

---

# MCF51AG128 ColdFire® Integrated Microcontroller Reference Manual

**Devices Supported:**  
MCF51AG128  
MCF51AG96

Document Number: MCF51AG128RM  
Rev. 5  
7/2010

## How to Reach Us:

### Home Page:

<http://www.freescale.com>

### E-mail:

support@freescale.com

### USA/Europe or Locations Not Listed:

Freescale Semiconductor  
Technical Information Center, CH370  
1300 N. Alma School Road  
Chandler, Arizona 85224  
+1-800-521-6274 or +1-480-768-2130  
support@freescale.com

### 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)  
support@freescale.com

### 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 303-675-2140  
Fax: 303-675-2150  
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2010. All rights reserved.

MCF51AG128RM  
Rev. 5  
7/2010

## Chapter 1 Device Overview

1.1	The MCF51AG128 Series Microcontrollers	1-1
1.1.1	Definition	1-1
1.1.2	MCF51AG128 Series Package Availability	1-1
1.1.3	MCF51AG128 Series Device Comparison	1-1
1.2	MCF51AG128 Series Block Diagram	1-3
1.2.1	Block Diagram	1-3
1.2.2	Functional Units	1-4
1.2.3	Module Versions	1-6
1.3	V1 ColdFire Core	1-7
1.3.1	User Programming Model	1-7
1.3.2	Supervisor Programming Model	1-7
1.4	System Clock Generation and Distribution	1-8
1.4.1	Clock Distribution Diagram	1-8
1.4.2	System Clocks	1-9

## Chapter 2 Pins and Connections

2.1	Device Pin Assignment	2-1
2.1.1	Pinout: 80-Pin LQFP	2-1
2.1.2	Pinout: 64-Pin LQFP and QFP	2-3
2.1.3	Pinout: 48-Pin LQFP	2-3
2.1.4	Recommended System Connections	2-7
2.1.5	Power (VDD, VSS, VDDA, and VSSA)	2-9
2.1.6	Oscillator (XTAL and EXTAL)	2-9
2.1.7	$\overline{\text{RESET}}$	2-10
2.1.8	$\overline{\text{IRQ/TPMCLK}}$	2-10
2.1.9	Background/Mode Select (BKGD/MS)	2-10
2.1.10	ADC Reference Pins ( $V_{\text{REFH}}$ , $V_{\text{REFL}}$ )	2-11
2.1.11	General-Purpose I/O and Peripheral Ports	2-11

## Chapter 3 Modes of Operation

3.1	Introduction	3-1
3.2	Summary of Modes	3-1
3.3	Overview	3-2
3.4	Debug Mode	3-3
3.5	Secure Mode	3-4
3.6	Run Mode	3-4
3.7	Wait Mode	3-4
3.8	Stop Modes	3-4
3.8.1	Stop2 Mode	3-5
3.8.2	Stop3 Mode	3-6

3.8.3	Stop4 Mode	3-6
3.8.3.1	LVD Enabled in Stop Mode	3-6
3.9	On-Chip Peripheral Modules in Stop and Wait Modes	3-7

## Chapter 4 Memory

4.1	MCF51AG128 Series Memory Map	4-1
4.2	Detailed Register Addresses and Bit Assignments	4-2
4.2.1	Peripherals I/O and Control Registers	4-3
4.2.2	Intelligent Event Controller (iEvent) Memory Map	4-15
4.2.3	Flash Module Reserved Memory Locations	4-17
4.2.4	ColdFire Rapid GPIO Memory Map	4-18
4.2.5	ColdFire Interrupt Controller Memory Map	4-19
4.2.6	DMA Controller Memory Map	4-20
4.3	RAM	4-22
4.4	Flash Memory	4-22
4.4.1	Features	4-23
4.4.2	Register Descriptions	4-23
4.4.2.1	Flash Clock Divider Register (FCDIV)	4-23
4.4.2.2	Flash Options Register (FOPT and NVOPT)	4-24
4.4.2.3	Flash Configuration Register (FCNFG)	4-25
4.4.2.4	Flash Protection Register (FPROT and NVPROT)	4-25
4.4.2.5	Flash Status Register (FSTAT)	4-27
4.4.2.6	Flash Command Register (FCMD)	4-28
4.4.3	Flash Command Operations	4-29
4.4.3.1	Writing the FCDIV Register	4-29
4.4.3.2	Command Write Sequence	4-31
4.4.4	Flash Commands	4-31
4.4.4.1	Erase Verify Command	4-31
4.4.4.2	Program Command	4-33
4.4.4.3	Burst Program Command	4-35
4.4.4.4	Sector Erase Command	4-37
4.4.4.5	Mass Erase Command	4-38
4.4.5	Illegal Flash Operations	4-40
4.4.5.1	Flash Access Violations	4-40
4.4.5.2	Flash Protection Violations	4-40
4.4.6	Operating Modes	4-41
4.4.6.1	WAIT Mode	4-41
4.4.6.2	Stop Mode	4-41
4.4.6.3	Background Debug Mode	4-41
4.4.7	Security	4-41
4.4.7.1	Unsecuring the MCU using Backdoor Key Access	4-42
4.4.8	Resets	4-43
4.4.8.1	Flash Reset Sequence	4-43
4.4.8.2	Reset While Flash Command Active	4-43



4.4.8.3	Program and Erase Times	4-43
4.5	Security	4-43

## Chapter 5

### Resets, Interrupts, and General System Control

5.1	Introduction	5-1
5.2	Features	5-1
5.3	Microcontroller Reset	5-1
5.3.1	Illegal Opcode Detect (ILOP)	5-2
5.3.2	Illegal Address Detect (ILAD)	5-2
5.3.3	Watchdog Timer (WDOG)	5-2
5.3.4	Loss of Clock Reset (LOC)	5-2
5.4	Interrupts and Exceptions	5-2
5.4.1	External Interrupt Request (IRQ) Pin	5-2
5.4.1.1	Pin Configuration Options	5-3
5.4.2	Interrupt Vectors, Sources, and Local Masks	5-3
5.5	Low-Voltage Detect (LVD) System	5-6
5.5.1	Power-On Reset Operation	5-6
5.5.2	LVD Reset Operation	5-7
5.5.3	Low-Voltage Warning (LVW) Interrupt Operation	5-7
5.6	MCLK Output	5-7
5.7	Peripheral Clock Gating	5-7
5.8	Reset, Interrupt, and System Control Registers and Control Bits	5-8
5.8.1	Interrupt Pin Request Status and Control Register (IRQSC)	5-8
5.8.2	System Reset Status Register (SRS)	5-9
5.8.3	System Options (SOPT1) Register	5-10
5.8.4	System MCLK Control Register (SMCLK)	5-11
5.8.5	System Device Identification Register (SDIDH, SDIDL)	5-11
5.8.6	System Power Management Status and Control 1 Register (SPMSC1)	5-12
5.8.7	System Power Management Status and Control 2 Register (SPMSC2)	5-13
5.8.8	System Options 2 (SOPT2) Register	5-15
5.8.9	System Clock Gating Control 1 Register (SCGC1)	5-16
5.8.10	System Clock Gating Control 2 Register (SCGC2)	5-17
5.8.11	System Clock Gating Control 3 Register (SCGC3)	5-18
5.8.12	System eGPIO Interrupt Status 1 (SEIS1) Register	5-19
5.8.13	System eGPIO Interrupt Status 2 (SEIS2) Register	5-19
5.8.14	System ADC Pin Enable 1 Register (SAPE1)	5-20
5.8.15	System ADC Pin Enable 2 Register (SAPE2)	5-21
5.8.16	System ADC Pin Enable 3 Register (SAPE3)	5-22
5.8.17	System Pin Positioning Register (SPINPS)	5-23
5.8.18	System PortA Input Buffer Enable Register (SIMPTA)	5-23
5.8.19	System PortB Input Buffer Enable Register (SIMPTB)	5-24
5.8.20	System PortC Input Buffer Enable Register (SIMPTC)	5-25
5.8.21	System PortD Input Buffer Enable Register (SIMPTD)	5-26
5.8.22	System PortE Input Buffer Enable Register (SIMPTE)	5-27

5.8.23	System PortF Input Buffer Enable Register (SIMPTF)	5-28
5.8.24	System PortG Input Buffer Enable Register (SIMPTG)	5-29
5.8.25	System PortH Input Buffer Enable Register (SIMPTH)	5-30
5.8.26	System PortJ Input Buffer Enable Register (SIMPTJ)	5-31

## Chapter 6

### Parallel Input/Output Control

6.1	Overview	6-2
6.1.1	Features	6-2
6.1.2	Modes of Operation	6-3
6.1.2.1	Operation in Wait Mode	6-3
6.1.2.2	Operation in Stop Mode	6-3
6.1.2.3	Operation in Active Background Mode	6-3
6.2	Memory Map and Registers	6-3
6.2.1	Overview	6-3
6.2.2	Module Memory Map	6-3
6.2.3	Register Descriptions	6-4
6.2.3.1	Port Data Registers (PTxD)	6-5
6.2.3.2	Data Direction Registers (PTxDD)	6-5
6.2.3.3	PORTx Pin Value Register (PTxPV)	6-6
6.2.3.4	PORTx Pulling Enable Register (PTxPUE)	6-6
6.2.3.5	PORTx Pullup/Pulldown Select Register (PTxPUS)	6-7
6.2.3.6	PORTx Drive Strength Control Register (PTxDS)	6-7
6.2.3.7	PORTx Slew Rate Enable Register (PTxSRE)	6-8
6.2.3.8	PORTx Passive Filter Enable Register (PTxPFE)	6-8
6.2.3.9	PORTx Interrupt Control Register (PTxIC)	6-8
6.2.3.10	PORTx Interrupt Pin Enable Register (PTxIPE)	6-9
6.2.3.11	PORTx Interrupt Flag Register (PTxIF)	6-9
6.2.3.12	PORTx Interrupt Edge Select Register (PTxIES)	6-10
6.2.3.13	PORTx Digital Filter Enable Register (PTxDFE)	6-10
6.2.3.14	PORTx Digital Filter Control Register (PTxDFC)	6-11
6.3	Functional Description	6-12
6.3.1	Port Data Logic	6-12
6.3.1.1	Operation When EGPIO Controls Pin	6-13
6.3.1.2	Operation When Another On-Chip Module Controls Pin	6-13
6.3.1.3	Pin Value Register	6-13
6.3.2	Port Control	6-14
6.3.3	Pin Interrupt	6-14
6.3.3.1	Edge Only Sensitivity	6-15
6.3.3.2	Edge and Level Sensitivity	6-16
6.3.3.3	Control of Pullup/Pulldown Resistors	6-16
6.3.3.4	Asynchronous Interrupt in Stop Mode	6-16
6.3.3.5	Pin Interrupt Initialization	6-17
6.3.4	Digital Filters	6-17
6.3.4.1	Initialization of Digital Filters	6-17

