x1
msac.w	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}

31.4.5.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PST/DDATA specification for these opcodes is shown in Table 31-27.

Table 31-27. PST/DDATA Specification for Supervisor-Mode Instructions

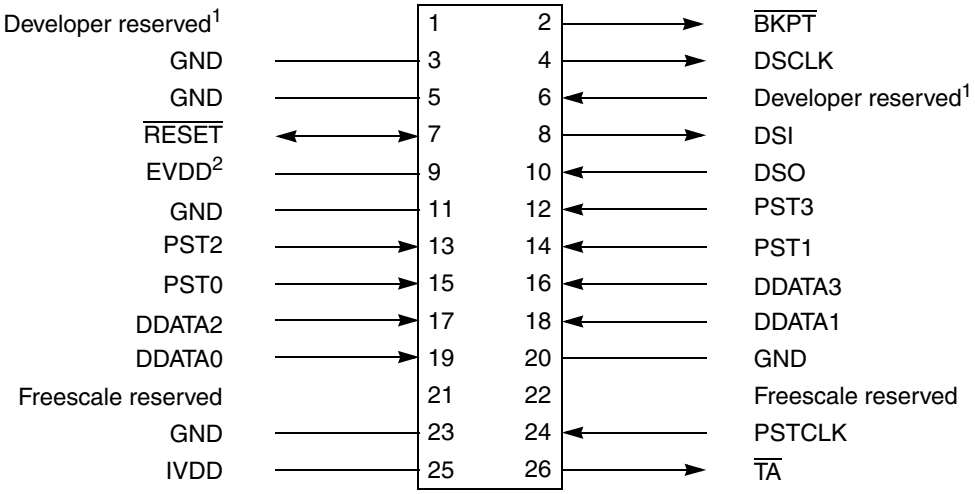
Instruction	Operand Syntax	PST/DDATA
cpushl	(Ax)	PST = 0x1
halt		PST = 0x1, PST = 0xF
move.l	Ay,USP	PST = 0x1
move.l	USP,Ax	PST = 0x1
move.w	SR,Dx	PST = 0x1
move.w	{Dy,#<data>},SR	PST = 0x1, {PST = 0x3}
movec.l	Ry,Rc	PST = 0x1
rte		PST = 0x7, {PST = 0xB, DD = source operand}, {PST = 0x3},{ PST = 0xB, DD =source operand}, PST = 0x5, {[PST = 0x9AB], DD = target address}
stldsr.w	#imm	PST = 0x1, {PST = 0xA, DD = destination operand, PST = 0x3}
stop	#<data>	PST = 0x1, PST = 0xE
wdebug.l	<ea>y	PST = 0x1, {PST = 0xB, DD = source, PST = 0xB, DD = source}

The move-to-SR and RTE instructions include an optional PST = 0x3 value, indicating an entry into user mode. Additionally, if the execution of a RTE instruction returns the processor to emulator mode, a multiple-cycle status of 0xD is signaled.

Similar to the exception processing mode, the stopped state (PST = 0xE) and the halted state (PST = 0xFF) display this status throughout the entire time the ColdFire processor is in the given mode.

31.4.6 Freescale-Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg connector arranged 2 x 13 as shown below.



[1] Pins reserved for BDM developer use.
[2] Supplied by target

Figure 31-44. Recommended BDM Connector

Chapter 32
IEEE 1149.1 Test Access Port (JTAG)

32.1 Introduction

The Joint Test Action Group (JTAG) is a dedicated user-accessible test logic compliant with the IEEE 1149.1 standard for boundary-scan testability, which helps with system diagnostic and manufacturing testing.

This architecture provides access to all data and chip control pins from the board-edge connector through the standard four-pin test access port (TAP) and the JTAG reset pin, $\overline{\text{TRST}}$.

32.1.1 Block Diagram

Figure 32-1 shows the block diagram of the JTAG module.

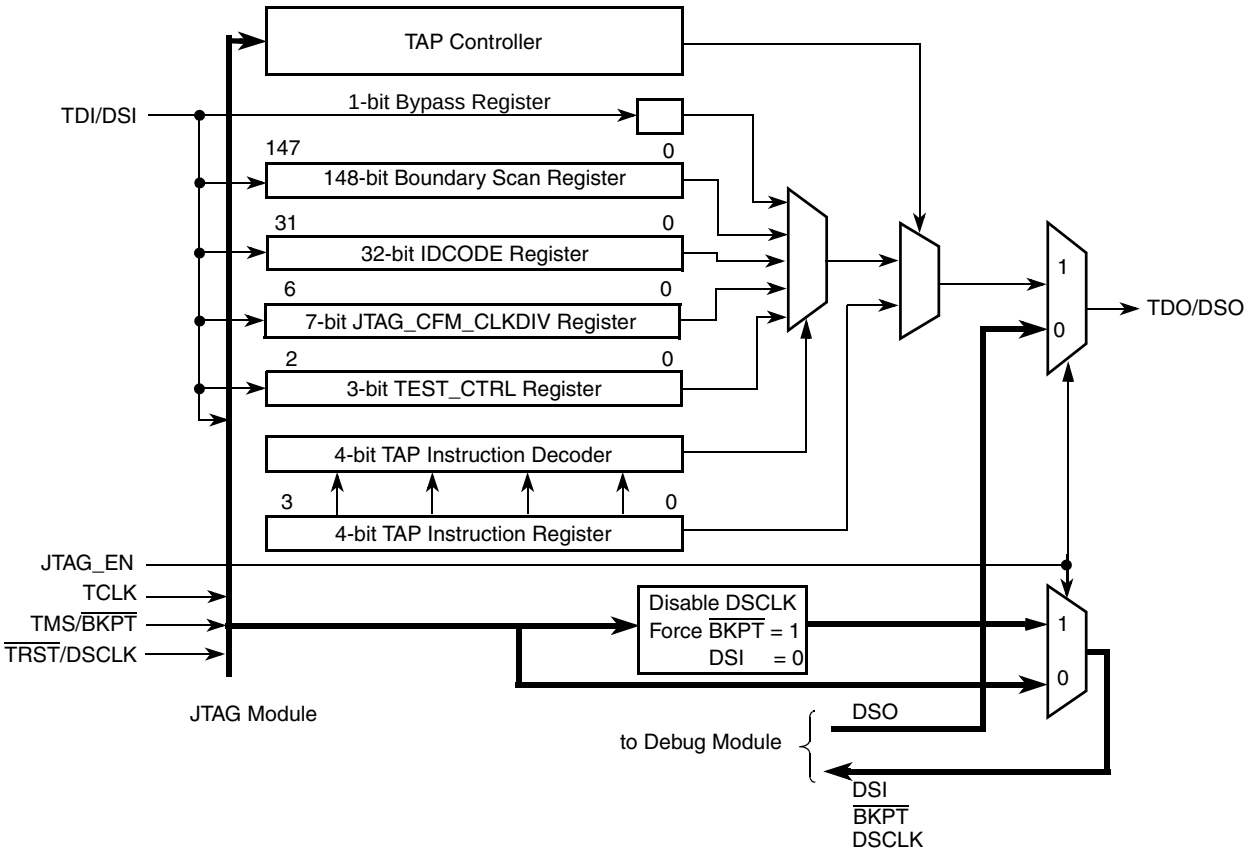


Figure 32-1. JTAG Block Diagram

32.1.2 Features

The basic features of the JTAG module are the following:

- Performs boundary-scan operations to test circuit board electrical continuity
- Bypasses instruction to reduce the shift register path to a single cell
- Sets chip output pins to safety states while executing the bypass instruction
- Samples the system pins during operation and transparently shifts out the result
- Selects between JTAG TAP controller and Background Debug Module (BDM) using a dedicated JTAG_EN pin

32.1.3 Modes of Operation

The JTAG_EN pin can select between the following modes of operation:

- JTAG mode (JTAG_EN = 1)
- Background debug mode (BDM)—for more information, refer to Section 31.4.1, “Background Debug Mode (BDM)”; (JTAG_EN = 0).

32.2 External Signal Description

The JTAG module has five input and one output external signals, as described in Table 32-1.

Table 32-1. Signal Properties

Name	Direction	Function	Reset State	Pull up
JTAG_EN	Input	JTAG/BDM selector input	—	—
TCLK	Input	JTAG Test clock input	—	Active
TMS/$\overline{\text{BKPT}}$	Input	JTAG Test mode select / BDM Breakpoint	—	Active
TDI/DSI	Input	JTAG Test data input / BDM Development serial input	—	Active
$\overline{\text{TRST}}$/DSCLK	Input	JTAG Test reset input / BDM Development serial clock	—	Active
TDO/DSO	Output	JTAG Test data output / BDM Development serial output	Hi-Z / 0	—

32.2.1 JTAG Enable (JTAG_EN)

The JTAG_EN pin selects between the debug module and JTAG. If JTAG_EN is low, the debug module is selected; if it is high, the JTAG is selected. Table 32-2 summarizes the pin function selected depending on JTAG_EN logic state.

Table 32-2. Pin Function Selected

	JTAG_EN = 0	JTAG_EN = 1	Pin Name
Module selected	BDM	JTAG	—
Pin Function	— $\overline{\text{BKPT}}$ DSI DSO DSCLK	TCLK TMS TDI TDO $\overline{\text{TRST}}$	TCLK $\overline{\text{BKPT}}$ DSI DSO DSCLK

When one module is selected, the inputs into the other module are disabled or forced to a known logic level, as shown in Table 32-3, to disable the corresponding module.

Table 32-3. Signal State to the Disable Module

	JTAG_EN = 0	JTAG_EN = 1
Disabling JTAG	$\overline{\text{TRST}}$ = 0 TMS = 1	—
Disabling BDM	—	Disable DSCLK DSI = 0 $\overline{\text{BKPT}}$ = 1

NOTE

The JTAG_EN does not support dynamic switching between JTAG and BDM modes.

32.2.2 Test Clock Input (TCLK)

The TCLK pin is a dedicated JTAG clock input to synchronize the test logic. Pulses on TCLK shift data and instructions into the TDI pin on the rising edge and out of the TDO pin on the falling edge. TCLK is independent of the processor clock. The TCLK pin has an internal pull-up resistor, and holding TCLK high or low for an indefinite period does not cause JTAG test logic to lose state information.

32.2.3 Test Mode Select/Breakpoint (TMS/$\overline{\text{BKPT}}$)

The TMS pin is the test mode select input that sequences the TAP state machine. TMS is sampled on the rising edge of TCLK. The TMS pin has an internal pull-up resistor.

The $\overline{\text{BKPT}}$ pin is used to request an external breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes.

32.2.4 Test Data Input/Development Serial Input (TDI/DSI)

The TDI pin receives serial test and data, which is sampled on the rising edge of TCLK. Register values are shifted in least significant bit (lsb) first. The TDI pin has an internal pull-up resistor.

The DSI pin provides data input for the debug module serial communication port.

32.2.5 Test Reset/Development Serial Clock ($\overline{\text{TRST}}$/DSCLK)

The $\overline{\text{TRST}}$ pin is an active low asynchronous reset input with an internal pull-up resistor that forces the TAP controller to the test-logic-reset state.

The DSCLK pin clocks the serial communication port to the debug module. Maximum frequency is 1/5 the processor clock speed. At the rising edge of DSCLK, data input on DSI is sampled and DSO changes state.

32.2.6 Test Data Output/Development Serial Output (TDO/DSO)

The TDO pin is the lsb-first data output. Data is clocked out of TDO on the falling edge of TCLK. TDO is tri-stateable and actively driven in the shift-IR and shift-DR controller states.

The DSO pin provides serial output data in BDM mode.

32.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin. All registers described below are shift-in and parallel load.

32.3.1 Instruction Shift Register (IR)

The JTAG module uses a -bit shift register with no parity. The IR transfers its value to a parallel hold register and applies an instruction on the falling edge of TCLK when the TAP state machine is in the update-IR state. To load an instruction into the shift portion of the IR, place the serial data on the TDI pin before each rising edge of TCLK. The msb of the IR is the bit closest to the TDI pin, and the lsb is the bit closest to the TDO pin. See Section 32.4.3, “JTAG Instructions” for a list of possible instruction codes.

32.3.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see Section 32.4.3.1, “IDCODE Instruction”.

IR[4:0]: 0_0001 (IDCODE) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PRN				DC						PIN										JEDEC											ID
W																																
Reset	See note[1]				See note[2]								See note[1]								0	0	0	0	0	0	0	1	1	1	0	1

[1] The reset values for PRN and PIN are device-dependent.

[2] Varies, depending on design center location.

Figure 32-2. IDCODE Register

Table 32-4. IDCODE Field Descriptions

Field	Description
31–28 PRN	Part revision number. Indicate the revision number of the device.
27–22 DC	Freescale design center number.
21–12 PIN	Part identification number. Indicate the device number. 0x048 MCF52230 0x049 MCF52231 0x04A MCF52233 0x04B MCF52234 0x04C MCF52235 0x04D MCF52236
11–1 JEDEC	Joint Electron Device Engineering Council ID bits. Indicate the reduced JEDEC ID for Freescale (0x0E).
0 ID	IDCODE register ID. This bit is set to 1 to identify the register as the IDCODE register and not the bypass register according to the IEEE standard 1149.1.

32.3.3 Bypass Register

The bypass register is a single-bit shift register path from TDI to TDO when the BYPASS instruction is selected.

32.3.4 JTAG_CFM_CLKDIV Register

The JTAG_CFM_CLKDIV register is a 7-bit clock divider for the CFM that is used with the LOCKOUT_RECOVERY instruction. It controls the period of the clock used for timed events in the CFM erase algorithm. The JTAG_CFM_CLKDIV register must be loaded before the lockout sequence can begin.