## Chapter 7 ColdFire Core

7.1	Introduction	7-1
7.1.1	Overview	7-1
7.2	Memory Map/Register Description	7-2
7.2.1	Data Registers (D0–D7)	7-3
7.2.2	Address Registers (A0–A6)	7-4
7.2.3	Supervisor/User Stack Pointers (A7 and OTHER_A7)	7-4
7.2.4	Condition Code Register (CCR)	7-5
7.2.5	Program Counter (PC)	7-6
7.2.6	Vector Base Register (VBR)	7-6
7.2.7	CPU Configuration Register (CPUCR)	7-7
7.2.8	Status Register (SR)	7-8
7.3	Functional Description	7-9
7.3.1	Instruction Set Architecture (ISA_C)	7-9
7.3.2	Exception Processing Overview	7-10
7.3.2.1	Exception Stack Frame Definition	7-12
7.3.3	Processor Exceptions	7-14
7.3.3.1	Access Error Exception	7-14
7.3.3.2	Address Error Exception	7-14
7.3.3.3	Illegal Instruction Exception	7-15
7.3.3.4	Privilege Violation	7-16
7.3.3.5	Trace Exception	7-16
7.3.3.6	Unimplemented Line-A Opcode	7-17
7.3.3.7	Unimplemented Line-F Opcode	7-17
7.3.3.8	Debug Interrupt	7-17
7.3.3.9	RTE and Format Error Exception	7-17
7.3.3.10	TRAP Instruction Exception	7-18
7.3.3.11	Unsupported Instruction Exception	7-18
7.3.3.12	Interrupt Exception	7-18
7.3.3.13	Fault-on-Fault Halt	7-18
7.3.3.14	Reset Exception	7-19
7.3.4	Instruction Execution Timing	7-22
7.3.4.1	Timing Assumptions	7-22
7.3.4.2	MOVE Instruction Execution Times	7-23
7.3.4.3	Standard One Operand Instruction Execution Times	7-24
7.3.4.4	Standard Two Operand Instruction Execution Times	7-25
7.3.4.5	Miscellaneous Instruction Execution Times	7-26
7.3.4.6	Branch Instruction Execution Times	7-27

## Chapter 8 Interrupt Controller (CF1\_INTC)

8.1	Introduction	8-1
8.1.1	Overview	8-2

8.1.2	Features .....	8-6
8.1.3	Modes of Operation .....	8-7
8.2	External Signal Description .....	8-7
8.3	Memory Map/Register Definition .....	8-7
8.3.1	Interrupt Mask Registers (INTC_IMR{H,L}) .....	8-8
8.3.2	Force Interrupt Register (INTC_FRC) .....	8-10
8.3.3	INTC Programmable Level 6, Priority {7,6} Registers (INTC_PL6P{7,6}) .....	8-11
8.3.4	INTC Wakeup Control Register (INTC_WCR) .....	8-12
8.3.5	Set Interrupt Mask Register (INTC_SIMR) .....	8-13
8.3.6	Clear Interrupt Mask Register (INTC_CIMR) .....	8-13
8.3.7	INTC Set Interrupt Force Register (INTC_SFRC) .....	8-14
8.3.8	INTC Clear Interrupt Force Register (INTC_CFRC) .....	8-15
8.3.9	INTC Software and Level- <i>n</i> IACK Registers ( <i>n</i> = 1,2,3,...,7) .....	8-16
8.3.10	Interrupt Request Level and Priority Assignments .....	8-17
8.4	Functional Description .....	8-20
8.4.1	Handling of Non-Maskable Level 7 Interrupt Requests .....	8-20
8.5	Initialization Information .....	8-20
8.6	Application Information .....	8-21
8.6.1	Emulation of the HCS08's 1-Level IRQ Handling .....	8-21
8.6.2	Using INTC_PL6P{7,6} Registers .....	8-21
8.6.3	More on Software IACKs .....	8-22

## Chapter 9

### High-Speed Comparator (S08HSCMPV2)

9.1	Introduction .....	9-1
9.2	Low Power Mode Operation .....	9-1
9.3	HSCMP Configuration Information .....	9-1
9.3.1	HSCMP Analog Input Configuration .....	9-1
9.3.2	TPM/HSCMP Configuration Information .....	9-2
9.3.3	HSCMP Clock Gating .....	9-2
9.4	Features .....	9-3
9.5	Block Diagram .....	9-3
9.6	Pin Descriptions .....	9-5
9.6.1	External Pins .....	9-5
9.7	Functional Description .....	9-5
9.7.1	HSCMP Functional Modes .....	9-6
9.7.1.1	Disabled Mode (# 1) .....	9-7
9.7.1.2	Continuous Mode (#s 2A & 2B) .....	9-8
9.7.1.3	Sampled, Non-Filtered Mode (#s 3A & 3B) .....	9-9
9.7.1.4	Sampled, Filtered Mode (#s 4A & 4B) .....	9-10
9.7.1.5	Windowed Mode (#s 5A & 5B) .....	9-12
9.7.1.6	Windowed/Resampled Mode (# 6) .....	9-14
9.7.1.7	Windowed/Filtered Mode (#7) .....	9-14
9.7.2	Power Modes .....	9-15
9.7.2.1	Wait Mode Operation .....	9-15

9.7.2.2	Stop Mode Operation	9-15
9.7.2.3	Background Mode Operation	9-16
9.7.3	Hysteresis	9-16
9.7.4	Startup and Operation	9-17
9.7.5	Low Pass Filter	9-17
9.7.5.1	Introduction	9-17
9.7.5.2	Enabling Filter Modes	9-18
9.7.5.3	Latency Issues	9-18
9.8	Interrupts	9-19
9.9	DMA Support	9-19
9.10	Memory Map & Register Definitions	9-20
9.10.1	Control Register 0 (HSCMPxCR0)	9-20
9.10.2	Control Register 1 (HSCMPxCR1)	9-21
9.10.3	Filter Period Register (HSCMPxFPR)	9-22
9.10.4	Status & Control Register (HSCMPxSCR)	9-22
9.10.5	Pin Control Register (HSCMPxPCR)	9-23

## Chapter 10

### Analog-to-Digital Converter (S08ADC12V3)

10.1	Introduction	10-1
10.1.1	ADC Clock Gating	10-1
10.1.2	Module Configurations	10-1
10.1.2.1	Channel Assignments	10-1
10.1.2.2	Alternate Clock	10-2
10.1.2.3	Configurations for Stop Modes	10-2
10.1.3	ADC Hardware Trigger	10-2
10.1.4	Features	10-3
10.1.5	Related Material	10-3
10.1.6	Block Diagram	10-3
10.2	External Signal Description	10-4
10.2.1	Analog Power ( $V_{DDAD}$ )	10-5
10.2.2	Analog Ground ( $V_{SSAD}$ )	10-5
10.2.3	Voltage Reference High ( $V_{REFH}$ )	10-5
10.2.4	Voltage Reference Low ( $V_{REFL}$ )	10-5
10.2.5	Analog Channel Inputs (ADx)	10-5
10.3	Register Definition	10-5
10.3.1	Status and Control Register 1A and 1B (ADCSC1A & ADCSC1B)	10-6
10.3.2	Status and Control Register 2 (ADCSC2)	10-8
10.3.3	Data Result Registers A & B (ADCRA and ADCRB)	10-9
10.3.4	Configuration Register (ADCCFG)	10-10
10.4	Functional Description	10-12
10.4.1	Clock Select and Divide Control	10-12
10.4.2	Hardware Trigger	10-12
10.4.3	Conversion Control	10-13
10.4.3.1	Initiating Conversions	10-13

10.4.3.2	Completing Conversions	10-14
10.4.3.3	Aborting Conversions	10-14
10.4.3.4	Power Control	10-14
10.4.3.5	Sample Time and Total Conversion Time	10-14
10.4.4	Temperature Sensor	10-15
10.4.5	MCU Wait Mode Operation	10-16
10.4.6	MCU Stop3 Mode Operation	10-16
10.4.6.1	Stop Mode With ADACK Disabled	10-16
10.4.6.2	Stop Mode With ADACK Enabled	10-16
10.4.7	MCU Partial Power Down Mode Operation	10-17
10.5	Initialization Information	10-17
10.5.1	ADC Module Initialization Example	10-17
10.5.1.1	Initialization Sequence	10-17
10.5.1.2	Pseudo-code Example	10-17
10.6	Application Information	10-19
10.6.1	External Pins and Routing	10-19
10.6.1.1	Analog Supply Pins	10-19
10.6.1.2	Analog Reference Pins	10-20
10.6.1.3	Analog Input Pins	10-20
10.6.2	Sources of Error	10-21
10.6.2.1	Sampling Error	10-21
10.6.2.2	Pin Leakage Error	10-21
10.6.2.3	Noise-Induced Errors	10-21
10.6.2.4	Code Width and Quantization Error	10-22
10.6.2.5	Linearity Errors	10-22
10.6.2.6	Code Jitter, Non-Monotonicity and Missing Codes	10-23

## Chapter 11

### Digital-to-Analog Converter (S08DACV1)

11.1	Introduction	11-1
11.1.1	Overview	11-1
11.1.2	Features	11-2
11.1.3	Modes of Operation	11-2
11.1.4	Block Diagram	11-2
11.2	Memory Map and Registers	11-3
11.2.1	Memory Map	11-3
11.2.2	Registers Descriptions	11-3
11.2.2.1	DAC Control Register (DACCTRL)	11-3
11.2.3	Functional Description	11-3
11.2.3.1	Operation	11-3
11.2.3.2	Voltage Reference Source Select	11-4
11.2.3.3	Other Concerns	11-4
11.3	Resets	11-4
11.4	Clocks	11-4
11.5	Interrupts	11-4

## Chapter 12

### Cyclic Redundancy Check (S08CRCV3)

12.1	Introduction	12-1
12.1.1	CRC Clock Gating	12-1
12.1.2	Features	12-2
12.1.3	Modes of Operation	12-2
12.1.4	Block Diagram	12-3
12.2	External Signal Description	12-3
12.3	Register Definition	12-3
12.3.1	Memory Map	12-3
12.3.2	Register Descriptions	12-4
12.3.2.1	CRC High Register (CRCH)	12-4
12.3.2.2	CRC Low Register (CRCL)	12-4
12.3.2.3	Transpose Register (TRANPOSE)	12-5
12.4	Functional Description	12-5
12.4.1	ITU-T (CCITT) Recommendations and Expected CRC Results	12-6
12.4.2	Programming Model Extension for CF1Core	12-7
12.4.3	Transpose Feature	12-7
12.5	Initialization Information	12-8

## Chapter 13

### FlexTimer Module (S08FTMV3)

13.1	Introduction	13-1
13.1.1	FTM Module-to-Module Interconnects	13-1
13.1.1.1	FTMx Synchronization Trigger	13-1
13.1.1.2	FTMx Fault	13-1
13.1.1.3	FTM1 Input Capture	13-2
13.1.1.4	FTMx to PDB Hardware Trigger	13-2
13.1.2	Features	13-3
13.1.3	Modes of Operation	13-3
13.1.4	Block Diagram	13-4
13.2	Signal Description	13-6
13.2.1	EXTCLK — FTM External Clock	13-6
13.2.2	FTMxCHn — FTM Channel (n) I/O Pin	13-6
13.2.3	FTMxFAULTj — FTM Fault Input	13-6
13.3	Memory Map and Register Definition	13-7
13.3.1	Module Memory Map	13-7
13.3.2	Register Descriptions	13-9
13.3.3	FTM Status and Control Register (FTMxSC)	13-9
13.3.4	FTM Counter Registers (FTMxCNTH:FTMxCNTL)	13-10
13.3.5	FTM Counter Modulo Registers (FTMxMODH:FTMxMODL)	13-11
13.3.6	FTM Channel (n) Status and Control Register (FTMxCnSC)	13-12
13.3.7	FTM Channel Value Registers (FTMxCnVH:FTMxCnVL)	13-14
13.3.8	FTM Counter Initial Value Registers (FTMxCNTINH:FTMxCNTINL)	13-15

13.3.9	FTM Capture and Compare Status Register (FTMxSTATUS)	13-15
13.3.10	FTM Features Mode Selection Register (FTMxMODE)	13-16
13.3.11	FTM Synchronization Register (FTMxSYNC)	13-17
13.3.12	FTM Initial State for Channels Output Register (FTMxOUTINIT)	13-19
13.3.13	FTM Output Mask Register (FTMxOUTMASK)	13-19
13.3.14	FTM Function for Linked Channels Register (FTMxCOMBINEm)	13-20
13.3.15	FTM Deadtime Insertion Control Register (FTMxDEADTIME)	13-21
13.3.16	FTM External Trigger Register (FTMxEXTTRIG)	13-23
13.3.17	FTM Channels Polarity Register (FTMxPOL)	13-24
13.3.18	FTM Fault Mode Status Register (FTMxFMS)	13-25
13.3.19	FTM Input Capture Filter Control Register (FTMxFILTERm)	13-26
13.3.20	FTM Fault Input Filter Control Register (FTMxFLTFILTER)	13-26
13.3.21	FTM Fault Control Register (FTMxFLTCTRL)	13-27
13.4	Functional Description	13-27
13.4.1	Clock Source	13-28
13.4.1.1	Counter Clock Source	13-28
13.4.2	Prescaler	13-29
13.4.3	Counter	13-29
13.4.3.1	Up Counting	13-29
13.4.3.2	Up-Down Counting	13-32
13.4.3.3	Free Running Counter	13-33
13.4.3.4	Counter Reset	13-34
13.4.4	Input Capture Mode	13-34
13.4.4.1	Filter for Input Capture Mode	13-35
13.4.5	Output Compare Mode	13-36
13.4.6	Edge-Aligned PWM (EPWM) Mode	13-38
13.4.7	Center-Aligned PWM (CPWM) Mode	13-39
13.4.8	Combine Mode	13-41
13.4.8.1	Asymmetrical PWM	13-50
13.4.9	Complementary Mode	13-50
13.4.10	Load of the Registers With Write Buffers	13-51
13.4.10.1	FTMxCNTINH:L Registers	13-51
13.4.10.2	FTMxMODH:L Registers	13-51
13.4.10.3	FTMxCnVH:L Registers	13-52
13.4.11	PWM Synchronization	13-52
13.4.11.1	Hardware Trigger	13-52
13.4.11.2	Software Trigger	13-53
13.4.11.3	Boundary Cycle	13-54
13.4.11.4	FTMxMODH:FTMxMODL Synchronization	13-55
13.4.11.5	FTMxCnVH:FTMxCnVL Synchronization	13-57
13.4.11.6	CHnOM Synchronization	13-58
13.4.11.7	FTM Counter Synchronization	13-59
13.4.11.8	Summary of PWM Synchronization	13-61
13.4.12	Deadtime Insertion	13-63
13.4.12.1	Deadtimer Insertion Corner Cases	13-64



13.4.13	Output Mask	13-66
13.4.14	Fault Control	13-67
13.4.14.1	Automatic Fault Clearing	13-68
13.4.14.2	Manual Fault Clearing	13-69
13.4.15	Polarity Control	13-70
13.4.16	Initialization	13-70
13.4.17	Features Priority	13-71
13.4.18	Channel Trigger Output	13-71
13.4.19	Initialization Trigger	13-72
13.4.20	Capture Test Mode	13-74
13.4.21	DMA	13-75
13.4.22	TPM Emulation	13-76
13.4.22.1	FTMxMODH:L and FTMxCnVH:L Synchronization	13-76
13.4.22.2	Free Running Counter	13-76
13.4.22.3	Write to FTMxSC	13-76
13.4.22.4	Write to FTMxCnSC	13-76
13.4.23	BDM Mode	13-77
13.5	Reset Overview	13-77
13.6	FTM Interrupts	13-77
13.6.1	Timer Overflow Interrupt	13-77
13.6.2	Channel (n) Interrupt	13-77
13.6.3	Fault Interrupt	13-77

## Chapter 14

### Inter-Integrated Circuit (S08IICV6)

14.1	Introduction	14-1
14.1.1	Features	14-2
14.1.2	Modes of Operation	14-2
14.1.3	Block Diagram	14-3
14.2	External Signal Description	14-3
14.2.1	SCL — Serial Clock Line	14-3
14.2.2	SDA — Serial Data Line	14-3
14.3	Register Definition	14-4
14.3.1	Module Memory Map	14-4
14.3.2	IIC Address Register 1 (IICA1)	14-4
14.3.3	IIC Frequency Divider Register (IICF)	14-5
14.3.4	IIC Control Register (IICC1)	14-8
14.3.5	IIC Status Register (IICS)	14-10
14.3.6	IIC Data I/O Register (IICD)	14-11
14.3.7	IIC Control Register 2 (IICC2)	14-13
14.3.8	IIC Programmable Input Glitch Filter (IICFLT)	14-13
14.3.9	IIC SMBus Control and Status Register (IIC SMB)	14-15
14.3.10	IIC Address Register 2 (IICA2)	14-16
14.3.11	IIC SCL Low Time Out Register High (IICSLTH)	14-16
14.3.12	IIC SCL Low Time Out register Low (IICSLTL)	14-17

14.4	Functional Description	14-18
14.4.1	IIC Protocol	14-18
14.4.1.1	START Signal	14-19
14.4.1.2	Slave Address Transmission	14-19
14.4.1.3	Data Transfer	14-19
14.4.1.4	STOP Signal	14-20
14.4.1.5	Repeated START Signal	14-20
14.4.1.6	Arbitration Procedure	14-20
14.4.1.7	Clock Synchronization	14-20
14.4.1.8	Handshaking	14-21
14.4.1.9	Clock Stretching	14-21
14.4.2	10-bit Address	14-22
14.4.2.1	Master-Transmitter Addresses a Slave-Receiver	14-22
14.4.2.2	Master-Receiver Addresses a Slave-Transmitter	14-22
14.4.3	Address Matching	14-23
14.4.4	System Management Bus Specification	14-23
14.4.4.1	Timeouts	14-23
14.4.4.2	FAST ACK and NACK	14-25
14.5	Resets	14-25
14.6	Interrupts	14-25
14.6.1	Byte Transfer Interrupt	14-26
14.6.2	Address Detect Interrupt	14-26
14.6.3	Exit from Low-Power/Stop Modes	14-26
14.6.4	Arbitration Lost Interrupt	14-26
14.6.5	Timeouts Interrupt in SMBus	14-27
14.6.6	Programmable Input Glitch Filter	14-27
14.6.7	Address Matching Wakeup	14-27
14.6.8	DMA Support	14-28
14.7	Initialization/Application Information	14-29
14.8	SMBALERT#	14-33

## Chapter 15

### DMA Controller Module

15.1	Introduction	15-1
15.1.1	Overview	15-1
15.1.2	Peripheral DMA request inputs	15-2
15.2	Low Power Mode Operation	15-3
15.2.1	DMA Clock Gating	15-3
15.2.2	Features	15-3
15.3	DMA Transfer Overview	15-4
15.4	Memory Map/Register Definition	15-5
15.4.1	DMA Request Control (DMAREQC)	15-5
15.4.2	Transfer Control Descriptor (TCDn)	15-6
15.4.2.1	Source Address Registers (SAR <sub><i>n</i></sub> )	15-7
15.4.2.2	Destination Address Registers (DAR <sub><i>n</i></sub> )	15-8

15.4.2.3 DMA Status Registers (DSR <sub>n</sub> ) and Byte Count Registers (BCR <sub>n</sub> )	15-8
15.4.2.4 DMA Control Registers (DCR <sub>n</sub> )	15-10
15.5 Functional Description	15-13
15.5.1 Transfer Requests (Cycle-Steal and Continuous Modes)	15-14
15.5.2 Channel Initialization and Startup	15-14
15.5.2.1 Channel Prioritization	15-14
15.5.2.2 Programming the DMA Controller Module	15-14
15.5.3 Dual-Address Data Transfer Mode	15-15
15.5.4 Advanced Data Transfer Controls: Auto-Alignment	15-16
15.5.5 Termination	15-16