32.3.5 TEST_CTRL Register

The TEST_CTRL register is a 3-bit shift register path from TDI to TDO when the ENABLE_TEST_CTRL instruction is selected. The TEST_CTRL transfers its value to a parallel hold register on the rising edge of TCLK when the TAP state machine is in the update-DR state.

32.3.6 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instruction is selected. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins or high impedance for tri-stated pins.

The boundary scan register contains bits for bonded-out and non bonded-out signals, excluding JTAG signals, analog signals, power supplies, compliance enable pins, and clock signals.

32.4 Functional Description

32.4.1 JTAG Module

The JTAG module consists of a TAP controller state machine, which is responsible for generating all control signals that execute the JTAG instructions and read/write data registers.

32.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. Figure 32-3 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 TCLK signal.

Asserting the $\overline{\text{TRST}}$ signal asynchronously resets the TAP controller to the test-logic-reset state. As Figure 32-3 shows, holding TMS at logic 1 while clocking TCLK through at least five rising edges also causes the state machine to enter the test-logic-reset state, whatever the initial state.

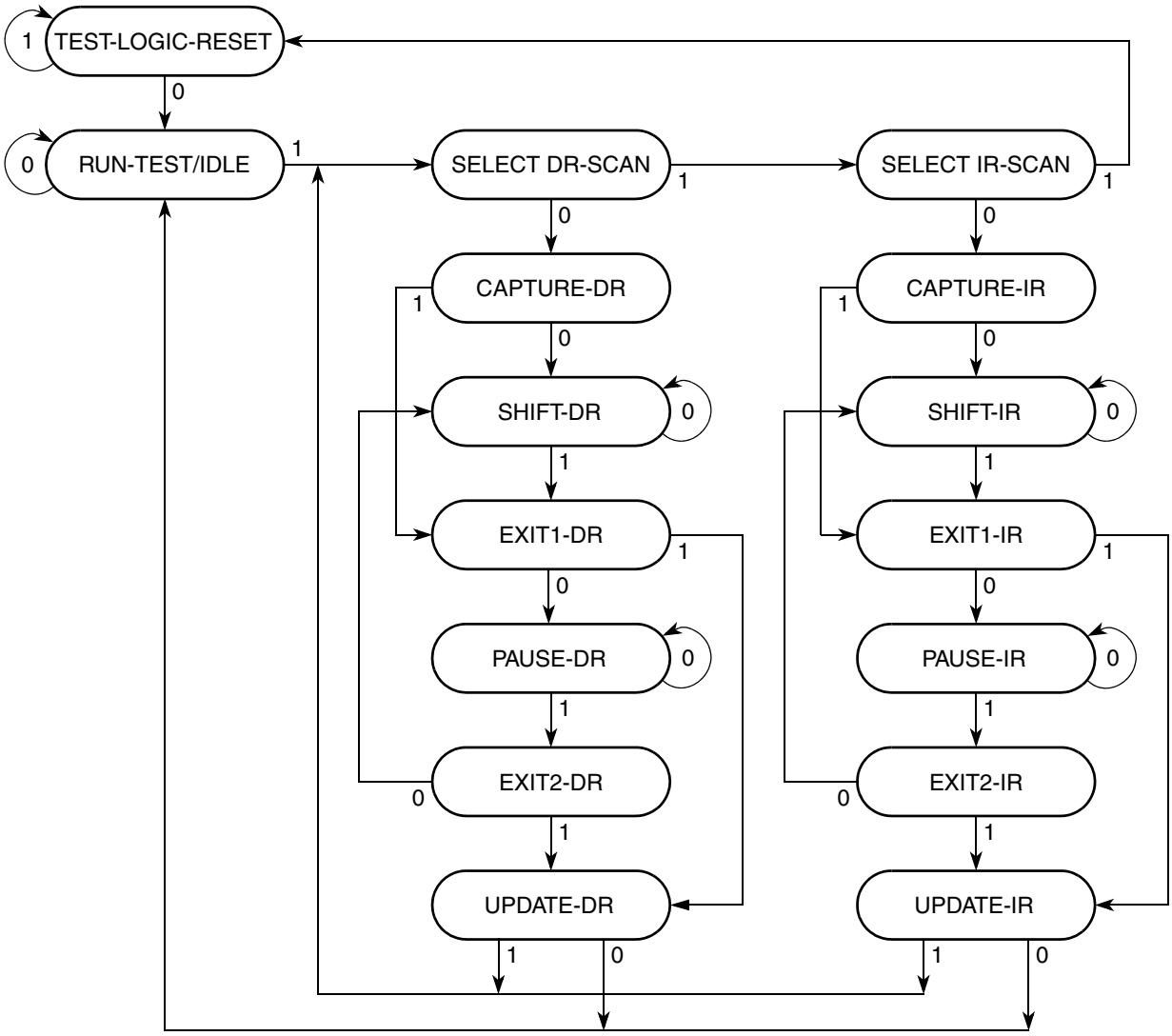


Figure 32-3. TAP Controller State Machine Flow

32.4.3 JTAG Instructions

Table 32-5 describes public and private instructions.

Table 32-5. JTAG Instructions

Instruction	IR[3:0]	Instruction Summary
EXTEST	0000	Selects boundary scan register while applying fixed values to output pins and asserting functional reset
IDCODE	0001	Selects IDCODE register for shift
SAMPLE/PRELOAD	0010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation

Table 32-5. JTAG Instructions (continued)

Instruction	IR[3:0]	Instruction Summary
TEST_LEAKAGE	0101	Selects bypass register while tri-stating all output pins and assert to high the jtag_leakage signal
ENABLE_TEST_CTRL	0110	Selects TEST_CTRL register
HIGHZ	1001	Selects bypass register while tri-stating all output pins and asserting functional reset
LOCKOUT_RECOVERY	1011	Allows for the erase of the TFM flash when the part is secure
CLAMP	1100	Selects bypass while applying fixed values to output pins and asserting functional reset
BYPASS	1111	Selects bypass register for data operations
Reserved	all others[1]	Decoded to select bypass register

1 Freescale reserves the right to change the decoding of the unused opcodes in the future.

32.4.3.1 IDCODE Instruction

The IDCODE instruction selects the 32-bit IDCODE register for connection as a shift path between the TDI and TDO pin. This instruction allows interrogation of the MCU to determine its version number and other part identification data. The shift register lsb is forced to logic 1 on the rising edge of TCLK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are also forced to fixed values on the rising edge of TCLK following entry into the capture-DR state.

IDCODE is the default instruction placed into the instruction register when the TAP resets. Thus, after a TAP reset, the IDCODE register is selected automatically.

32.4.3.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- SAMPLE - obtain a sample of the system data and control signals present at the MCU input pins and before the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCLK in the capture-DR state when the IR contains the $2 opcode. The sampled data is accessible by shifting it through the boundary scan register to the TDO output by using the shift-DR state. The data capture and the shift operation are transparent to system operation.

NOTE

External synchronization is required to achieve meaningful results because there is no internal synchronization between TCLK and the system clock.

- PRELOAD - initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data shifting out on the TDO pin and shifting in initialization data. The update-DR state and the falling edge of TCLK can then transfer this data to the update cells. The data is applied to the external output pins by the EXTEST or CLAMP instruction.

32.4.3.3 EXTEST Instruction

The external test (EXTEST) instruction selects the boundary scan register. It forces all output pins and bidirectional pins configured as outputs to the values preloaded with the SAMPLE/PRELOAD instruction and held in the boundary scan update registers. EXTEST can also configure the direction of bidirectional pins and establish high-impedance states on some pins. EXTEST asserts internal reset for the MCU system logic to force a predictable internal state while performing external boundary scan operations.

32.4.3.4 TEST_LEAKAGE Instruction

The TEST_LEAKAGE instruction forces the jtag_leakage output signal to high. It is intended to tri-state all output pad buffers and disable all of the part's pad input buffers except TEST and TRST. The jtag_leakage signal is asserted at the rising edge of TCLK when the TAP controller transitions from update-IR to run-test/idle state. After asserted, the part disables the TCLK, TMS, and TDI inputs into JTAG and forces these JTAG inputs to logic 1. The TAP controller remains in the run-test/idle state until the TRST input is asserted (logic 0).

32.4.3.5 ENABLE_TEST_CTRL Instruction

The ENABLE_TEST_CTRL instruction selects a 3-bit shift register (TEST_CTRL) for connection as a shift path between the TDI and TDO pin. When the user transitions the TAP controller to the UPDATE_DR state, the register transfers its value to a parallel hold register. It allows the control chip to test functions independent of the JTAG TAP controller state.

32.4.3.6 HIGHZ Instruction

The HIGHZ instruction eliminates the need to backdrive the output pins during circuit-board testing. HIGHZ turns off all output drivers, including the 2-state drivers, and selects the bypass register. HIGHZ also asserts internal reset for the MCU system logic to force a predictable internal state.

32.4.3.7 LOCKOUT_RECOVERY Instruction

If a user inadvertently enables security on a MCU, the LOCKOUT_RECOVERY instruction allows the disabling of security by the complete erasure of the internal flash contents including the configuration field. This does not compromise security as the entire contents of the user's secured code stored in flash gets erased before security is disabled on the MCU on the next reset or power-up sequence.

The LOCKOUT_RECOVERY instruction selects a 7-bit shift register for connection as a shift path between the TDI pin and the TDO pin. When the user transitions the TAP controller to the UPDATE-DR state, the 7-bit shift register is loaded into the 7-bit JTAG_TFM_CLKDIV register and this value is output to the TFM's clock divider circuit. When the user transitions the TAP controller to the RUN-TEST/IDLE state, the erase signal to the TFM asserts and the lockout sequence starts. The controller must remain in that state until the erase sequence has completed. After the lockout recovery sequence has completed, the user must reset the JTAG TAP controller and the MCU to return to normal operation.

32.4.3.8 CLAMP Instruction

The CLAMP instruction selects the bypass register and asserts internal reset while simultaneously forcing all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded and held in the boundary scan update register. 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.

32.4.3.9 BYPASS Instruction

The BYPASS instruction selects the bypass register, creating a single-bit shift register path from the TDI pin to the TDO pin. BYPASS enhances test efficiency by reducing the overall shift path when a device other than the ColdFire processor is the device under test on a board design with multiple chips on the overall boundary scan chain. The shift register lsb is forced to logic 0 on the rising edge of TCLK after entry into the capture-DR state. Therefore, the first bit shifted out after selecting the bypass register is always logic 0. This differentiates parts that support an IDCODE register from parts that support only the bypass register.

32.5 Initialization/Application Information

32.5.1 Restrictions

The test logic is a static logic design, and TCLK can be stopped in a high or low state without loss of data. However, the system clock is not synchronized to TCLK internally. Any mixed operation using the test logic and system functional logic requires external synchronization.

Using the EXTEST instruction requires a circuit-board test environment that avoids device-destructive configurations in which MCU output drivers are enabled into actively driven networks.

Low-power stop mode considerations:

- The TAP controller must be in the test-logic-reset state to enter or remain in the low-power stop mode. Leaving the test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
- The TCLK input is not blocked in low-power stop mode. To consume minimal power, the TCLK input should be externally connected to V_{DD}.
- The TMS, TDI, and $\overline{\text{TRST}}$ pins include on-chip pull-up resistors. For minimal power consumption in low-power stop mode, these three pins should be connected to V_{DD} or left unconnected.

32.5.2 Nonscan Chain Operation

Keeping the TAP controller in the test-logic-reset state ensures that the scan chain test logic is transparent to the system logic. It is recommended that TMS, TDI, TCLK, and $\overline{\text{TRST}}$ be pulled up. $\overline{\text{TRST}}$ could be connected to ground. However, because there is a pull-up on $\overline{\text{TRST}}$, some amount of current results. The internal power-on reset input initializes the TAP controller to the test-logic-reset state on power-up without asserting $\overline{\text{TRST}}$.

Appendix A
Register Memory Map Quick Reference

Table A-1 summarizes the address, name, and byte assignment for registers within the MCF52235 CPU space. Table A-2 lists an overview of the memory map for the on-chip modules, and Table A-3 is a detailed memory map including all of the registers for on-chip modules.