## Chapter 16

### Intelligent Event Controller (iEvent)

16.1 Introduction	16-1
16.1.1 Overview	16-1
16.1.2 iEvent Input Resources Distribution	16-5
16.1.3 Low Power Mode Operation	16-5
16.1.4 Clock Gating	16-5
16.1.5 Features	16-5
16.1.6 Modes of Operation	16-6
16.2 External Signal Description	16-6
16.3 Memory Map and Register Definition	16-6
16.3.1 Memory Map	16-7
16.3.2 Register Descriptions	16-7
16.3.2.1 iEvent Data Register “n” (IEVENT_DR <sub>n</sub> )	16-7
16.3.2.2 iEvent Control Register “n” (IEVENT_CR <sub>n</sub> )	16-9
16.3.2.3 iEvent Input Mux Configuration Register “n” (IEVENT_IMXCR <sub>n</sub> )	16-11
16.3.2.4 iEvent Boolean Function Evaluation Configuration Register “n” (IEVENT_BFE <sub>n</sub> )	16-12
16.4 Functional Description	16-14
16.4.1 Configuration Examples for the Boolean Function Evaluation	16-15
16.4.2 Event Output Finite State Machine	16-16
16.4.3 Broadcast Done Details	16-17
16.4.4 Software Channel Maintenance	16-18
16.4.4.1 Channel Soft Resets	16-18
16.4.4.2 Input Capture and Optional Forced Broadcast Done	16-18

## Chapter 17

### Watchdog Timer

17.1 Introduction	17-1
17.2 Features	17-1
17.3 Functional Overview	17-2
17.3.1 Unlocking and Updating the Watchdog	17-3
17.3.2 The Watchdog Configuration Time (WCT)	17-4

17.3.3	Refreshing the Watchdog	17-4
17.3.4	Windowed Mode of Operation	17-4
17.3.5	Watchdog Disabled Mode of Operation	17-4
17.3.6	Low Power Modes of Operation	17-5
17.3.7	Debug Modes of Operation	17-5
17.4	Testing the Watchdog	17-5
17.4.1	Quick Test	17-6
17.4.2	Byte Test	17-6
17.5	Backup Reset Generator	17-7
17.5.1	Generated Resets and Interrupts	17-7
17.6	Register Space	17-8
17.6.1	Memory Map	17-8
17.6.2	Register Description	17-8
17.6.2.1	Watchdog Status and Control Register High (WDOG_ST_CTRL_H)	17-9
17.6.2.2	Watchdog Control and Status Register Low (WDOG_ST_CTRL_L)	17-10
17.6.2.3	Watchdog Time-out Value Register High (WDOG_TO_VAL_H)	17-11
17.6.2.4	Watchdog Time-out Value Register Low (WDOG_TO_VAL_L)	17-11
17.6.2.5	Watchdog Window Register High (WDOG_WIN_H)	17-12
17.6.2.6	Watchdog Window Register Low (WDOG_WIN_L)	17-13
17.6.2.7	Watchdog Refresh Register (WDOG_REFRESH)	17-13
17.6.2.8	Watchdog Unlock Register (WDOG_UNLOCK)	17-14
17.6.2.9	Watchdog Timer Output Register High (WDOG_TIMER_OUT_H)	17-14
17.6.2.10	Watchdog Timer Output Register Low (WDOG_TIMER_OUT_L)	17-15
17.6.2.11	Watchdog Reset Count Register (WDOG_RST_CNT)	17-15
17.7	Watchdog Operation with 8-bit access	17-16
17.7.1	General Guideline	17-16
17.7.2	Refresh and Unlock operations with 8-bit access	17-16
17.8	Restrictions on Watchdog Operation	17-17

## Chapter 18

### External Watchdog Monitor (EWM)

18.1	Introduction	18-1
18.1.1	Features	18-1
18.1.2	Modes of Operation	18-2
18.1.2.1	Stop Mode	18-2
18.1.2.2	Wait Mode	18-2
18.1.2.3	Debug Mode	18-2
18.1.3	Block Diagram	18-3
18.2	External Signal Description	18-3
18.2.1	EWM_out	18-3
18.2.2	EWM_in	18-4
18.3	Memory Map	18-4
18.4	Register Definition	18-5
18.4.1	EWM Compare Low Register (EWMxCMPL)	18-5
18.4.2	EWM Compare High Register (EWMxCMPH)	18-5

18.4.3	EWM Service Register (EWMxSERV)	18-6
18.4.4	EWM Control Register (EWMxCTRL)	18-6
18.5	Functional Description	18-6
18.5.1	EWM Counter	18-7
18.5.2	EWM Compare Registers	18-7
18.5.3	EWM Refresh Mechanism	18-7

## Chapter 19

### Internal Clock Source (S08ICSV3)

19.1	Introduction	19-1
19.1.1	Features	19-2
19.1.2	Block Diagram	19-2
19.1.3	Modes of Operation	19-3
19.1.3.1	FLL Engaged Internal (FEI)	19-3
19.1.3.2	FLL Engaged External (FEE)	19-3
19.1.3.3	FLL Bypassed Internal (FBI)	19-3
19.1.3.4	FLL Bypassed Internal Low Power (FBILP)	19-4
19.1.3.5	FLL Bypassed External (FBE)	19-4
19.1.3.6	FLL Bypassed External Low Power (FBELP)	19-4
19.1.3.7	Stop (STOP)	19-4
19.2	External Signal Description	19-4
19.3	Register Definition	19-4
19.3.1	ICS Control Register 1 (ICSC1)	19-5
19.3.2	ICS Control Register 2 (ICSC2)	19-7
19.3.3	ICS Trim Register (ICSTRM)	19-7
19.3.4	ICS Status and Control (ICSSC)	19-8
19.4	Functional Description	19-10
19.4.1	Operational Modes	19-10
19.4.1.1	FLL Engaged Internal (FEI)	19-10
19.4.1.2	FLL Engaged External (FEE)	19-11
19.4.1.3	FLL Bypassed Internal (FBI)	19-11
19.4.1.4	FLL Bypassed Internal Low Power (FBILP)	19-11
19.4.1.5	FLL Bypassed External (FBE)	19-11
19.4.1.6	FLL Bypassed External Low Power (FBELP)	19-12
19.4.1.7	Stop	19-12
19.4.2	Mode Switching	19-12
19.4.3	Bus Frequency Divider	19-13
19.4.4	Low Power Bit Usage	19-13
19.4.5	DCO Maximum Frequency with 32.768 kHz Oscillator	19-13
19.4.6	Internal Reference Clock	19-13
19.4.7	External Reference Clock	19-14
19.4.8	Fixed Frequency Clock	19-14
19.4.9	Local Clock	19-14

## Chapter 20

### Rapid GPIO (RGPIO)

20.1	Introduction	20-1
20.1.1	Overview	20-1
20.1.2	Features	20-2
20.1.3	Modes of Operation	20-3
20.2	External Signal Description	20-3
20.2.1	Overview	20-3
20.2.2	Detailed Signal Descriptions	20-3
20.3	Memory Map/Register Definition	20-4
20.3.1	RGPIO Data Direction (RGPIO_DIR)	20-5
20.3.2	RGPIO Data (RGPIO_DATA)	20-5
20.3.3	RGPIO Pin Enable (RGPIO_ENB)	20-6
20.3.4	RGPIO Clear Data (RGPIO_CLR)	20-6
20.3.5	RGPIO Set Data (RGPIO_SET)	20-7
20.3.6	RGPIO Toggle Data (RGPIO_TOG)	20-7
20.4	Functional Description	20-8
20.5	Initialization Information	20-8
20.6	Application Information	20-8
20.6.1	Application 1: Simple Square-Wave Generation	20-9
20.6.2	Application 2: 16-bit Message Transmission using SPI Protocol	20-9

## Chapter 21

### Real-Time Counter (S08RTCv2)

21.1	Introduction	21-1
21.2	Low Power Mode Operation	21-1
21.2.1	RTC Clock Gating	21-1
21.2.2	Features	21-2
21.2.3	Modes of Operation	21-2
21.2.3.1	Wait Mode	21-2
21.2.3.2	Stop Modes	21-2
21.2.3.3	Active Background Mode	21-2
21.2.4	Block Diagram	21-3
21.3	External Signal Description	21-3
21.4	Register Definition	21-3
21.4.1	RTC Status and Control Register (RTCSC)	21-4
21.4.2	RTC Counter Register (RTCCNT)	21-5
21.4.3	RTC Modulo Register (RTCMOD)	21-5
21.4.4	RTC DMA Enable Register (RTCSC1)	21-5
21.5	Functional Description	21-6
21.5.1	RTC Operation Example	21-7
21.6	Initialization/Application Information	21-8

## Chapter 22

### Serial Communication Interface (SCI\_FlexV1)

22.1	Introduction	22-1
22.1.1	SCI Clock Gating	22-1
22.1.2	Module Block Diagram	22-1
22.1.3	Features	22-3
22.1.4	Modes of Operation	22-3
22.1.4.1	Run Mode	22-4
22.1.4.2	Wait Mode	22-4
22.1.4.3	Stop Mode	22-4
22.2	Memory Map and Registers	22-4
22.2.1	Module Memory Map	22-4
22.2.2	Register Quick Reference	22-5
22.2.3	Register Descriptions	22-6
22.2.3.1	SCI Baud Rate Registers (SCIxBDH, SCIxBDL)	22-6
22.2.3.2	SCI Control Register 1 (SCIxC1)	22-7
22.2.3.3	SCI Control Register 2 (SCIxC2)	22-8
22.2.3.4	SCI Status Register 1 (SCIxS1)	22-9
22.2.3.5	SCI Status Register 2 (SCIxS2)	22-11
22.2.3.6	SCI Control Register 3 (SCIxC3)	22-12
22.2.3.7	SCI Data Register (SCIxD)	22-13
22.2.3.8	SCI Match Address Registers 1 and 2 (SCIxMA1, SCIxMA2)	22-14
22.2.3.9	SCI Control Register 4 (SCIxC4)	22-14
22.2.3.10	SCI Control Register 5 (SCIxC5)	22-15
22.3	Functional Description	22-16
22.3.1	Data Format	22-16
22.3.2	Baud Rate Generation	22-17
22.3.3	Transmitter	22-19
22.3.3.1	Transmitter Character Length	22-19
22.3.3.2	Character Transmission	22-20
22.3.3.3	Transmitting Break Characters	22-21
22.3.3.4	Idle Characters	22-21
22.3.3.5	Inversion of Transmitted Output	22-22
22.3.4	Receiver	22-23
22.3.4.1	Receiver Character Length	22-24
22.3.4.2	Character Reception	22-24
22.3.4.3	Data Sampling	22-24
22.3.4.4	Framing Errors	22-29
22.3.4.5	Receiving Break Characters	22-29
22.3.4.6	Inversion of Receiver Input	22-29
22.3.4.7	Baud Rate Tolerance	22-30
22.3.4.8	Receiver Wakeup	22-31
22.3.4.9	Match Address Operation	22-33
22.3.5	Single-Wire Operation	22-33
22.3.6	Loop Operation	22-34

22.4	Reset	22-34
22.5	Interrupts	22-34
22.5.1	System Level Interrupt Sources	22-34
22.5.1.1	Recovery from Wait Mode	22-35
22.5.2	Description of Interrupt Operation	22-35
22.5.2.1	TDRE Description	22-35
22.5.2.2	TC Description	22-35
22.5.2.3	RDRF Description	22-35
22.5.2.4	OR Description	22-35
22.5.2.5	IDLE Description	22-36
22.5.2.6	NF Description	22-36
22.5.2.7	FE Description	22-36
22.5.2.8	PF Description	22-36
22.5.2.9	LBKDIF Description	22-36
22.5.2.10	RXEDGIF Description	22-36
22.5.3	Exit from Low-Power Modes	22-37
22.6	DMA Operation	22-38

## Chapter 23

### Serial Peripheral Interface (S08SPIV6)

23.1	Introduction	23-1
23.1.1	Features	23-1
23.1.2	Modes of Operation	23-1
23.1.3	Block Diagrams	23-2
23.1.3.1	SPI System Block Diagram	23-2
23.1.3.2	SPI Module Block Diagram	23-3
23.2	External Signal Description	23-4
23.2.1	SPSCK — SPI Serial Clock	23-5
23.2.2	MOSI — Master Data Out, Slave Data In	23-5
23.2.3	MISO — Master Data In, Slave Data Out	23-5
23.2.4	$\overline{SS}$ — Slave Select	23-5
23.3	Register Definition	23-5
23.3.1	SPI Control Register 1 (SPIxC1)	23-5
23.3.2	SPI Control Register 2 (SPIxC2)	23-7
23.3.3	SPI Baud Rate Register (SPIxBR)	23-8
23.3.4	SPI Status Register (SPIxS)	23-10
23.3.5	SPI Data Registers (SPIxDH:SPIxDL)	23-11
23.3.6	SPI Match Registers (SPIxMH:SPIxML)	23-11
23.4	Functional Description	23-12
23.4.1	General	23-12
23.4.2	Master Mode	23-13
23.4.3	Slave Mode	23-13
23.4.4	SPI Transmission by DMA	23-15
23.4.4.1	Transmit by DMA	23-15
23.4.4.2	Receive by DMA	23-16



23.4.5	Data Transmission Length	23-16
23.4.6	SPI Clock Formats	23-17
23.4.7	SPI Baud Rate Generation	23-19
23.4.8	Special Features	23-20
23.4.8.1	SS Output	23-20
23.4.8.2	Bidirectional Mode (MOMI or SISO)	23-20
23.4.9	Error Conditions	23-21
23.4.9.1	Mode Fault Error	23-21
23.4.10	Low Power Mode Options	23-22
23.4.10.1	SPI in Run Mode	23-22
23.4.10.2	SPI in Wait Mode	23-22
23.4.10.3	SPI in Stop Mode	23-23
23.4.10.4	Reset	23-23
23.4.10.5	Interrupts	23-23
23.4.11	SPI Interrupts	23-23
23.4.11.1	MODF	23-24
23.4.11.2	SPRF	23-24
23.4.11.3	SPTEF	23-24
23.4.11.4	SPMF	23-24
23.5	Initialization/Application Information	23-25
23.5.1	SPI Module Initialization Example	23-25
23.5.1.1	Initialization Sequence	23-25
23.5.1.2	Pseudo—Code Example	23-25

## Chapter 24

### Programmable Delay Block (S08PDBV1)

24.1	Introduction	24-1
24.1.1	Low Power Mode Operation	24-1
24.1.2	PDB Clock Gating	24-1
24.1.3	PDB Input Trigger Assignments	24-1
24.1.4	PDB Output Usage	24-2
24.1.4.1	PDB Interfacing to ADC Hardware Trigger	24-2
24.1.4.2	PDB Interfacing to HSCMP Windowing Function	24-2
24.1.5	Features	24-4
24.1.6	Modes of Operation	24-4
24.1.7	Block Diagram	24-4
24.2	Memory Map/Register Definition	24-9
24.2.1	PDB Control Register 1 (PDBxCTRL1)	24-9
24.2.2	PDB Control Register 2 (PDBxCTRL2)	24-10
24.2.3	PDB Delay Registers (PDBxDLYAL, PDBxDLYAH, PDBxDLYBL, & PDBxDLYBH)	24-11
24.2.4	PDB Modulus Registers (PDBxMODH & PDBxMODL)	24-12
24.2.5	PDB Counter Registers (PDBxCNTH & PDBxCNTL)	24-12
24.2.6	PDB Status and Control Register (PDBxSCR)	24-12
24.3	Functional Description	24-13

24.3.1	Miscellaneous Concerns and SoC Integration	24-13
24.3.2	Impacts of using the prescaler on timing resolution	24-14
24.3.3	Resets	24-14
24.3.4	Clocks	24-14
24.3.5	Interrupts	24-14

## Chapter 25

### Timer/PWM Module (S08TPMV3)

25.1	Introduction	25-1
25.1.1	TPM Clock Gating	25-1
25.1.2	TPM3 Module Interconnects	25-1
25.1.2.1	TPM3 Input Capture	25-1
25.1.3	Features	25-2
25.1.4	Modes of Operation	25-2
25.1.5	Block Diagram	25-3
25.2	Signal Description	25-5
25.2.1	Detailed Signal Descriptions	25-5
25.2.1.1	EXTCLK — External Clock Source	25-5
25.2.1.2	TPMxCn — TPM Channel n I/O Pins	25-5
25.3	Register Definition	25-8
25.3.1	TPM Status and Control Register (TPMxSC)	25-8
25.3.2	TPM-Counter Registers (TPMxCNTH:TPMxCNTL)	25-9
25.3.3	TPM Counter Modulo Registers (TPMxMODH:TPMxMODL)	25-10
25.3.4	TPM Channel n Status and Control Register (TPMxCnSC)	25-11
25.3.5	TPM Channel Value Registers (TPMxCnVH:TPMxCnVL)	25-12
25.4	Functional Description	25-14
25.4.1	Counter	25-14
25.4.1.1	Counter Clock Source	25-14
25.4.1.2	Counter Overflow and Modulo Reset	25-15
25.4.1.3	Counting Modes	25-15
25.4.1.4	Manual Counter Reset	25-15
25.4.2	Channel Mode Selection	25-15
25.4.2.1	Input Capture Mode	25-15
25.4.2.2	Output Compare Mode	25-16
25.4.2.3	Edge-Aligned PWM Mode	25-16
25.4.2.4	Center-Aligned PWM Mode	25-17
25.5	Reset Overview	25-18
25.5.1	General	25-18
25.5.2	Description of Reset Operation	25-19
25.6	Interrupts	25-19
25.6.1	General	25-19
25.6.2	Description of Interrupt Operation	25-19
25.6.2.1	Timer Overflow Interrupt (TOF) Description	25-19
25.6.2.2	Channel Event Interrupt Description	25-20

## Chapter 26

### Version 1 ColdFire Debug (CF1\_DEBUG)

26.1	Introduction	26-1
26.1.1	Overview	26-2
26.1.2	Features	26-3
26.1.3	Modes of Operations	26-4
26.2	External Signal Descriptions	26-6
26.3	Memory Map/Register Definition	26-7
26.3.1	Configuration/Status Register (CSR)	26-9
26.3.2	Extended Configuration/Status Register (XCSR)	26-12
26.3.3	Configuration/Status Register 2 (CSR2)	26-15
26.3.4	Configuration/Status Register 3 (CSR3)	26-18
26.3.5	BDM Address Attribute Register (BAAR)	26-19
26.3.6	Address Attribute Trigger Register (AATR)	26-20
26.3.7	Trigger Definition Register (TDR)	26-21
26.3.8	Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)	26-24
26.3.9	Address Breakpoint Registers (ABLR, ABHR)	26-26
26.3.10	Data Breakpoint and Mask Registers (DBR, DBMR)	26-27
26.3.10.1	Resulting Set of Possible Trigger Combinations	26-28
26.3.11	PST Buffer (PSTB)	26-28
26.4	Functional Description	26-29
26.4.1	Background Debug Mode (BDM)	26-29
26.4.1.1	CPU Halt	26-30
26.4.1.2	Background Debug Serial Interface Controller (BDC)	26-32
26.4.1.3	BDM Communication Details	26-32
26.4.1.4	BDM Command Set Descriptions	26-35
26.4.1.5	BDM Command Set Summary	26-38
26.4.1.6	Serial Interface Hardware Handshake Protocol	26-53
26.4.1.7	Hardware Handshake Abort Procedure	26-55
26.4.2	Real-Time Debug Support	26-58
26.4.3	Real-Time Trace Support with the Visibility Bus Enabled (CSR[VBD] = 0)	26-58
26.4.3.1	Begin Execution of Taken Branch (PST = 0x5)	26-59
26.4.4	Trace Support With the Visibility Bus Disabled (CSR[VBD] = 1)	26-60
26.4.4.1	Begin Execution of Taken Branch (PST = 0x05)	26-63
26.4.4.2	PST Trace Buffer (PSTB) Entry Format	26-64
26.4.4.3	PST/DDATA Example	26-64
26.4.4.4	Processor Status, Debug Data Definition	26-66
26.4.5	Freescall-Recommended BDM Pinout	26-70

## Appendix A

### Revision History

A.1	Changes Between Rev. 2 and Rev. 3	27-1
A.2	Changes Between Rev. 3 and Rev. 4	27-2
A.3	Changes Between Rev. 4 and Rev. 5	27-3

## About This Book

The primary objective of this reference manual is to define the MCF51AG128 processor for software and hardware developers. In addition, this manual supports the MCF51AG96. This book is written from the perspective of the MCF51AG128, and unless otherwise noted, the information applies also to the MCF51AG96. This device has the same functionality as the MCF51AG128; any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are detailed in *MCF51CN128 Data Sheet* (document MCF51AG128). Refer to [Table 1-2](#) for a summary of the differences.