Table A-1. CPU Space Register Memory Map

Address	Name	Mnemonic	Size (bits)
CPU @ 0x800	Other Stack Pointer	OTHER_A7	32
CPU @ 0x801	Vector Base Register	VBR	32
CPU @ 0x804	MAC Status Register	MACSR	8
CPU @ 0x805	MAC Mask Register	MASK	16
CPU @ 0x806	MAC Accumulator 0	ACC0	16
CPU @ 0x80E	Status Register	SR	16
CPU @ 0x80F	Program Counter	PC	32
CPU @ 0xC04	Flash Base Address Register	FLASHBAR	32
CPU @ 0xC05	RAM Base Address Register	RAMBAR	32

Table A-2. Module Memory Map Overview

Address	Module	Size
0x0000_0000	On-chip Flash/RAM Array	1G
IPSBAR + 0x00_0000	System Control Module	64 bytes
IPSBAR + 0x00_0040	Reserved	64 bytes
IPSBAR + 0x00_0080	Reserved	128 bytes
IPSBAR + 0x00_0100	DMA (Channel 0)	16 bytes
IPSBAR + 0x00_0110	DMA (Channel 1)	16 bytes
IPSBAR + 0x00_0120	DMA (Channel 2)	16 bytes
IPSBAR + 0x00_0130	DMA (Channel 3)	16 bytes
IPSBAR + 0x00_0140	Reserved	196 bytes
IPSBAR + 0x00_0200	UART0	64 bytes
IPSBAR + 0x00_0240	UART1	64 bytes
IPSBAR + 0x00_0280	UART2	64 bytes
IPSBAR + 0x00_02C0	Reserved	64 bytes
IPSBAR + 0x00_0300	I^2C	64 bytes
IPSBAR + 0x00_0340	QSPI	64 bytes
IPSBAR + 0x00_0380	Reserved	64 bytes
IPSBAR + 0x00_03C0	RTC	64 bytes
IPSBAR + 0x00_0400	DMA Timer 0	64 bytes
IPSBAR + 0x00_0440	DMA Timer 1	64 bytes
IPSBAR + 0x00_0480	DMA Timer 2	64 bytes
IPSBAR + 0x00_04C0	DMA Timer 3	64 bytes
IPSBAR + 0x00_0500	Reserved	1792 bytes
IPSBAR + 0x00_0C00	Interrupt Controller 0	256 bytes
IPSBAR + 0x00_0D00	Interrupt Controller 1	256 bytes
IPSBAR + 0x00_0E00	Reserved	256
IPSBAR + 0x00_0F00	Global Interrupt Acknowledge Registers	256 bytes
IPSBAR + 0x00_1000	FEC Registers and MIB RAM	1K
IPSBAR + 0x00_1400	FEC FIFO Memory	1K
IPSBAR + 0x00_1800	Reserved	1M - 6K
IPSBAR + 0x10_0000	Ports	64K
IPSBAR + 0x11_0000	Reset Controller, Chip Configuration, and Power Management	64K
IPSBAR + 0x12_0000	Clock Module	64K
IPSBAR + 0x13_0000	Edge Port 0	64K
IPSBAR + 0x14_0000	Edge Port 1	64K
IPSBAR + 0x15_0000	Programmable Interval Timer 0	64K

Table A-2. Module Memory Map Overview (continued)

Address	Module	Size
IPSBAR + 0x16_0000	Programmable Interval Timer 1	64K
IPSBAR + 0x17_0000	Reserved	64K
IPSBAR + 0x18_0000	Reserved	64K
IPSBAR + 0x19_0000	ADC	64K
IPSBAR + 0x1A_0000	General Purpose Timer A	64K
IPSBAR + 0x1B_0000	PWM	64K
IPSBAR + 0x1C_0000	FlexCAN	64K
IPSBAR + 0x1D_0000	CFM (Flash) Control Registers	64K
IPSBAR + 0x1E_0000	Ethernet Physical Transceiver	64K
IPSBAR + 0x1F_0000	Random Number Generator H/W Accelerator	64K
IPSBAR + 0x20_0000	Reserved	62M
IPSBAR + 0x400_0000	CFM (Flash) Memory for IPS Reads and Writes	256K
IPSBAR + 0x0408_0000	Reserved	1G - 64M - 256K
IPSBAR + 0x8000_0000	Reserved	2G

Table A-3. Register Memory Map

Address	Name	Mnemonic	Size
SCM Registers			
IPSBAR + 0x0000	Internal Peripheral System Base Address Register	IPSBAR	32
IPSBAR + 0x0008	Peripheral Power Management Register - High	PPMRH	32
IPSBAR + 0x000C	Peripheral Power Management Register - Low	PPMRL	32
IPSBAR + 0x0010	Core Reset Status Register	CRSR	8
IPSBAR + 0x0011	Core Watchdog Control Register	CWCR	8
IPSBAR + 0x0012	Low-Power Interrupt Control Register	LPICR	8
IPSBAR + 0x0013	Core Watchdog Service Register	CWSR	8
IPSBAR + 0x0014	DMA Request Control Register	DMAREQC	32
IPSBAR + 0x001C	Default Bus Master Park Register	MPARK	32
IPSBAR + 0x0020	Master Privilege Register	MPR	8
IPSBAR + 0x0024	Peripheral Access Control Register 0	PACR0	8
IPSBAR + 0x0025	Peripheral Access Control Register 1	PACR1	8
IPSBAR + 0x0026	Peripheral Access Control Register 2	PACR2	8
IPSBAR + 0x0027	Peripheral Access Control Register 3	PACR3	8
IPSBAR + 0x0028	Peripheral Access Control Register 4	PACR4	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0029	Peripheral Access Control Register 5	PACR5	8
IPSBAR + 0x002A	Peripheral Access Control Register 6	PACR6	8
IPSBAR + 0x002B	Peripheral Access Control Register 7	PACR7	8
IPSBAR + 0x002C	Peripheral Access Control Register 8	PACR8	8
IPSBAR + 0x0030	Grouped Peripheral Access Control Register 0	GPACR0	8
IPSBAR + 0x0031	Grouped Peripheral Access Control Register 1	GPACR1	8
DMA Registers			
IPSBAR + 0x0100	Source Address Register 0	SAR0	32
IPSBAR + 0x0104	Destination Address Register 0	DAR0	32
IPSBAR + 0x0108	Byte Count Register 0 / DMA Status Register 0	BCR0 / DSR0	32
IPSBAR + 0x010C	DMA Control Register 0	DCR0	32
IPSBAR + 0x0110	Source Address Register 1	SAR1	32
IPSBAR + 0x0114	Destination Address Register 1	DAR1	32
IPSBAR + 0x0118	Byte Count Register 1 / DMA Status Register 1	BCR1 / DSR1	32
IPSBAR + 0x011C	DMA Control Register 1	DCR1	32
IPSBAR + 0x0120	Source Address Register 2	SAR2	32
IPSBAR + 0x0124	Destination Address Register 2	DAR2	32
IPSBAR + 0x0128	Byte Count Register 2 / DMA Status Register 2	BCR2 / DSR2	32
IPSBAR + 0x012C	DMA Control Register 2	DCR2	32
IPSBAR + 0x0130	Source Address Register 3	SAR3	32
IPSBAR + 0x0134	Destination Address Register 3	DAR3	32
IPSBAR + 0x0138	Byte Count Register 3 / DMA Status Register 3	BCR3 / DSR3	32
IPSBAR + 0x013C	DMA Control Register 3	DCR3	32
UART Registers			
IPSBAR + 0x0200	UART Mode Register 0[1]	UMR10, UMR20	8
IPSBAR + 0x0204	(Read) UART Status Register 0	USR0	8
	(Write) UART Clock Select Register 0[1]	UCSR0	8
IPSBAR + 0x0208	(Read) Reserved		8
	(Write) UART Command Register 0	UCR0	8
IPSBAR + 0x020C	(Read) UART Receive Buffer 0	URB0	8
	(Write) UART Transmit Buffer 0	UTB0	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0210	(Read) UART Input Port Change Register 0	UIPCR0	8
	(Write) UART Auxiliary Control Register 0[1]	UACR0	8
IPSBAR + 0x0214	(Read) UART Interrupt Status Register 0	UISR0	8
	(Write) UART Interrupt Mask Register 0	UIMR0	8
IPSBAR + 0x0218	(Read) Reserved		8
	UART Baud Rate Generator Register 10	UBG10	8
IPSBAR + 0x021C	(Read) Reserved		8
	UART Baud Rate Generator Register 20	UBG20	8
IPSBAR + 0x0234	(Read) UART Input Port Register 0	UIP0	8
	(Write) Reserved		8
IPSBAR + 0x0238	(Read) Reserved		8
	(Write) UART Output Port Bit Set Command Register 0	UOP10	8
IPSBAR + 0x023C	(Read) Reserved		8
	(Write) UART Output Port Bit Reset Command Register 0	UIP00	8
IPSBAR + 0x0240	UART Mode Registers 1[1]	UMR11, UMR21	8
IPSBAR + 0x0244	(Read) UART Status Register 1	USR1	8
	(Write) UART Clock Select Register 1[1]	UCSR1	8
IPSBAR + 0x0248	(Read) Reserved		8
	(Write) UART Command Register 1	UCR1	8
IPSBAR + 0x024C	(UART/Read) UART Receive Buffer 1	URB1	8
	(UART/Write) UART Transmit Buffer 1	UTB1	8
IPSBAR + 0x0250	(Read) UART Input Port Change Register 1	UIPCR1	8
	(Write) UART Auxiliary Control Register 1[1]	UACR1	8
IPSBAR + 0x0254	(Read) UART Interrupt Status Register 1	UISR1	8
	(Write) UART Interrupt Mask Register 1	UIMR1	8
IPSBAR + 0x0258	(Read) Reserved		8
	UART Baud Rate Generator Register 11	UBG11	8
IPSBAR + 0x025C	(Read) Reserved		8
	UART Baud Rate Generator Register 21	UBG21	8
IPSBAR + 0x0274	(Read) UART Input Port Register 1	UIP1	8
	(Write) Reserved		8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0278	(Read) Reserved		8
	(Write) UART Output Port Bit Set Command Register 1	UOP11	8
IPSBAR + 0x027C	(Read) Reserved		8
	(Write) UART Output Port Bit Reset Command Register 1	UIP01	8
IPSBAR + 0x0280	UART Mode Register 2[1]	UMR12, UMR22	8
IPSBAR + 0x0284	(Read) UART Status Register 2	USR2	8
	(Write) UART Clock Select Register 2[1]	UCSR2	8
IPSBAR + 0x0288	(Read) Reserved		8
	(Write) UART Command Register 2	UCR2	8
IPSBAR + 0x028C	(Read) UART Receive Buffer 2	URB2	8
	(Write) UART Transmit Buffer 2	UTB2	8
IPSBAR + 0x0290	(Read) UART Input Port Change Register 2	UIPCR2	8
	(Write) UART Auxiliary Control Register 2[1]	UACR2	8
IPSBAR + 0x0294	(Read) UART Interrupt Status Register 2	UISR2	8
	(Write) UART Interrupt Mask Register 2	UIMR2	8
IPSBAR + 0x0298	(Read) Reserved		8
	UART Baud Rate Generator Register 12	UBG12	8
IPSBAR + 0x029C	(Read) Reserved		8
	UART Baud Rate Generator Register 22	UBG22	8
IPSBAR + 0x02B4	(Read) UART Input Port Register 2	UIP2	8
	(Write) Reserved		8
IPSBAR + 0x02B8	(Read) Reserved		8
	(Write) UART Output Port Bit Set Command Register 2	UOP12	8
IPSBAR + 0x02BC	(Read) Reserved		8
	(Write) UART Output Port Bit Reset Command Register 2	UIP02	8
I^2C Registers			
IPSBAR + 0x0300	I^2C Address Register	I2ADR	8
IPSBAR + 0x0304	I^2C Frequency Divider Register	I2FDR	8
IPSBAR + 0x0308	I^2C Control Register	I2CR	8
IPSBAR + 0x030C	I^2C Status Register	I2SR	8
IPSBAR + 0x0310	I^2C Data I/O Register	I2DR	8
QSPI Registers			