The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, the reader needs to make sure to use the most recent version of the documentation.

To locate any published errata or updates for this document, refer to the world-wide web at <http://www.freescale.com/coldfire>.

## Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with this ColdFire processor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire<sup>®</sup> architecture.

## Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about ColdFire architecture.

## General Information

Useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual* (MCF5200PRM/AD)
- *Using Microprocessors and Microcomputers: The Motorola Family*, William C. Wray, Ross Bannatyne, Joseph D. Greenfield
- *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, David A. Patterson and John L. Hennessy.

## ColdFire Documentation

ColdFire documentation is available from the sources listed on the back cover of this manual, as well as our web site, <http://www.freescale.com/coldfire>.

- Reference manuals — These books provide details about individual ColdFire implementations and are intended to be used in conjunction with the *ColdFire Programmers Reference Manual*.
- Data sheets — Data sheets provide specific data regarding pin-out diagrams, bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs — Each device has a product brief that provides an overview of its features. This document is roughly equivalent to the overview (Chapter 1) of an device's reference manual.
- Application notes — These short documents address specific design issues useful to programmers and engineers working with Freescale Semiconductor processors.

Additional literature is published as new processors become available. For a current list of ColdFire documentation, refer to <http://www.freescale.com/coldfire>.

## Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
mnemonics	In code and tables, instruction mnemonics are shown in lowercase.
<i>italics</i>	Italics indicate variable command parameters. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields, or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
word	A 16-bit data unit <sup>1</sup>
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a don't care.
<i>n</i>	Used to express an undefined numerical value
~	NOT logical operator
&	AND logical operator
	OR logical operator

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

|| Field concatenation operator  
 OVERBAR An overbar indicates that a signal is active-low.

## Register Figure Conventions

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

— Undefined at reset.  
 u Unaffected by reset.  
 [signal\_name] Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

R 

0

 W 

--

 Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.

R 

1

 W 

--

 Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.

R 

FIELDNAME

 W 

--

 Indicates a read/write bit.

R 

FIELDNAME

 W 

--

 Indicates a read-only bit field in a memory-mapped register.

R 

FIELDNAME

 W 

FIELDNAME
-----------

 Indicates a write-only bit field in a memory-mapped register.

R 

FIELDNAME

 W 

w1c
-----

 Write 1 to clear: indicates that writing a 1 to this bit field clears it.

R 

0
FIELDNAME

 W 

FIELDNAME
-----------

 Indicates a self-clearing bit.

# Chapter 1

## Device Overview

### 1.1 The MCF51AG128 Series Microcontrollers

#### 1.1.1 Definition

The MCF51AG128 derivative devices are members of the low-cost, low-power, and high-performance ColdFire V1 Family of 32-bit microcontroller units (MCUs) designed for industrial and appliance applications.

- Operate at processor core speeds up to 50.33 MHz (peripherals operate at half of this speed)
- Integrate technologies that are important for today's consumer and industrial applications, such as, DMA, intelligent event controller (iEvent), EWM, FTM/TPM modules, and CRC hardware accelerator

#### 1.1.2 MCF51AG128 Series Package Availability

The device packages available for the MCF51AG128 series are summarized in [Table 1-1](#).

**Table 1-1. MCF51AG128 Series Package Availability**

Packages	MCF51AG128	MCF51AG96
80-pin LQFP	Yes	Yes
64-pin LQFP	Yes	Yes
64-pin QFP	Yes	Yes
48-pin LQFP	Yes	Yes

#### 1.1.3 MCF51AG128 Series Device Comparison

[Table 1-2](#) compares the MCF51AG128 series microcontrollers.

**Table 1-2. MCF51AG128 Series Device Comparison**

Feature	MCF51AG128			MCF51AG96		
	80-pin	64-pin	48-pin	80-pin	64-pin	48-pin
Flash memory size (KB)	128			96		
RAM size (KB)	16					

Table 1-2. MCF51AG128 Series Device Comparison (continued)

Feature	MCF51AG128			MCF51AG96		
	80-pin	64-pin	48-pin	80-pin	64-pin	48-pin
ColdFire V1 core with BDM (background debug module)	Yes					
HSCMP (analog comparator)	2	2	1	2	2	1
ADC (analog-to-digital converter) channels (12-bit)	24	19	12	24	19	12
CRC (cyclic redundancy check)	Yes					
DAC	2	2	1	2	2	1
DMA controller	4-ch					
iEvent (intelligent Event module)	Yes					
EWM (External Watchdog Monitor)	Yes					
WDOG (Watchdog timer)	Yes					
RTC	Yes					
DBG (debug module)	Yes					
IIC (inter-integrated circuit)	1	1	No	1	1	No
IRQ (interrupt request input)	Yes					
INTC (interrupt controller)	Yes					
LVD (low-voltage detector)	Yes					
ICS (internal clock source)	Yes					
OSC (crystal oscillator)	Yes					
Port I/O <sup>1</sup>	69	53	39	69	53	39
RGPIO (rapid general-purpose I/O)	16	16	15	16	16	15
SCI (serial communications interface)	2					
SPI1 (serial peripheral interface)	Yes					
SPI2 (serial peripheral interface)	Yes	No	No	Yes	No	No
FTM1 (flexible timer module) channels	6 <sup>2</sup>					
FTM2 channels	6 <sup>2</sup>					
TPM3 (timer pulse-width modulator) channels	2					
Debug Visibility Bus	Yes	No	No	Yes	No	No

<sup>1</sup> Up to 16 pins on Ports E and F are shared with the ColdFire Rapid GPIO module.

<sup>2</sup> Some pins of FTMx might not be bonded on small package, therefore these channels could be used as soft timer only.



## 1.2 MCF51AG128 Series Block Diagram

### 1.2.1 Block Diagram

Figure 1-1 shows the connections between the MCF51AG128 series pins and functional units.

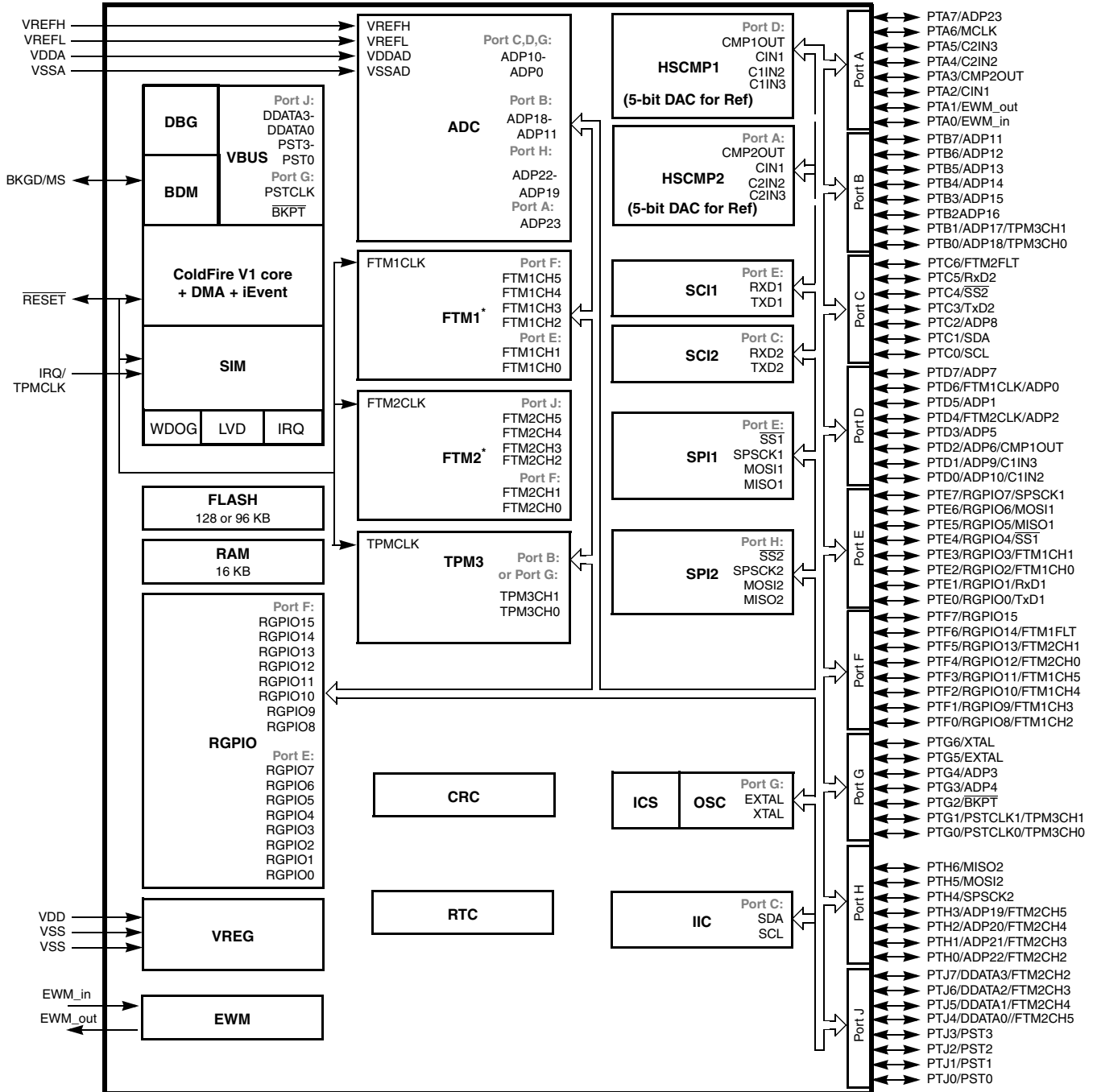


Figure 1-1. MCF51AG128 Series Block Diagram

## 1.2.2 Functional Units

Table 1-3 describes the functional units of the MCF51AG128 series microcontrollers.