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0340	QSPI Mode Register	QMR	16
IPSBAR + 0x0344	QSPI Delay Register	QDLYR	16
IPSBAR + 0x0348	QSPI Wrap Register	QWR	16
IPSBAR + 0x034C	QSPI Interrupt Register	QIR	16
IPSBAR + 0x0350	QSPI Address Register	QAR	16
IPSBAR + 0x0354	QSPI Data Register	QDR	16
Real Time Clock Registers			
IPSBAR + 0x03C0	RTC Hours and Minutes Register	HOURMIN	32
IPSBAR + 0x03C4	RTC Seconds Counter Register	SECONDS	32
IPSBAR + 0x03C8	RTC Hours and Minutes Alarm Register	ALRM_HM	32
IPSBAR + 0x03CC	RTC Seconds Alarm Register	ALRM_SEC	32
IPSBAR + 0x03D0	RTC Control Register	RTCCTL	32
IPSBAR + 0x03D4	RTC Interrupt Status Register	RTCISR	32
IPSBAR + 0x03D8	RTC Interrupt Enable Register	RTCIENR	32
IPSBAR + 0x03DC	RTC Stopwatch Minutes Register	STPWCH	32
IPSBAR + 0x03E0	RTC Days Counter Register	DAYS	32
IPSBAR + 0x03E4	RTC Days Alarm Register	ALRM_DAY	32
DMA Timer Registers			
IPSBAR + 0x0400	DMA Timer Mode Register 0	DTMR0	16
IPSBAR + 0x0402	DMA Timer Extended Mode Register 0	DTXMR0	8
IPSBAR + 0x0403	DMA Timer Event Register 0	DTER0	8
IPSBAR + 0x0404	DMA Timer Reference Register 0	DTRR0	32
IPSBAR + 0x0408	DMA Timer Capture Register 0	DTCR0	32
IPSBAR + 0x040C	DMA Timer Counter Register 0	DTCN0	32
IPSBAR + 0x0440	DMA Timer Mode Register 1	DTMR1	16
IPSBAR + 0x0442	DMA Timer Extended Mode Register 1	DTXMR1	8
IPSBAR + 0x0443	DMA Timer Event Register 1	DTER1	8
IPSBAR + 0x0444	DMA Timer Reference Register 1	DTRR1	32
IPSBAR + 0x0448	DMA Timer Capture Register 1	DTCR1	32
IPSBAR + 0x044C	DMA Timer Counter Register 1	DTCN1	32
IPSBAR + 0x0480	DMA Timer Mode Register 2	DTMR2	16
IPSBAR + 0x0482	DMA Timer Extended Mode Register 2	DTXMR2	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0483	DMA Timer Event Register 2	DTER2	8
IPSBAR + 0x0484	DMA Timer Reference Register 2	DTRR2	32
IPSBAR + 0x0488	DMA Timer Capture Register 2	DTCR2	32
IPSBAR + 0x048C	DMA Timer Counter Register 2	DTCN2	32
IPSBAR + 0x04C0	DMA Timer Mode Register 3	DTMR3	16
IPSBAR + 0x04C2	DMA Timer Extended Mode Register 3	DTXMR3	8
IPSBAR + 0x04C3	DMA Timer Event Register 3	DTER3	8
IPSBAR + 0x04C4	DMA Timer Reference Register 3	DTRR3	32
IPSBAR + 0x04C8	DMA Timer Capture Register 3	DTCR3	32
IPSBAR + 0x04CC	DMA Timer Counter Register 3	DTCN3	32
Interrupt Controller 0			
IPSBAR + 0x0C00	Interrupt Pending Register High 0	IPRH0	32
IPSBAR + 0x0C04	Interrupt Pending Register Low 0	IPRL0	32
IPSBAR + 0x0C08	Interrupt Mask Register High 0	IMRH0	32
IPSBAR + 0x0C0C	Interrupt Mask Register Low 0	IMRL0	32
IPSBAR + 0x0C10	Interrupt Force Register High 0	INTFRCH0	32
IPSBAR + 0x0C14	Interrupt Force Register Low 0	INTFRCL0	32
IPSBAR + 0x0C18	Interrupt Request Level Register 0	IRLR0	8
IPSBAR + 0x0C19	Interrupt Acknowledge Level and Priority Register 0	IACKLPR0	8
IPSBAR + 0x0C41	Interrupt Control Register 0-01	ICR001	8
IPSBAR + 0x0C42	Interrupt Control Register 0-02	ICR002	8
IPSBAR + 0x0C43	Interrupt Control Register 0-03	ICR003	8
IPSBAR + 0x0C44	Interrupt Control Register 0-04	ICR004	8
IPSBAR + 0x0C45	Interrupt Control Register 0-05	ICR005	8
IPSBAR + 0x0C46	Interrupt Control Register 0-06	ICR006	8
IPSBAR + 0x0C47	Interrupt Control Register 0-07	ICR007	8
IPSBAR + 0x0C48	Interrupt Control Register 0-08	ICR008	8
IPSBAR + 0x0C49	Interrupt Control Register 0-09	ICR009	8
IPSBAR + 0x0C4A	Interrupt Control Register 0-10	ICR010	8
IPSBAR + 0x0C4B	Interrupt Control Register 0-11	ICR011	8
IPSBAR + 0x0C4C	Interrupt Control Register 0-12	ICR012	8
IPSBAR + 0x0C4D	Interrupt Control Register 0-13	ICR013	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0C4E	Interrupt Control Register 0-14	ICR014	8
IPSBAR + 0x0C4F	Interrupt Control Register 0-15	ICR015	8
IPSBAR + 0x0C50	Interrupt Control Register 0-16	ICR016	8
IPSBAR + 0x0C51	Interrupt Control Register 0-17	ICR017	8
IPSBAR + 0x0C52	Interrupt Control Register 0-18	ICR018	8
IPSBAR +0xC53	Interrupt Control Register 0-19	ICR019	8
IPSBAR + 0x0C54	Interrupt Control Register 0-20	ICR020	8
IPSBAR + 0x0C55	Interrupt Control Register 0-21	ICR021	8
IPSBAR + 0x0C56	Interrupt Control Register 0-22	ICR022	8
IPSBAR + 0x0C57	Interrupt Control Register 0-23	ICR023	8
IPSBAR + 0x0C58	Interrupt Control Register 0-24	ICR024	8
IPSBAR + 0x0C59	Interrupt Control Register 0-25	ICR025	8
IPSBAR + 0x0C5A	Interrupt Control Register 0-26	ICR026	8
IPSBAR + 0x0C5B	Interrupt Control Register 0-27	ICR027	8
IPSBAR + 0x0C5C	Interrupt Control Register 0-28	ICR028	8
IPSBAR + 0x0C5D	Interrupt Control Register 0-29	ICR029	8
IPSBAR + 0x0C5E	Interrupt Control Register 0-30	ICR030	8
IPSBAR + 0x0C5F	Interrupt Control Register 0-31	ICR031	8
IPSBAR + 0x0C60	Interrupt Control Register 0-32	ICR032	8
IPSBAR + 0x0C61	Interrupt Control Register 0-33	ICR033	8
IPSBAR + 0x0C62	Interrupt Control Register 0-34	ICR034	8
IPSBAR + 0x0C63	Interrupt Control Register 0-35	ICR035	8
IPSBAR + 0x0C64	Interrupt Control Register 0-36	ICR036	8
IPSBAR + 0x0C65	Interrupt Control Register 0-37	ICR037	8
IPSBAR + 0x0C66	Interrupt Control Register 0-38	ICR038	8
IPSBAR + 0x0C67	Interrupt Control Register 0-39	ICR039	8
IPSBAR + 0x0C68	Interrupt Control Register 0-40	ICR040	8
IPSBAR + 0x0C69	Interrupt Control Register 0-41	ICR041	8
IPSBAR + 0x0C6A	Interrupt Control Register 0-42	ICR042	8
IPSBAR + 0x0C6B	Interrupt Control Register 0-43	ICR043	8
IPSBAR + 0x0C6C	Interrupt Control Register 0-44	ICR044	8
IPSBAR + 0x0C6D	Interrupt Control Register 0-45	ICR045	8
IPSBAR + 0x0C6E	Interrupt Control Register 0-46	ICR046	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0C6F	Interrupt Control Register 0-47	ICR047	8
IPSBAR + 0x0C70	Interrupt Control Register 0-48	ICR048	8
IPSBAR + 0x0C71	Interrupt Control Register 0-49	ICR049	8
IPSBAR + 0x0C72	Interrupt Control Register 0-50	ICR050	8
IPSBAR + 0x0C73	Interrupt Control Register 0-51	ICR051	8
IPSBAR + 0x0C74	Interrupt Control Register 0-52	ICR052	8
IPSBAR + 0x0C75	Interrupt Control Register 0-53	ICR053	8
IPSBAR + 0x0C76	Interrupt Control Register 0-54	ICR054	8
IPSBAR + 0x0C77	Interrupt Control Register 0-55	ICR055	8
IPSBAR + 0x0C78	Interrupt Control Register 0-56	ICR056	8
IPSBAR + 0x0C79	Interrupt Control Register 0-57	ICR057	8
IPSBAR + 0x0C7A	Interrupt Control Register 0-58	ICR058	8
IPSBAR + 0x0C7B	Interrupt Control Register 0-59	ICR059	8
IPSBAR + 0x0C7C	Interrupt Control Register 0-60	ICR060	8
IPSBAR + 0x0C7D	Interrupt Control Register 0-61	ICR061	8
IPSBAR + 0x0C7E	Interrupt Control Register 0-62	ICR062	8
IPSBAR + 0x0CE0	Software Interrupt Acknowledge Register 0	SWACKR0	8
IPSBAR + 0x0CE4	Level 1 Interrupt Acknowledge Register 0	L1IACKR0	8
IPSBAR + 0x0CE8	Level 2 Interrupt Acknowledge Register 0	L2IACKR0	8
IPSBAR + 0x0CEC	Level 3 Interrupt Acknowledge Register 0	L3IACKR0	8
IPSBAR + 0x0CF0	Level 4 Interrupt Acknowledge Register 0	L4IACKR0	8
IPSBAR + 0x0CF4	Level 5 Interrupt Acknowledge Register 0	L5IACKR0	8
IPSBAR + 0x0CF8	Level 6 Interrupt Acknowledge Register 0	L6IACKR0	8
IPSBAR + 0x0CFC	Level 7 Interrupt Acknowledge Register 0	L7IACKR0	8
Interrupt Controller 1			
IPSBAR + 0x0D00	Interrupt Pending Register High 11	IPRH1	32
IPSBAR + 0x0D04	Interrupt Pending Register Low 11	IPRL1	32
IPSBAR + 0x0D08	Interrupt Mask Register High 11	IMRH1	32
IPSBAR + 0x0D0C	Interrupt Mask Register Low 1	IMRL1	32
IPSBAR + 0x0D10	Interrupt Force Register High 1	INTFRCH1	32
IPSBAR + 0x0D14	Interrupt Force Register Low 1	INTFRCL1	32
IPSBAR + 0x0D18	Interrupt Request Level Register 1	IRLR1	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0D19	Interrupt Acknowledge Level and Priority Register 1	IACKLPR1	8
IPSBAR + 0x0D41	Interrupt Control Register 1-01	ICR101	8
IPSBAR + 0x0D42	Interrupt Control Register 1-02	ICR102	8
IPSBAR + 0x0D43	Interrupt Control Register 1-03	ICR103	8
IPSBAR + 0x0D44	Interrupt Control Register 1-04	ICR104	8
IPSBAR + 0x0D45	Interrupt Control Register 1-05	ICR105	8
IPSBAR + 0x0D46	Interrupt Control Register 1-06	ICR106	8
IPSBAR + 0x0D47	Interrupt Control Register 1-07	ICR107	8
IPSBAR + 0x0D48	Interrupt Control Register 1-08	ICR108	8
IPSBAR + 0x0D49	Interrupt Control Register 1-09	ICR109	8
IPSBAR + 0x0D4A	Interrupt Control Register 1-10	ICR110	8
IPSBAR + 0x0D4B	Interrupt Control Register 1-11	ICR111	8
IPSBAR + 0x0D4C	Interrupt Control Register 1-12	ICR112	8
IPSBAR + 0x0D4D	Interrupt Control Register 1-13	ICR113	8
IPSBAR + 0x0D4E	Interrupt Control Register 1-14	ICR114	8
IPSBAR + 0x0D4F	Interrupt Control Register 1-15	ICR115	8
IPSBAR + 0x0D50	Interrupt Control Register 1-16	ICR116	8
IPSBAR + 0x0D51	Interrupt Control Register 1-17	ICR117	8
IPSBAR + 0x0D52	Interrupt Control Register 1-18	ICR118	8
IPSBAR + 0x0C53	Interrupt Control Register 1-19	ICR119	8
IPSBAR + 0x0D54	Interrupt Control Register 1-20	ICR120	8
IPSBAR + 0x0D55	Interrupt Control Register 1-21	ICR121	8
IPSBAR + 0x0D56	Interrupt Control Register 1-22	ICR122	8
IPSBAR + 0x0D57	Interrupt Control Register 1-23	ICR123	8
IPSBAR + 0x0D58	Interrupt Control Register 1-24	ICR124	8
IPSBAR + 0x0D59	Interrupt Control Register 1-25	ICR125	8
IPSBAR + 0x0D5A	Interrupt Control Register 1-26	ICR126	8
IPSBAR + 0x0D5B	Interrupt Control Register 1-27	ICR127	8
IPSBAR + 0x0D5C	Interrupt Control Register 1-28	ICR128	8
IPSBAR + 0x0D5D	Interrupt Control Register 1-29	ICR129	8
IPSBAR + 0x0D5E	Interrupt Control Register 1-30	ICR130	8
IPSBAR + 0x0D5F	Interrupt Control Register 1-31	ICR131	8
IPSBAR + 0x0D61	Interrupt Control Register 1-32	ICR132	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0D61	Interrupt Control Register 1-33	ICR133	8
IPSBAR + 0x0C62	Interrupt Control Register 1-34	ICR134	8
IPSBAR + 0x0D63	Interrupt Control Register 1-35	ICR135	8
IPSBAR + 0x0D64	Interrupt Control Register 1-36	ICR136	8
IPSBAR + 0x0D65	Interrupt Control Register 1-37	ICR137	8
IPSBAR + 0x0D66	Interrupt Control Register 1-38	ICR138	8
IPSBAR + 0x0D67	Interrupt Control Register 1-39	ICR139	8
IPSBAR + 0x0D68	Interrupt Control Register 1-40	ICR140	8
IPSBAR + 0x0D69	Interrupt Control Register 1-41	ICR141	8
IPSBAR + 0x0D6A	Interrupt Control Register 1-42	ICR142	8
IPSBAR + 0x0D6B	Interrupt Control Register 1-43	ICR143	8
IPSBAR + 0x0D6C	Interrupt Control Register 1-44	ICR144	8
IPSBAR + 0x0D6D	Interrupt Control Register 1-45	ICR145	8
IPSBAR + 0x0D6E	Interrupt Control Register 1-46	ICR146	8
IPSBAR + 0x0D6F	Interrupt Control Register 1-47	ICR147	8
IPSBAR + 0x0D71	Interrupt Control Register 1-48	ICR148	8
IPSBAR + 0x0D71	Interrupt Control Register 1-49	ICR149	8
IPSBAR + 0x0D72	Interrupt Control Register 1-50	ICR150	8
IPSBAR + 0x0D73	Interrupt Control Register 1-51	ICR151	8
IPSBAR + 0x0D74	Interrupt Control Register 1-52	ICR152	8
IPSBAR + 0x0D75	Interrupt Control Register 1-53	ICR153	8
IPSBAR + 0x0D76	Interrupt Control Register 1-54	ICR154	8
IPSBAR + 0x0D77	Interrupt Control Register 1-55	ICR155	8
IPSBAR + 0x0D78	Interrupt Control Register 1-56	ICR156	8
IPSBAR + 0x0D79	Interrupt Control Register 1-57	ICR157	8
IPSBAR + 0x0D7A	Interrupt Control Register 1-58	ICR158	8
IPSBAR + 0x0D7B	Interrupt Control Register 1-59	ICR159	8
IPSBAR + 0x0D7C	Interrupt Control Register 1-60	ICR160	8
IPSBAR + 0x0D7D	Interrupt Control Register 1-61	ICR161	8
IPSBAR + 0x0D7E	Interrupt Control Register 1-62	ICR162	8
IPSBAR + 0x0DE0	Software Interrupt Acknowledge Register 1	SWACKR1	8
IPSBAR + 0x0DE4	Level 1 Interrupt Acknowledge Register 1	L1IACKR1	8
IPSBAR + 0x0DE8	Level 2 Interrupt Acknowledge Register 1	L2IACKR1	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x0DEC	Level 3 Interrupt Acknowledge Register 1	L3IACKR1	8
IPSBAR + 0x0DF1	Level 4 Interrupt Acknowledge Register 1	L4IACKR0	8
IPSBAR + 0x0DF4	Level 5 Interrupt Acknowledge Register 1	L5IACKR1	8
IPSBAR + 0x0DF8	Level 6 Interrupt Acknowledge Register 1	L6IACKR1	8
IPSBAR + 0x0DFC	Level 7 Interrupt Acknowledge Register 1	L7IACKR1	8
Global Interrupt Acknowledge Registers			
IPSBAR + 0x0FE4	Global Level 1 Interrupt Acknowledge Register	GL1IACK	8
IPSBAR + 0x0FE8	Global Level 2 Interrupt Acknowledge Register	GL2IACK	8
IPSBAR + 0x0FEC	Global Level 3 Interrupt Acknowledge Register	GL3IACK	8
IPSBAR + 0x0FF0	Global Level 4 Interrupt Acknowledge Register	GL4IACK	8
IPSBAR + 0x0FF4	Global Level 5 Interrupt Acknowledge Register	GL5IACK	8
IPSBAR + 0x0FF8	Global Level 6 Interrupt Acknowledge Register	GL6IACK	8
IPSBAR + 0x0FFC	Global Level 7 Interrupt Acknowledge Register	GL7IACK	8
Fast Ethernet Controller Registers			
IPSBAR + 0x1004	Interrupt Event Register	EIR	32
IPSBAR + 0x1008	Interrupt Mask Register	EIMR	32
IPSBAR + 0x1010	Receive Descriptor Active Register	RDAR	32
IPSBAR + 0x1014	Transmit Descriptor Active Register	TDAR	32
IPSBAR + 0x1024	Ethernet Control Register	ECR	32
IPSBAR + 0x1040	MII Data Register	MDATA	32
IPSBAR + 0x1044	MII Speed Control Register	MSCR	32
IPSBAR + 0x1064	MIB Control/Status Register	MIBC	32
IPSBAR + 0x1084	Receive Control Register	RCR	32
IPSBAR + 0x10C4	Transmit Control Register	TCR	32
IPSBAR + 0x10E4	Physical Address Low Register	PALR	32
IPSBAR + 0x10E8	Physical Address High Register	PAUR	32
IPSBAR + 0x10EC	Opcode / Pause Duration Register	OPD	32
IPSBAR + 0x1118	Descriptor Individual Upper Address Register	IAUR	32
IPSBAR + 0x111C	Descriptor Individual Lower Address Register	IALR	32
IPSBAR + 0x1120	Descriptor Group Upper Address Register	GAUR	32
IPSBAR + 0x1124	Descriptor Group Lower Address Register	GALR	32
IPSBAR + 0x1144	Transmit FIFO Watermark	TFWR	32

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x114C	FIFO Receive Bound Register	FRBR	32
IPSBAR + 0x1150	FIFO Receive FIFO Start Register	FRSR	32
IPSBAR + 0x1180	Receive Descriptor Ring Start Register	ERDSR	32
IPSBAR + 0x1184	Transmit Buffer Descriptor Ring Start Register	ETDSR	32
IPSBAR + 0x1188	Receive Buffer Size Register	EMRBR	32
GPIO Registers			
IPSBAR + 0x10_0000	Reserved	—	8
IPSBAR + 0x10_0001	Reserved	—	8
IPSBAR + 0x10_0002	Reserved	—	8
IPSBAR + 0x10_0003	Reserved	—	8
IPSBAR + 0x10_0004	Reserved	—	8
IPSBAR + 0x10_0005	Reserved	—	8
IPSBAR + 0x10_0006	Reserved	—	8
IPSBAR + 0x10_0007	Reserved	—	8
IPSBAR + 0x10_0008	Port NQ Out Data Register	PORTNQ	8
IPSBAR + 0x10_0009	Reserved	—	8
IPSBAR + 0x10_000A	Port AN Output Data Register	PORTAN	8
IPSBAR + 0x10_000B	Port AS Output Data Register	PORTAS	8
IPSBAR + 0x10_000C	Port QS Output Data Register	PORTQS	8
IPSBAR + 0x10_000D	Reserved	—	8
IPSBAR + 0x10_000E	Port TA Output Data Register	PORTTA	8
IPSBAR + 0x10_000F	Port TC Output Data Register	PORTTC	8
IPSBAR + 0x10_0010	Port TD Output Data Register	PORTTD	8
IPSBAR + 0x10_0011	Port UA Output Data Register	PORTUA	8
IPSBAR + 0x10_0012	Port UB Output Data Register	PORTUB	8
IPSBAR + 0x10_0013	Port UC Output Data Register	PORTUC	8
IPSBAR + 0x10_0014	Port DD Output Data Register	PORTDD	8
IPSBAR + 0x10_0015	Port LD Output Data Register	PORTLD	8
IPSBAR + 0x10_0016	Port GP Output Data Register	PORTGP	8
IPSBAR + 0x10_0017	Reserved	—	8
IPSBAR + 0x10_0018	Reserved	—	8
IPSBAR + 0x10_0019	Reserved	—	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x10_001A	Reserved	—	8
IPSBAR + 0x10_001B	Reserved	—	8
IPSBAR + 0x10_001C	Reserved	—	8
IPSBAR + 0x10_001D	Reserved	—	8
IPSBAR + 0x10_001E	Reserved	—	8
IPSBAR + 0x10_001F	Reserved	—	8
IPSBAR + 0x10_0020	Port NQ Data Direction Register	DDRNQ	8
IPSBAR + 0x10_0021	Reserved	—	8
IPSBAR + 0x10_0022	Port AN Data Direction Register	DDRAN	8
IPSBAR + 0x10_0023	Port AS Data Direction Register	DDRAS	8
IPSBAR + 0x10_0024	Port QS Data Direction Register	DDRQS	8
IPSBAR + 0x10_0025	Reserved	—	8
IPSBAR + 0x10_0026	Port TA Data Direction Register	DDRTA	8
IPSBAR + 0x10_0027	Port TC Data Direction Register	DDRTC	8
IPSBAR + 0x10_0028	Port TD Data Direction Register	DDRTD	8
IPSBAR + 0x10_0029	Port UA Data Direction Register	DDRUA	8
IPSBAR + 0x10_002A	Port UB Data Direction Register	DDRUB	8
IPSBAR + 0x10_002B	Port UC Data Direction Register	DDRUC	8
IPSBAR + 0x10_002C	Port DD Data Direction Register	DDRDD	8
IPSBAR + 0x10_002D	Port LD Data Direction Register	DDRLD	8
IPSBAR + 0x10_002E	Port GP Data Direction Register	DDRGP	8
IPSBAR + 0x10_002F	Reserved	—	8
IPSBAR + 0x10_0030	Reserved	—	8
IPSBAR + 0x10_0031	Reserved	—	8
IPSBAR + 0x10_0032	Reserved	—	8
IPSBAR + 0x10_0033	Reserved	—	8
IPSBAR + 0x10_0034	Reserved	—	8
IPSBAR + 0x10_0035	Reserved	—	8
IPSBAR + 0x10_0036	Reserved	—	8
IPSBAR + 0x10_0037	Reserved	—	8
IPSBAR + 0x10_0038	Port NQ Pin Data/Set Data Register	SETNQ	8
IPSBAR + 0x10_0039	Reserved	—	8
IPSBAR + 0x10_003A	Port AN Pin Data/Set Data Register	SETAN	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x10_003B	Port AS Pin Data/Set Data Register	SETAS	8
IPSBAR + 0x10_003C	Port QS Pin Data/Set Data Register	SETQS	8
IPSBAR + 0x10_003D	Reserved	—	8
IPSBAR + 0x10_003E	Port TA Pin Data/Set Data Register	SETTA	8
IPSBAR + 0x10_003F	Port TC Pin Data/Set Data Register	SETTC	8
IPSBAR + 0x10_0040	Port TD Pin Data/Set Data Register	SETTD	8
IPSBAR + 0x10_0041	Port UA Pin Data/Set Data Register	SETUA	8
IPSBAR + 0x10_0042	Port UB Pin Data/Set Data Register	SETUB	8
IPSBAR + 0x10_0043	Port UC Pin Data/Set Data Register	SETUC	8
IPSBAR + 0x10_0044	Port DD Pin Data/Set Data Register	SETDD	8
IPSBAR + 0x10_0045	Port LD Pin Data/Set Data Register	SETLD	8
IPSBAR + 0x10_0046	Port GP Pin Data/Set Data Register	SETGP	8
IPSBAR + 0x10_0047	Reserved	—	8
IPSBAR + 0x10_0048	Reserved	—	8
IPSBAR + 0x10_0049	Reserved	—	8
IPSBAR + 0x10_004A	Reserved	—	8
IPSBAR + 0x10_004B	Reserved	—	8
IPSBAR + 0x10_004C	Reserved	—	8
IPSBAR + 0x10_004D	Reserved	—	8
IPSBAR + 0x10_004E	Reserved	—	8
IPSBAR + 0x10_004F	Reserved	—	8
IPSBAR + 0x10_0050	Port NQ Clear Output Data Register	CLRNQ	8
IPSBAR + 0x10_0051	Reserved	—	8
IPSBAR + 0x10_0052	Port AN Clear Output Data Register	CLRAN	8
IPSBAR + 0x10_0053	Port AS Clear Output Data Register	CLRAS	8
IPSBAR + 0x10_0054	Port QS Clear Output Data Register	CLRQS	8
IPSBAR + 0x10_0055	Reserved	—	8
IPSBAR + 0x10_0056	Port TA Clear Output Data Register	CLRTA	8
IPSBAR + 0x10_0057	Port TC Clear Output Data Register	CLRTC	8
IPSBAR + 0x10_0058	Port TD Clear Output Data Register	CLRTD	8
IPSBAR + 0x10_0059	Port UA Clear Output Data Register	CLRUA	8
IPSBAR + 0x10_005A	Port UB Clear Output Data Register	CLRUB	8
IPSBAR + 0x10_005B	Port UC Clear Output Data Register	CLRUC	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x10_005C	Port DD Clear Output Data Register	CLRDD	8
IPSBAR + 0x10_005D	Port LD Clear Output Data Register	CLRLD	8
IPSBAR + 0x10_005E	Port GP Clear Output Data Register	CLRGP	8
IPSBAR + 0x10_005F	Reserved	—	8
IPSBAR + 0x10_0060	Reserved	—	8
IPSBAR + 0x10_0061	Reserved	—	8
IPSBAR + 0x10_0062	Reserved	—	8
IPSBAR + 0x10_0063	Reserved	—	8
IPSBAR + 0x10_0064	Reserved	—	8
IPSBAR + 0x10_0065	Reserved	—	8
IPSBAR + 0x10_0066	Reserved	—	8
IPSBAR + 0x10_0067	Reserved	—	8
IPSBAR + 0x10_0068	Port NQ Pin Assignment Register	PNQPAR	16
IPSBAR + 0x10_006A	Port AN Pin Assignment Register	PANPAR	8
IPSBAR + 0x10_006B	Port AS Pin Assignment Register	PASPAR	8
IPSBAR + 0x10_006C	Port QS Pin Assignment Register	PQSPAR	16
IPSBAR + 0x10_006E	Port TA Pin Assignment Register	PTAPAR	8
IPSBAR + 0x10_006F	Port TC Pin Assignment Register	PTCPAR	8
IPSBAR + 0x10_0070	Port TD Pin Assignment Register	PTDPAR	8
IPSBAR + 0x10_0071	Port UA Pin Assignment Register	PUAPAR	8
IPSBAR + 0x10_0072	Port UB Pin Assignment Register	PUBPAR	8
IPSBAR + 0x10_0073	Port UC Pin Assignment Register	PUCPAR	8
IPSBAR + 0x10_0074	Port DD Pin Assignment Register	PDDPAR	32
IPSBAR + 0x10_0075	Port LD Pin Assignment Register	PLDPAR	16
IPSBAR + 0x10_0076	Port GP Pin Assignment Register	PGPPAR	32
IPSBAR + 0x10_0077	Reserved	—	8
IPSBAR + 0x10_0078	Port Wired OR Control Register	PWOR	16
IPSBAR + 0x10_007A	Port Drive Strength Register 1	PDSR1	16
IPSBAR + 0x10_007C	Port Drive Strength Register 0	PDSR0	32
Reset Control, Chip Configuration, and Power Management Registers			
IPSBAR + 0x11_0000	Reset Control Register	RCR	8
IPSBAR + 0x11_0001	Reset Status Register	RSR	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x11_0004	Chip Configuration Register	CCR	16
IPSBAR + 0x11_0007	Low-Power Control Register	LPCR	8
IPSBAR + 0x11_0008	Reset Configuration Register	RCON	16
IPSBAR + 0x11_000A	Chip Identification Register	CIR	16
IPSBAR + 0x11_000C	Real Time Clock Divide Register	RTCDF	32
Clock Module Registers			
IPSBAR + 0x12_0000	Synthesizer Control Register	SYNCR	16
IPSBAR + 0x12_0002	Synthesizer Status Register	SYNSR	16
IPSBAR + 0x12_0007	Low Power Control Register	LPCR	16
IPSBAR + 0x12_0008	Clock Control High Register	CCHR	8
IPSBAR + 0x12_000C	Real Time Clock Divide Register	RTCDR	32
Edge Port Registers			
IPSBAR + 0x13_0000	EPORT0 Pin Assignment Register	EPPAR0	16
IPSBAR + 0x13_0002	EPORT0 Data Direction Register	EPDDR0	8
IPSBAR + 0x13_0003	EPORT0 Interrupt Enable Register	EPIER0	8
IPSBAR + 0x13_0004	EPORT0 Data Register	EPDR0	8
IPSBAR + 0x13_0005	EPORT0 Pin Data Register	EPPDR0	8
IPSBAR + 0x13_0006	EPORT0 Flag Register	EPFR0	8
IPSBAR + 0x14_0000	EPORT1 Pin Assignment Register	EPPAR1	16
IPSBAR + 0x14_0002	EPORT1 Data Direction Register	EPDDR1	8
IPSBAR + 0x14_0003	EPORT1 Interrupt Enable Register	EPIER1	8
IPSBAR + 0x14_0004	EPORT1 Data Register	EPDR1	8
IPSBAR + 0x14_0005	EPORT1 Pin Data Register	EPPDR1	8
IPSBAR + 0x14_0006	EPORT1 Flag Register	EPFR1	8
Programmable Interrupt Timer 0 Registers			
IPSBAR + 0x15_0000	PIT Control and Status Register 0	PCSR 0	16
IPSBAR + 0x15_0002	PIT Modulus Register 0	PMR 0	16
IPSBAR + 0x15_0004	PIT Count Register 0	PCNTR 0	16
Programmable Interrupt Timer 1 Registers			
IPSBAR + 0x16_0000	PIT Control and Status Register 1	PCSR 1	16
IPSBAR + 0x16_0002	PIT Modulus Register 1	PMR 1	16
IPSBAR + 0x16_0004	PIT Count Register 1	PCNTR 1	16