**Table 1-3. MCF51AG128 Series Functional Units**

Functional Unit	Function
CF1CORE (V1 ColdFire core)	Executes programs and interrupt handlers
BDM (background debug module)	Provides single pin debugging interface (part of the V1 ColdFire core)
DBG (debug)	Provides debugging and emulation capabilities (part of the V1 ColdFire core)
VBUS (debug visibility bus)	Allows for real-time program traces (part of the V1 ColdFire core)
SIM (system integration module)	Controls resets and chip level interfaces between modules
Flash (flash memory)	Provides storage for program code, constants and variables
RAM (random-access memory)	Provides storage for program variables
Port I/O	General purpose I/O or as the pins of on-chip peripherals function
RGPIO (rapid general-purpose input/output)	Allows for I/O port access at CPU clock speeds
VREG (voltage regulator)	Controls power management across the device
LVD (low-voltage detect)	Monitors internal and external supply voltage levels, and generates a reset or interrupt when the voltages are too low
CF1_INTIC (interrupt controller)	Controls and prioritizes all device interrupts
ADC (analog-to-digital converter)	Measures analog voltages at up to 12 bits of resolution
FTM1, FTM2 (flexible timer/pulse-width modulators)	Provide a variety of timing-based features
TPM3 (timer/pulse-width modulator)	Provides a variety of timing-based features
CRC (cyclic redundancy check)	Accelerates computation of CRC values for ranges of memory
HSCMP1, HSCMP2 (analog comparators)	Compare two analog inputs
DAC1, DAC2 (digital-to-analog converter)	Provide programmable voltage reference for HSCMPx
IIC (inter-integrated circuit)	Supports standard IIC communications protocol
ICS (internal clock source)	Provides clocking options for the device, including a frequency-locked loop (FLL) for multiplying slower reference clock sources
OSC (crystal oscillator)	Allows a crystal or ceramic resonator to be used as the system clock source or reference clock for the FLL
SCI1, SCI2 (serial communications interfaces)	Serial communications UARTs capable of supporting RS-232 and LIN protocols
SPI1, SPI2 (8/16-bit serial peripheral interfaces)	Provide 8/16-bit 4-pin synchronous serial interface
DMA	Provides the means to directly transfer data between system memory and I/O peripherals
iEvent	Highly programmable module for creating combinational boolean outputs for use as interrupt requests, DMA transfer requests or hardware triggers

**Table 1-3. MCF51AG128 Series Functional Units (continued)**

Functional Unit	Function
EWM (External Watchdog Monitor)	Additional watchdog system to help reset external circuits
WDOG (Watchdog timer)	Keeps a watch on the system functioning and resets it in case of its failure
RTC (Real Time Counter)	Provides a constant time-base with optional interrupt

## 1.2.3 Module Versions

Table 1-4 provides the functional version of the on-chip modules.

**Table 1-4. Module Versions**

Module		Version
High-Speed Analog Comparator	(HSCMP)	2
Analog-to-Digital Converter	(ADC12)	3
ColdFire V1 Core	(CF1)	1
Cyclic Redundancy Check	(CRC)	3
Flexible Timer Pulse Width Modulator	(FTM)	3
General Purpose I/O	(eGPIO)	1
Inter-Integrated Circuit	(IIC)	6
Internal Clock Source	(ICS)	3
Oscillator	(XOSC)	1
Real-Time Counter	(RTC)	2
Serial Communications Interface	(SCI_flx)	1
Serial Peripheral Interface	(SPI)	6
DMA controller (4 channel)	(DMA_4)	1
Digital -to-Analog Converter	(DAC)	1
intelligent Event Controller	(iEvent)	1
External Watchdog Monitor	(EWM)	1
Timer Pulse Width Modulator	(TPM)	3
Watchdog timer	(WDOG)	1
Voltage Regulator	(PMC)	1

## 1.3 V1 ColdFire Core

The MCF51AG128 series devices contain a version of the V1 ColdFire core optimized for area and low power. This CPU implements ColdFire instruction set architecture revision C (ISA\_C) and adds support for real-time program trace capability:

- No hardware support for MAC/EMAC and DIV instructions<sup>1</sup>
- Upward compatibility with all other ColdFire cores (V2–V5)
- Debug visibility bus for real-time program tracing

For more details on the V1 ColdFire core, see [Chapter 7, “ColdFire Core.”](#)

### 1.3.1 User Programming Model

[Figure 1-2](#) illustrates the integer portion of the user programming model. It 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)

### 1.3.2 Supervisor Programming Model

System programmers use the supervisor programming model to implement operating system functions. All accesses that affect the control features of ColdFire processors must be made in supervisor mode and can be accessed only by privileged instructions. The supervisor programming model consists of the registers available in user mode as well as the registers listed in [Figure 1-2](#).

31	...	20	19	18	17	16	15	...	0		
							CCR			SR	Status register
										OTHER_A7	Supervisor A7 stack pointer
							Must be zeros			VBR	Vector base register
										CPUCR	CPU Configuration Register

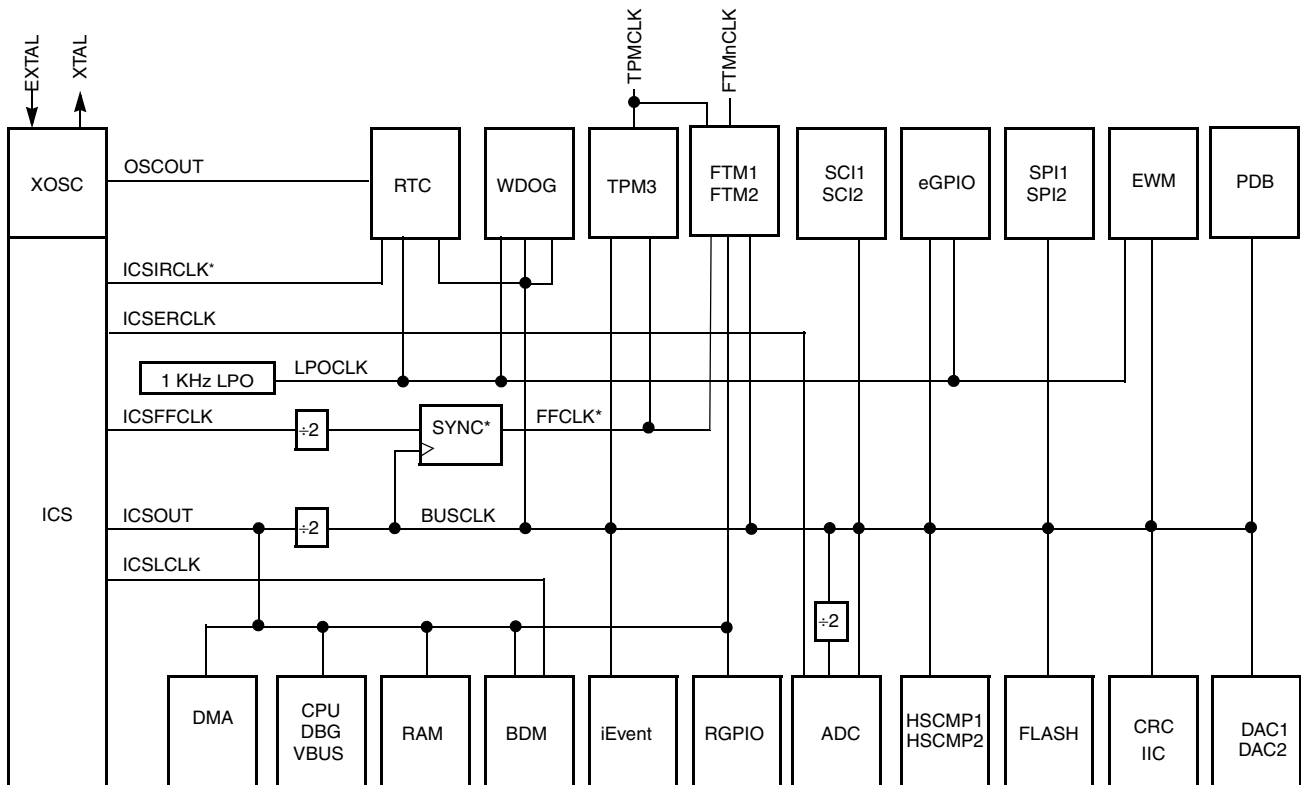
**Figure 1-2. Supervisor Programming Model**

<sup>1</sup> These operations can be emulated via software functions.

## 1.4 System Clock Generation and Distribution

### 1.4.1 Clock Distribution Diagram

Figure 1-3 shows how clocks from the ICS and OSC are distributed to the microcontroller's other functional units. Some modules in the microcontroller have selectable clock inputs. All memory-mapped registers associated with the modules (except RGPIO) are clocked with BUSCLK. The RGPIO registers are clocked with the CPU clock (ICSOUT).



Note: The ADC has minimum and maximum frequency requirements. See [Chapter 10, "Analog-to-Digital Converter \(S08ADC12V3\)"](#) and the *MCF51AG128 Data Sheet* (document MCF51AG128). Flash memory has frequency requirements for program and erase operations.

\* The fixed frequency clock (FFCLK) is internally synchronized to the bus clock (BUSCLK) and must not exceed one half of the bus clock frequency.

\* The OSCOUT clock is mapped on the ERCLK port of the RTC, and is also available in stop 3 and stop 4 modes.

\* The ICSIRCLK is mapped on the IRCLK port of the RTC.

**Figure 1-3. Clock Distribution Diagram**

## 1.4.2 System Clocks

Table 1-5 describes each of the system clocks.