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
ADC Registers			
IPSBAR + 0x19_0000	Control Register 1	CTRL1	16
IPSBAR + 0x19_0002	Control Register 2	CTRL2	16
IPSBAR + 0x19_0004	Zero Crossing Control Register	ADZCC	16
IPSBAR + 0x19_0006	Channel List Register 1	ADLST1	16
IPSBAR + 0x19_0008	Channel List Register 2	ADLST2	16
IPSBAR + 0x19_000A	Sample Disable Register	ADSDIS	16
IPSBAR + 0x19_000C	Status Register	ADSTAT	16
IPSBAR + 0x19_000E	Limit Status Register	ADLSTAT	16
IPSBAR + 0x19_0010	Zero Crossing Status Register	ADZCSTAT	16
IPSBAR + 0x19_0012–0x19_0020	Result Registers 0-7	ADRSLT0–7	8x16
IPSBAR + 0x19_0022–0x19_0030	Low Limit Registers 0-7	ADLLMT0-7	8x16
IPSBAR + 0x19_0032–0x19_0040	High Limit Registers 0-7	ADHLMT0-7	8x16
IPSBAR + 0x19_0042–0x19_0050	Offset Registers 0-7	ADOFS0-7	8x16
IPSBAR + 0x19_0052	Power Control Register	POWER	16
IPSBAR + 0x19_0054	Voltage Reference Register	CAL	16
General Purpose Timer A Registers			
IPSBAR + 0x1A_0000	GPTA IC/OC Select Register	GPTAIOS	8
IPSBAR + 0x1A_0001	GPTA Compare Force Register	GPTACFORC	8
IPSBAR + 0x1A_0002	GPTA Output Compare 3 Mask Register	GPTAOC3M	8
IPSBAR + 0x1A_0003	GPTA Output Compare 3 Data Register	GPTAOC3D	8
IPSBAR + 0x1A_0004	GPTA Counter Register	GPTACNT	16
IPSBAR + 0x1A_0006	GPTA System Control Register 1	GPTASCR1	8
IPSBAR + 0x1A_0008	GPTA Toggle-on-Overflow Register	GPTATOV	8
IPSBAR + 0x1A_0009	GPTA Control Register 1	GPTACTL1	8
IPSBAR + 0x1A_000B	GPTA Control Register 2	GPTACTL2	8
IPSBAR + 0x1A_000C	GPTA Interrupt Enable Register	GPTAIE	8
IPSBAR + 0x1A_000D	GPTA System Control Register 2	GPTASCR2	8
IPSBAR + 0x1A_000E	GPTA Flag Register 1	GPTAFLG1	8
IPSBAR + 0x1A_000F	GPTA Flag Register 2	GPTAFLG2	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x1A_0010	GPTA Channel 0 Register	GPTAC0	16
IPSBAR + 0x1A_0012	GPTA Channel 1 Register	GPTAC1	16
IPSBAR + 0x1A_0014	GPTA Channel 2 Register	GPTAC2	16
IPSBAR + 0x1A_0016	GPTA Channel 3 Register	GPTAC3	16
IPSBAR + 0x1A_0018	Pulse Accumulator Control Register	GPTAPACTL	8
IPSBAR + 0x1A_0019	Pulse Accumulator Flag Register	GPTPAFLG	8
IPSBAR + 0x1A_001A	Pulse Accumulator Counter Register	GPTAPACNT	8
IPSBAR + 0x1A_001D	GPTA Port Data Register	GPTAPORT	8
IPSBAR + 0x1A_001E	GPTA Port Data Direction Register	GPTADDR	8
Pulse Width Modulator			
IPSBAR + 0x1B_0000	PWM Enable Register	PWME	8
IPSBAR + 0x1B_0001	PWM Polarity Register	PWMPOL	8
IPSBAR + 0x1B_0002	PWM Clock Select Register	PWMCLK	8
IPSBAR + 0x1B_0003	PWM Prescale Clock Select Register	PWMPRCLK	8
IPSBAR + 0x1B_0004	PWM Center Align Enable Register	PWMCAE	8
IPSBAR + 0x1B_0005	PWM Control Register	PWMCTL	8
IPSBAR + 0x1B_0008	PWM Scale A Register	PWMSCLA	8
IPSBAR + 0x1B_0009	PWM Scale B Register	PWMSCLB	8
IPSBAR + 0x1B_000C	PWM channel Counter Register 0	PWMCNT0	8
IPSBAR + 0x1B_000D	PWM channel Counter Register 1	PWMCNT1	8
IPSBAR + 0x1B_000E	PWM channel Counter Register 2	PWMCNT2	8
IPSBAR + 0x1B_000F	PWM channel Counter Register 3	PWMCNT3	8
IPSBAR + 0x1B_0010	PWM channel Counter Register 4	PWMCNT4	8
IPSBAR + 0x1B_0011	PWM channel Counter Register 5	PWMCNT5	8
IPSBAR + 0x1B_0012	PWM channel Counter Register 6	PWMCNT6	8
IPSBAR + 0x1B_0013	PWM channel Counter Register 7	PWMCNT7	8
IPSBAR + 0x1B_0014	PWM Channel Period Register 0	PWMPER0	8
IPSBAR + 0x1B_0015	PWM Channel Period Register 1	PWMPER1	8
IPSBAR + 0x1B_0016	PWM Channel Period Register 2	PWMPER2	8
IPSBAR + 0x1B_0017	PWM Channel Period Register 3	PWMPER3	8
IPSBAR + 0x1B_0018	PWM Channel Period Register 4	PWMPER4	8
IPSBAR + 0x1B_0019	PWM Channel Period Register 5	PWMPER5	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x1B_001A	PWM Channel Period Register 6	PWMPER6	8
IPSBAR + 0x1B_001B	PWM Channel Period Register 7	PWMPER7	8
FlexCAN Registers			
IPSBAR + 0x1C_0000	Module Configuration Register	CANMCR	16
IPSBAR + 0x1C_0004	FlexCAN Control Register	CANCTRL	32
IPSBAR + 0x1C_0008	Free Running TImer	TIMER	32
IPSBAR + 0x1C_000C	Reserved	—	32
IPSBAR + 0x1C_0010	Rx Global Mask	RXGMASK	32
IPSBAR + 0x1C_0014	Rx Buffer 14 Mask	RX14MASK	32
IPSBAR + 0x1C_0018	Rx Buffer 15 Mask	RX15MASK	32
IPSBAR + 0x1C_001C	Error Counter Register	ERRCNT	32
IPSBAR + 0x1C_0020	Error and Status	ERRSTAT	32
IPSBAR + 0x1C_0024	Reserved	—	32
IPSBAR + 0x1C_0028	Interrupt Mask Register	IMASK	32
IPSBAR + 0x1C_002C	Reserved	—	32
IPSBAR + 0x1C_0030	Interrupt Flag Register	IFLAG	32
IPSBAR + 0x1C_0080	Message Buffer 0 - Message Buffer 15	MBUFF0–MBUFF15	16x16bytes
Flash Registers			
IPSBAR + 0x1D_0000	CFM Configuration Register	CFMMCR	16
IPSBAR + 0x1D_0002	CFM Clock Divider Register	CFMCLKD	8
IPSBAR + 0x1D_0008	CFM Security Register	CFMSEC	32
IPSBAR + 0x1D_0010	CFM Protection Register	CFMPROT	32
IPSBAR + 0x1D_0014	CFM Supervisor Access Register	CFMSACC	32
IPSBAR + 0x1D_0018	CFM Data Access Register	CFMDACC	32
IPSBAR + 0x1D_0020	CFM User Status Register	CFMUSTAT	8
IPSBAR + 0x1D_0024	CFM Command Register	CFMCMD	8
IPSBAR + 0x1D_004A	CFM Clock Select Register	CFMCLKSEL	16
Ethernet Physical Transceiver Registers			
IPSBAR + 0x1E_0000	Ethernet Physical Transceiver Control Register 0	EPHYCTL0	8
IPSBAR + 0x1E_0001	Ethernet Physical Transceiver Control Register 1	EPHYCTL1	8
IPSBAR + 0x1E_0002	Ethernet Physical Transceiver Status Register	EPHYSR	8

Table A-3. Register Memory Map (continued)

Address	Name	Mnemonic	Size
IPSBAR + 0x1E_0003	Reserved	—	8
Random Number Generator H/W Accelerator Registers			
IPSBAR + 0x1F_0000	Random Number Generator Control Register	RNGCR	32
IPSBAR + 0x1F_0004	Random Number Generator Status Register 1	RNGSR	32
IPSBAR + 0x1F_0008	Random Number Generator Entropy Register	RNGER	32
IPSBAR + 0x1F_000C	Random Number Generator Output Register	RNGOUT	32-

[1] UMR1*n*, UMR2*n*, and UCSR*n* should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

Appendix B
Revision History

This appendix describes corrections to the *MCF52235 Reference Manual*. For convenience, the corrections are grouped by revision.

B.1 Changes between Rev. 6 and Rev. 7

Table 1. MCF52235RM Rev. 6 to Rev. 7 Changes

Location in Rev. 7	Description
Section 27.3.1	Updated QMR register with a NOTE
Section 7.7.1.4	Updated description of Clock Control High Register (CCHR)

B.2 Changes between Rev. 5 and Rev. 6

Table 2. MCF52235RM Rev. 5 to Rev. 6 Changes

Location in Rev. 5	Description
Fig 7-1 / Page 7-3	Updated Clock module block diagram.
Table 13-6 / Page 13-10	In the CWCR[CWRI] field description, changed “The interrupt level for the CWT is programmed in the interrupt control register 7 (ICR7)...” to “The interrupt level for the CWT is programmed in the interrupt control register 8 (ICR8)...”.
Fig 13-7 / Page 13-11	Added M3 Master block.
Fig 13-8, Table 13-7/ Page 13-13	Added M3_PRTY bit fields and description.
Section 28.1/ Page 28-1	Added note, “The MCF52235 does not have the 8 MHz internal oscillator feature and all references to it should be ignored and all control and status bits should also not be used.”
Section 28.5.2/ Page 28-25	Clarified ADC clock unit.

B.3 Changes between Rev. 4 and Rev. 5

Table 3. MCF52235RM Rev. 4 to Rev. 5 Changes

Location in Rev. 4	Description
Throughout	• Formatting, layout, spelling, and grammar corrections. • Removed the “Preliminary” designation from the title page and the page footers.
Chapter 1	Added information about the MCF52232 and MCF52236 devices.
Table 2-1 / Page 2-3	Synchronized the “Pin Functions by Primary and Alternate Purpose” table in the device reference manual and data sheet.
Figure 6-3 / Page 6-4	Updated the figure to show the OD bit (bit 31).

Table 3. MCF52235RM Rev. 4 to Rev. 5 Changes (continued)

Location in Rev. 4	Description
Section 7.6.2 / Page 7-5	Deleted the sentence “If CLKMOD0 is driven low during reset, XTAL is sampled to determine clocking mode.”
Table 7-4 / Page 7-7	Modified the SYNCR[RFD] description to explain that changing RFD may cause a glitch in the PLL clock.
Section 7.8.3 / Page 7-12	Deleted the sentence “The RFD is not in the feedback loop of the PLL, so changing the RFD divisor does not affect PLL operation”.
Table 12-5 / Page 12-5	• Added missing information to the RCON[RLOAD] and RCON[MODE] field descriptions. • Deleted the sentence “The default mode can be overridden during reset configuration” from the RCON[MODE] field description.
Table 13-1 / Page 13-2	Corrected PACR*n* addresses.
Section 13.5.4 / Page 13-8	Updated the section to reflect the fact that the CWT does not cause a hardware reset.
Table 13-12 / Page 13-18	Added an entry for PACR5 and a footnote to clarify the meaning of “—”.
Section 15-1 / Page 15-2	Deleted the sentence beginning with “For many peripheral devices...”.
Table 15-3 / Page 15-6	Deleted the entry for the (nonexistent) GSWIACK register.
Table 15-13 / Page 15-13	Added the missing EPHY interrupt source (36).
Section 15.3.8 / Page 15-19	Deleted references to the (nonexistent) GSWIACK register.
Chapter 17	Added the section “EzPort Lockout Recovery”.
Figure 17-3 / Page 17-5	Updated the FLASHBAR figure to show that WP is read-only with a reset value of 1.
Section 18.4.6 / Page 18-7	Added a subsection, “Duplicate Frame Transmission”.
Table 18-9 / Page 18-17	Corrected the ending address of the MIB block counters In the top-level module memory map (was IPSBAR+0x13FF, is IPSBAR+0x12FF).
Section 20.4 / Page 20-12	Deleted the sentence “BCR*n* decrements when an address transfer write completes for a single-address access (DCR*n*[SAA] = 0), or when SAA equals 1.”
Figure 26-6 / Page 26-9	Added a note to clarify the UCSR*n* reset values.
Figure 26-22 / Page 26-21	• Corrected the label of the top signal (was U*n*TXD, is U*n*RXD). • Corrected the text in the footnote (was TXRTS, is RXRTS).
Figure 26-23 / Page 23-24	Corrected the U*n*TXD label (was “Input”, is “Output”).
Figure 26-25 / Page 26-24	• Corrected a label on the bottom row (was UMR1*n*[PT]=2, is UMR1*n*[PT]=1). • Deleted duplicate UMR1*n*[PM]=11 label.
Section 26.4.6 / Page 26-26	• Reordered and renumbered the subsections. • Added example DMA configuration steps.
Table 28-19 / Page 28-20	Changed the description for SEL_VREFH=0 and SEL_VREFL=0 (were “Internal VRx”, are “VRH” and “VRL”, respectively).
Section 29.3.2.5.1 / Page 29-18	Added missing numerical values to the output example.
Section 29.3.2.6.1 / Page 29-20	Added missing numerical values to the output example.
Section 30.5.1 / Page 30-29	Corrected the section to reflect the fact that there are 19 total FlexCAN interrupts (added 16 individual interrupts per MB).
Table 31-4 / Page 31-6	Changed the reset values for PBR1, PBR2, and PBR3 (was “0x0000_0000”, is “See Section”).

Table 3. MCF52235RM Rev. 4 to Rev. 5 Changes (continued)

Location in Rev. 4	Description
Section 31.5.1 / Page 31-19	Combined second and third sentences in bullet #2 (was “This type of halt is always first made pending in the processor. Next, the processor samples for pending halt and interrupt conditions once per instruction”, is “This type of halt is always first marked as pending in the pocessor, which samples for pending halt and interrupt conditions once per instruction”.)
Appendix A	• Corrected PACR*n* addresses. • Deleted the entry for the (nonexistent) GSWIACK register.
End of document	Moved the revision history from the *MCF52235 Reference Manual Errata* document to Appendix B of this reference manual.

B.4 Changes between Rev. 3 and Rev. 4

Table 4. MCF52235RM Rev. 3 to Rev. 4 Changes

Location in Rev. 3	Description
Throughout	Formatting, layout, spelling, and grammar corrections.
Table 2-1 / Page 2-3	Changed the pin number for $\overline{\text{IRQ11}}$ on the 80 LQFP package (was “—”, is 41).
Table 3-1 / Page 3-3	• For the PC, changed the reset value (was “Undefined”, is “Contents of Location 0x0000_0004”) and the “Written with MOVEC entry” value (was “Yes”, is “No”). • Changed the reset value for the OTHER_A7 (was “Undefined”, is “Contents of Location 0x0000_0000”). • Changed the reset value for the RAMBAR (was “0x0000_0000”, is “See Section”).
Section 3.2.4 / Page 3-5	Deleted section and moved its information into Section 3.2.
Figure 3-5 / Page 3-6	• Modified the figure to show that Bits 4:0 are read/write. • Changed the access (was “Access: User read-only”, is “Access: User read/write”).
Table 3-2 / Page 3-6	Removed the last sentence in the C bit field description.
Figure 3-8 / Page 3-7	Modified the figure to show that Bits 4:0 are read/write.
Section 3.4 / Page 3-9	Changed the last sentence in step 2 to “The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address”.
Section 3.6.6 / Page 3-25	Added the following note after the table: The execution times for moving the contents of the Racc, Raccext[01,23], MACSR, or Rmask into a destination location <ea>x shown in this table represent the best-case scenario when the store instruction is executed and there are no load or M{S}AC instructions in the EMAC execution pipeline. In general, these store operations require only a single cycle for execution, but if preceded immediately by a load, MAC, or MSAC instruction, the depth of the EMAC pipeline is exposed and the execution time is four cycles.
Figure 4-4 / Page 4-5	Updated the figure to show that bits 11:0 are read-write.
Figure 4-5 / Page 4-10	Updated the MASK register figure to show that bits 31:16 are read-only and are always set.
Equation 4-3 / Page 4-13	Added a minus sign to the exponent so that it is “–(i + 1 – N)”.
Section 7.7.1.4 / Page 7-10	Added the following note: **Note:** The CCHR can be written at any time. However, changes will take effect only after the PLL is disabled and re-enabled.
Table 11-1 / Page 11-2	Changed the RAMBAR reset value (was “0x0000_0000”, is “See Section”).

Table 4. MCF52235RM Rev. 3 to Rev. 4 Changes (continued)

Location in Rev. 3	Description
Section 11.2.1 / Page 11-2	• Corrected section to show the proper RAMBAR figure and field description table. • Changed the last bullet to “A reset clears the RAMBAR’s priority, backdoor write-protect, and valid bits, and sets the backdoor enable bit. This enables the backdoor port and invalidates the processor port to the SRAM. (The RAMBAR must be initialized before the core can access the SRAM.) All other bits are unaffected.”.
Section 18.5.4.6 / Page 18-27	Added the following text to the MMFR description: “Before accessing the MII registers via the MMFR, the software must poll EIR[MII] to make sure that an access is not currently in progress.”
Figure 19-3 / Page 19-5	Corrected the reset value for ANDIS, DIS100, and DIS10 (was 0, is 1).
Figure 19-4 / Page 19-7	Corrected the name of bit 4 (was PHYADD4, is PHYADD3).
Figure 19-5 / Page 19-7	Corrected the reset value for 100DIS and 10DIS (was 0, is 1).
Section 19.3.3.2 / Page 19-11	• Corrected the name of bit 11 (was PDWN, is 10THD). • Corrected the description of bit 11 (the proper description is found in Revision 2 of the reference manual).
Figure 19-8 / Page 19-13	Corrected the reset value for bit 11 (was 01, is 0).
Section 19.3.3.4 / Page 19-13	• Corrected the reset value for PHYID (was 0b000000, is 0b000110). • Corrected the PHYID field description (was “Composed of bits 15:10.”, is “Composed of bits 19:24”).
Table 19-10 / Page 19-14	Added a description of the SELECTORFIELD field.
Section 19.3.3.6 / Page 19-15	• Updated the register figure and field description table to show that bits 12:11 are reserved. • Added a description of the SELECTORFIELD field.
Section 19.3.3.7 / Page 19-16	Provided a concise name for bits 10:0 (was “Message/Unformatted Code Field [10:0]”, is CODEFIELD).
Section 19.3.3.9 / Page 19-18	Provided a concise name for bits 10:0 (was “Message/Unformatted Code Field [10:0]”, is CODEFIELD).
Figure 19-16 / Page 19-20	• Updated the figure to show that the register is read-only. • Added the following footnote to ANCMODE: “This bit is valid only when ANNC is set.”
Figure 19-17 / Page 19-21	• Corrected the reset value for FEFLTD (was 1, is 0). • Corrected the reset value for bit 12 (was 1, is 0). • Corrected the reset value for bit 11 (was 0, is 1). • Corrected the reset value for JBDE (was 0, is 1). • Corrected the reset value for POLCORD [was “(1)”, is 0].
Table 19-17 / Page 19-21	• Corrected the name of bit 13 (was MIILBO, is MIILBD). • Corrected the description of bit 12 (is “Reserved, should be cleared.”) • Corrected the description of bit 11 (is “Reserved, should be set.”)
Figure 25-1 / Page 25-1	Corrected signal name (was QSPI_CS[:0], is QSPI_CS[3:0]).
Section 26.2 / Page 26-3	Changed "An internal interrupt request signal notifies the interrupt controller..." to "A request signal is provided to notify the interrupt controller...".
Table 26-6 / Page 26-9	Changed “DTIN” to “DT*n*IN” (to maintain consistent signal names throughout chapter).
Section 26.4.5.2 / Page 26-26	Changed "...complete normally without exception processing..." to "...complete normally without an error termination...".
Section 27.6 / Page 27-12	Changed programming examples from assembly language to pseudocode.
Table 29-1 / Page 29-2	Deleted reference to nonexistent SCMISR register from footnote 2.

Table 4. MCF52235RM Rev. 3 to Rev. 4 Changes (continued)

Location in Rev. 3	Description
Table 31-10 / Page 31-16	• Added the following note to the PBR0[Address] field description: **Note:** PBR0[0] should always be loaded with a 0. • Changed the bit range in the Field column (was 31–1, is 31–0).
Figure 31-8 / Page 31-16	Changed the address of PBR3 (was 0x1C, is 0x1B).
Table 31-22 / Page 31-39	Changed the initial state of the CSR (was 0x0, is 0x0090_0000).
Section 31.6.2 / Page 31-43	Added the following note at the end of this section: The debug module requires the use of the internal bus to perform BDM commands. For this processor core, if the processor is executing a tight loop that is contained within a single aligned longword, the processor may never grant the internal bus to the debug module, for example: align4 label1: nop bra.b label1 or align4 label2: bra.w label2 The processor grants the internal bus if these loops are forced across two longwords.
Figure 32-2 / Page 32-4	Updated the IDCODE register figure to indicate that the reset values for both PRN and PIN are device-dependent.
Appendix A	Deleted entries for nonexistent CACR, ACR0, and ACR1 registers.

B.5 Changes between Rev. 2 and Rev. 3

Table 5. MCF52235RM Rev. 2 to Rev. 3 Changes

Location in Rev. 2	Description
Throughout	Formatting, layout, spelling, and grammar corrections.
Section 6.5.1.3 / Page 6-5	Changed field name from “External entropy” to “ENT”.
Section 6.5.1.4 / Page 6-6	Changed field name from “Random output” to “RANDOM_OUTPUT”.
Chapter 7	Added field description tables to Sections 7.7.1.3, 7.7.1.4, and 7.7.1.5.
Section 8.4.2 / Page 8-13	Replaced erroneous sample assembly code for RTC initialization with valid C code.
Table 9-2 / Page 9-2	Deleted superfluous table.
Table 12-6 / Page 12-5	Added missing part identification number for the MCF52231.
Figure 13-5 / Page 13-9	Corrected name of bit 0 (was CWTIC, is CWTIF).
Chapter 15	Added missing information on GSWIACK and GL*m*IACK registers.
Section 11.1.2 / Page 11-1	Removed text “...within the 256-MByte address space (0x8000_0000-0x8FFF_FFFF)”.
Table 11-2 / Page 11-2	Removed text “...within the processor’s 256-MByte address space...” and “For proper operation, the base address must be set to between 0x8000_0000 and 0x8FFF-8C000.”.
Chapter 14	• Changed naming convention for the port pin data/set data registers: Was: PORT*n*P/SET*n* (e.g., PORTNQP/SETNQ) Is: SET*n* (e.g., SETNQ) • Changed naming convention for the bits in the port pin data/set data registers: Was: PORT*n*P*x* (e.g., PORTNQP6) Is: SET*nx* (e.g., SETNQP6)

Table 5. MCF52235RM Rev. 2 to Rev. 3 Changes (continued)

Location in Rev. 2	Description
Section 14.6.5.4 / Page 14-14	Changed PDSR definition (was PDSR [48 bits], is PDSR0 [32 bits] and PDSR1 [16 bits]).
Figure 16-3 / Page 16-4	Replaced register figure with correct 8-bit version.
Section 17.3.2 / Page 17-4	Deleted erroneous reference to external boot mode.
Table 18-11 / Page 18-19	Added RMON_R_DROP counter.
Section 18.5.4.5 / Page 18-25	• Added missing ECR register figure. • Corrected cross-reference in the note of Table 18-16.
Section 18.5.4.23 / Page 18-41	Corrected IPSBAR offset of EMRBR (was 0x11B8, is 0x1188).
Chapter 19	• Reorganized information throughout entire chapter. • Updated register addresses to include proper IPSBAR offsets. • Converted register field descriptions to SRS format. • Corrected register mnemonics as necessary to ensure consistent register naming. • Numerous grammar and stylistic corrections.
Table 20-4 / Page 20-8	Deleted erroneous reference to nonexistent AT bit.
Chapter 22	Deleted erroneous references to nonexistent PIT2 and PIT3 modules.
Section 23.6.13 / Page 23-12	Deleted reference to nonexistent CF bits in the figure and bit descriptions for the GPTFLG2 register.
Chapter 24	Updated register figures and tables to include correct register addresses.
Chapter 25	Added missing equations in Section 25.4.
Chapter 27	Updated register figures and tables to include correct register addresses.
Table 30-1 / Page 30-5	Corrected RXGMASK address (was 0xC_0010, is 0x1C_0010).
Section 30.3.1/ Page 30-6	Added missing illustration of BIts 15:0 in the CANMCR figure and updated field description table accordingly.
Section 30.3.7 / Page 30-15	Added missing IMASK register figure and updated field description table accordingly.
Section 30.3.8 / Page 30-15	Added missing IFLAG register figure and updated field description table accordingly.
Figure 30-9 / Page 30-13	Corrected register mnemonic (was CANCTRL, is ERRSTAT).
Appendix A	• Added GSWIACK register. • Removed trailing R from the names of the global level m IACK registers. • Updated PDSR register names (PDSR0 at IPSBAR+0x10_007C, PDSR1 at IPSBAR+0x10_007A). • Added FEC registers. • Renamed GPIO port pin data/set data registers using the new naming convention (see entry for Chapter 14).

B.6 Changes between Rev. 1 and Rev. 2

Table 6. MCF52235RM Rev. 1 to Rev. 2 Changes

Location in Rev. 1	Description
Throughout	Language, punctuation, and layout improvements.
Title page	Added “This product incorporates SuperFlash® technology licensed from SST” statement.

Table 6. MCF52235RM Rev. 1 to Rev. 2 Changes (continued)

Location in Rev. 1	Description
Chapter 1	• Corrected missing and incomplete sentences. • Updated block diagram to include correct peripheral signal names. • Revised package information.
Section 1.4.3 / Page 1-10	Corrected second sentence to read "... a 256-bit boundary-scan register...".
Table 2-1 / Section 2.2	• Set table caption to repeat on every page. • Changed footnote 11 to "VDD1, VDD2, VDDPLL and PHY_VDD pins are for decoupling only, and should NOT have power directly applied to them." and corrected references so that only pins VDDPLL, PHY_VDDA, PHY_VDDRX, PHY_VDDTX, and VDD reference this footnote. • Corrected pin numbers as follows: QSPI_CS0, 80 LQFP package: should be 28 instead of 58 FlexCAN SYNCA, 112 LQFP package: should be 28 instead of — FlexCAN SYNCA, 80 LQFP package: should be 20 instead of — FlexCAN SYNCB, 112 LQFP package: should be 27 instead of — FlexCAN SYNCB, 80 LQFP package: should be 19 instead of — VSSX, 121MAPBGA package: should be — instead of an empty cell
Section 3.2.11 / Page 3-8	Added cross-reference to FLASHBAR register.
Chapter 6	• Added register and bit acronyms. • Removed references to FIFO-based functionality for RNGOUT. • Formatted register figures and descriptions in accordance with manual conventions.
Chapter 7	Replaced "Low-Power Divider Register (LPDR)" with "Low-Power Control Register (LPCR)" and corrected its address to match the rest of the document.
Chapter 8	Replaced all register addresses with correct values and updated several register names.
Table 8-1 / Page 8-3	Corrected register names in memory map table - changed RCCTL to RTCCTL, DAYS to DAYR, ALARM_DAY to DAY_ALARM.
Section 8.2.1 / Page 8-3	Updated explanation of reset condition (POR resets RTC) in HOURMIN and SECONDS register descriptions
Section 8.2.1.5 / Page 8-7	Deleted extraneous XTL bit description in RTCCTL register description.
Section 8.3.3 / Page 8-12	Deleted extraneous sentence "For example, to turn off the LCD controller..." in Minute Stopwatch description.
Section 11.1.1 / Page 11-1	Corrected SRAM size.
Chapter 14	• Corrected register addresses to include proper offset (IPSBAR+0x10_....). • Updated register figures to SRS standards. • Corrected register figure titles and added field description tables. • In section 14.6.5, changed "If multiple pins are configured for the one function, then the rsult is undefined" to "Some signals can be assigned to different pins (see Table 2-1). However, a signal should not be assigned to more than one pin at the same time. If a signal is assigned to two or more pins simultaneously, the result is undefined."
Figure 14-1 / Page 14-2	Revised signal names to match the names in Chapter 2.
Section 15.1 / Page 15-2	Added cross-reference to exception vector assignments table in the ColdFire Core chapter.
Section 15.1.1.3 / Page 15-3	Updated exception vector mapping instructions and added cross-reference to exception vector assignments table in the ColdFire Core chapter.
Table 15-2 / Page 15-4	• Changed the first interrupt controller number from INTC to INTC0. • Added abbreviation ICBA (Interrupt Controller Base Address) to base address column.

Table 6. MCF52235RM Rev. 1 to Rev. 2 Changes (continued)

Location in Rev. 1	Description
Table 15-3 / Page 15-5	• Replaced references to IPSBAR with proper reference to ICBA in module offset column. • Added *n* to appropriate register names.
Figure 15-2 / Page 15-6	• Changed register name from IPSBMT to IPRL*n*. • Changed field label from INT[16:1] to INT[15:1].
Figure 15-4 / Page 15-8	Changed field label from INT_MASK[16:1] to INT_MASK[15:1].
Figure 15-6 / Page 15-9	Changed field label from INTFRCL[16:1] to INTFRCL[15:1].
Section 15.3	Replaced references to IPSBAR with proper reference to ICBA in register figures.
Section 15.3.3 / Page 15-10	Removed duplicate INTFRCL*n* register figure (Figure 15-7).
Table 15-14 / Page 15-15	Corrected FlexCAN section such that source descriptions and flag clearing mechanisms match the corresponding flags.
Section 15.3.7 / Page 15-17	Changed the placeholder letter in the register name from "*n*" to "*m*" (i.e., L*m*IACK).
Chapter 17	• Added clarifying text and reference to Table 17-1 to footnote of CFMSEC, CFMPROT, CFMSACC, and CFMDACC registers. • Added missing titles to register field description tables. • Changed prefix for hexadecimal numbers from $ to 0x.Updated register addresses to include correct IPSBAR offsets.
Section 17.3 / Page 17-4	Added definition and description of FLASHBAR register.
Chapter 20	• Added missing registers to Table 20-1. • Updated register figures to include proper register names and addresses. • Combined Sections 20.3.4 and 20.3.4.1, and revised text to clarify the structure of the BCR*n* and DSR*n* registers. • Added missing figure and bit descriptions for the DCR*n* registers.
Section 21.6 / Page 21-8	• Added cross-reference to CFMCLKD register. • Changed "f_{SYS}" to "$f_{SYS/2}$". • Updated values and examples to reflect the 60 MHz system clock. • Added clarifying text to example for calculating FCLK.
Chapter 22	Deleted references to nonexistent PIT2 and PIT3 modules.
Chapter 24	Changed signal names DT*n*IN to DTIN*n* and DT*n*OUT to DTOUT*n* to match the convention used in the rest of the document.
Section 24.1.2 / Page 24-2	• Changed maximum timeout period from 266,521 seconds (~74 hours) to 293,203 s (~81 hours) and related frequency from 66 MHz to 60 MHz. • Changed resolution from 15 ns to 17 ns and related frequency from 66 MHz to 60 MHz.
Section 24.4.2 / Page 24-10	Changed example frequency from 66 MHz to 60 MHz and the result of Equation 24-2 from 2.00 seconds to 2.20 seconds.
Section 25.1.3 / Page 25-2	• Changed minimum baud rate from 129.4 Kbps to 117.6 Kbps. • Changed maximum baud rate from 16.6 Mbps to 15 Mbps. • Changed frequency from 66 MHz to 60 MHz.
Table 25-8 / Page 25-14	• Changed internal bus clock speed from 66 MHz to 60 MHz. • Changed QSPI_CLK values to match the correct 60 MHz clock speed per Equation 25-1 (15 MHz, 7.5 MHz, 3.75 MHz, 1.88 MHz, 937.5 kHz, and 117.6 kHz for QMR = 2, 4, 8, 16, 32, and 255, respectively).
Section 25.5 / Page 25-16	Changed step 1 from "... a QSPI_CLK frequency of 4.125 MHz (assuming a 66-MHz..." to "...a QSPI_CLK frequency of 3.75 MHz (assuming a 60-MHz...".

Table 6. MCF52235RM Rev. 1 to Rev. 2 Changes (continued)

Location in Rev. 1	Description
Chapter 26	Changed signal names to match the convention used in the rest of the document (DT*n*IN to DTIN*n*, DT*n*OUT to DTOUT*n*, U*n*$\overline{\text{RTS}}$ to $\overline{\text{URTS}}$*n*, U*n*$\overline{\text{CTS}}$ to $\overline{\text{UCTS}}$*n*, U*n*RXD to URXD*n*, U*n*TXD to UTXD*n*).
Section 26.4.1.2.1 / Page 26-19	• Changed numerator in Equation 26-1 from $f_{sys/2}$ to f_{sys}. • Changed values in Equation 26.2 to reflect a 60-MHz clock.
Chapter 28	Deleted superfluous Table 28-5.
Section 28.5.7 / Page 28-32	Deleted extraneous sentence at end of first paragraph.
Figure 28-15 / Page 28-26	Changed address of CTRL2 register from IPSBAR+0x19_0001 to IPSBAR+0x19_0002.
Table 31-5 / Page 31-8	Added bit description for the BKD bit in the Configuration/Status Register (CSR).
Section 32.4.3 / Page 32-7	Added missing Table 32-5 (JTAG instructions).
Table A-3	• Corrected spelling of IPSBAR for the ICR034, ICR134, and GPTACFORC registers. • Added missing registers CFMCLKSEL, ICR016, and ICR116. • Added leading zeros to addresses as necessary to adhere to four-digit address convention. • Corrected address of the PPMRH, PPMRL, and GPTAPACNT registers. • Corrected addresses of the DMA controller module registers to match the values in the text. • Corrected several GPIO register names to match the memory map. • Corrected several real-time clock register names. • Updated ADC register entries to show correct register names, addresses, and bit sizes.

B.7 Changes between Rev. 0 and Rev. 1

Table 7. MCF52235RM Rev. 0 to Rev. 1 Changes

Location in Rev. 0	Description
Throughout	Corrected various spelling, grammar, style, cross-reference, and layout errors.
Table 2-1 / Section 2.2	• Added pin assignments to 121MAPBGA packaging. • Added footnote describing limited functionality when using external PHY. • Corrected various pin assignments and functions. • Deleted duplicate CANTX and CANRX footnote.
Chapter 7	Removed references to 1:1 PLL mode, as it is not available on MCF521x and MCF522xx parts.
Figure 7-1 / Page 7-3	Added PLL pre-divider block.
Table 7-3 / Page 7-5	Added register name (CCHR) to Clock Control High Register and changed reset value from 0x00 to 0x04.
Table 7-4 / Page 7-7	In the description for MFD, changed footnote 1 to include correct equations and values.
Section 7.7.1.4 / Page 7-10	Changed register name from PFD to CCHR and changed reset value from 0b000 to 0b100.
Section 7.8.2 / Page 7-11	Added text to first sentence to: “... reference frequency (i.e., clock frequency divided by the pre-division factor specified by CCHR)...”
Table 8-1 / Section 8.2	Corrected first column to display the proper IPSBAR offset.
Chapter 8	Corrected the addresses in the register figures to show the proper IPSBAR address instead of $BASE_ADDRESS address.
Fig. 8-13 / Page 8-14	Deleted invalid reference to ARM instruction code segment.

Table 7. MCF52235RM Rev. 0 to Rev. 1 Changes (continued)

Location in Rev. 0	Description
Table 9-3/ Page 9-3	• Corrected field order - Bit 13 is CDRNGA, Bit 12 is CDEPHY. • In description for CDGPT, replaced “ICOC” with “GPT”.
Section 9.2.4.1 / Page 9-9	Corrected LPCR figure and table - STPMD is a 2-bit field (bits 4 and 3), and bit 1 is LVDSE.
Chapter 11	Corrected conditional text entries to ensure proper display of memory sizes.
Table 11-2 / Page 11-3	Filled in table in PRIU/PRIL field description.
Figure 14-1 / Page 14-2	Added FEC signals and arranged signals in same order as in Table 2-1.
Table 14-1 / Page 14-5	Corrected register name at offset $007C to “PDSR” instead of “PDRR”.
Section 14.6.5.4 / Page 14-14	Added clear references to footnotes of Figures 14-25, 14-26, and 14-27.
Section 18.5.4.5 / Page 18-25	Added note about loss of functionality when using external PHY.
Section 18.5.4.7 / Page 18-28	Changed formula in paragraph above Table 18-19 to “... 1/(2*5)” instead of “... 1/10” to match layout of equation in Table 18-18 better.
Table 18-19 / Page 18-29	Changed system clock frequency from 66 MHz to 60 MHz to reflect highest possible clock frequency.
Section 19.2.9 / Page 19-4	Replaced “Flashes in half-duplex mode when a collision occurs on the network...” with “Flashes when a collision occurs on a network in half duplex mode...”.
Section 20.4	Removed duplicate register figures.
Section 27.6.1 / Page 27-12	Added missing line of code to note: `I2CR = 0x80 ; re-enable`
Section 27.6.2 / Page 27-13	Replaced instances of MBB with IBB.
Section 29.2	Replaced “Address” with “IPSBAR Offset” in the register figures.