**Table 1-5. System Clocks**

Clock	Description
ICSOUT	<p>This clock source is used as the CPU clock and is divided by two to generate the peripheral bus clock. Control bits in the ICS control registers determine which of three clock sources is connected:</p> <ul style="list-style-type: none"> <li>• Internal reference clock</li> <li>• External reference clock</li> <li>• Frequency-locked loop (FLL)</li> </ul> <p>This clock drives the CPU, DBG, VBUS, RAM, RGPIO, FTM, and BDM directly, and is divided by two to clock all peripherals (BUSCLK). The FTM modules can select ICSOUT as their clock input.</p>
ICSLCLK	<p>This clock source is derived from the 10/20 MHz digitally controlled oscillator (DCOL) of the ICS when configured to run off of the internal or external reference clock. Development tools can select this internal self-clocked source (~10 MHz) to speed up BDC communications in systems where the bus clock is slow.</p>
ICSERCLK	<p>ICS External Reference Clock—This is the external reference clock and can be selected as the alternate clock for the ADC.</p>
ICSIRCLK	<p>ICS Internal Reference Clock—This is the internal reference clock and can be selected as the real-time counter (RTC) clock source.</p>
ICSFFCLK	<p>ICS Fixed-Frequency Clock—This generates the fixed frequency clock (FFCLK) after being synchronized to the bus clock. FFCLK can be selected as clock source for the TPM3 and FTMx modules.</p>
LPOCLK	<p>Low-Power Oscillator Clock—This clock is generated from an internal low-power oscillator that is completely independent of the ICS module. The LPOCLK can be selected as the clock source to the WDOG, EWM, and RTC. The LPOCLK can also be used as the clock source for digital filter part of eGPIO.</p>
TPMCLK	<p>TPM Clock—An optional external clock source for the FTMx and TPM3. This clock must be limited to one-quarter the frequency of the bus clock for synchronization. Refer to the SOPT2[TPMCCFG] bit description in <a href="#">Section 5.8.8, “System Options 2 (SOPT2) Register,”</a> for details on using TPMCLK with the FTMx.</p>
FTM $n$ CLK	<p>FTM Clock—An optional external clock source for the FTMs. This clock must be limited to one-quarter the frequency of the bus clock for synchronization.</p>
ADACK (not shown)	<p>The ADC module also has an internally generated asynchronous clock which allows it to run in STOP mode (ADACK). This signal is not available externally.</p>

---

## Chapter 2

# Pins and Connections

This section describes signals that connect to package pins. It includes pinout diagrams, recommended system connections, and detailed discussions of signals.

### 2.1 Device Pin Assignment

#### 2.1.1 Pinout: 80-Pin LQFP

[Figure 2-1](#) shows the pinout of the 80-pin LQFP.



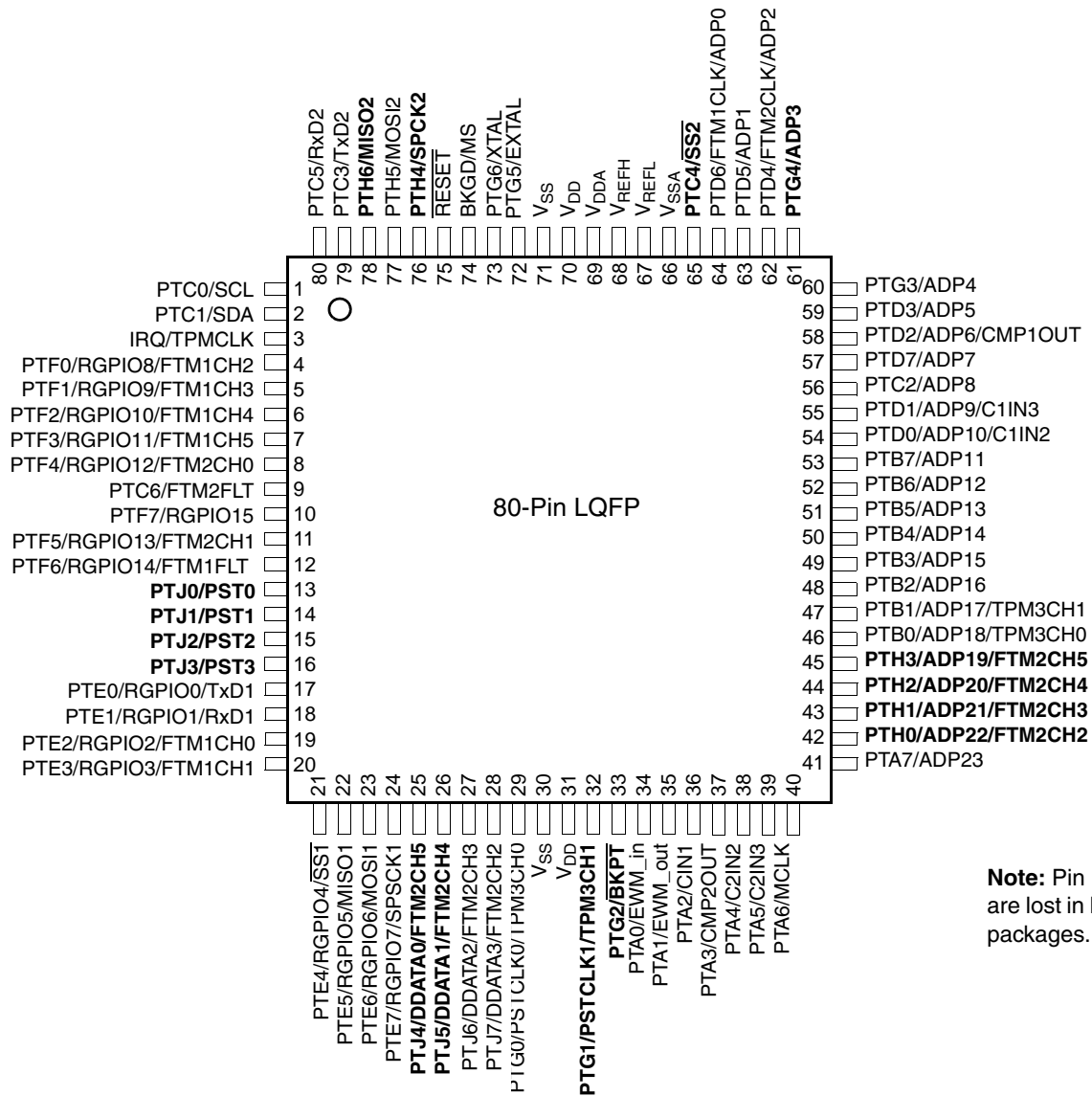


Figure 2-1. 80-Pin LQFP

Note: Pin names in bold are lost in lower pin count packages.

### 2.1.2 Pinout: 64-Pin LQFP and QFP

Figure 2-2 shows the pinout of the 64-pin LQFP and QFP.

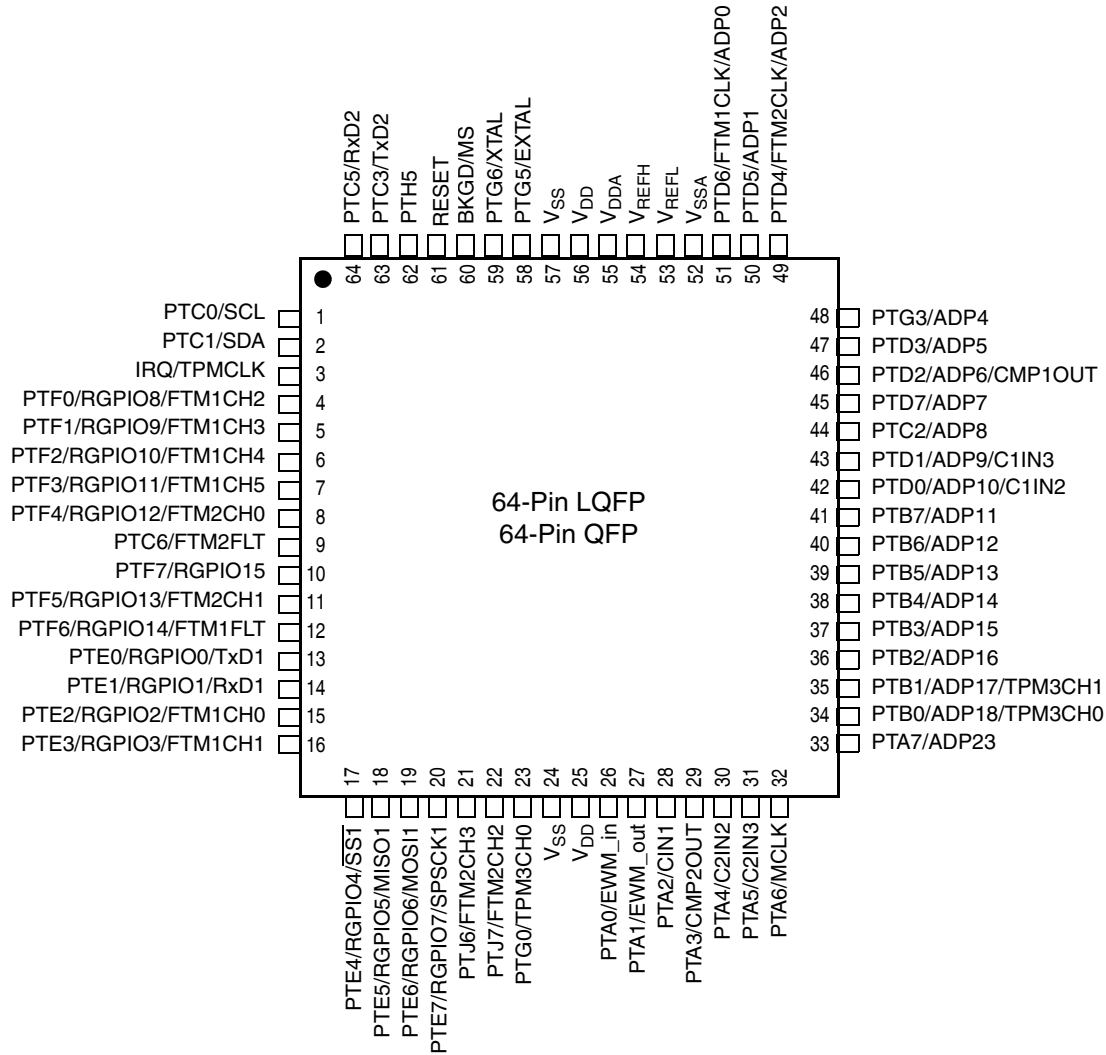


Figure 2-2. 64-Pin LQFP and QFP

### 2.1.3 Pinout: 48-Pin LQFP

Figure 2-3 shows the pinout of the 48-pin LQFP.

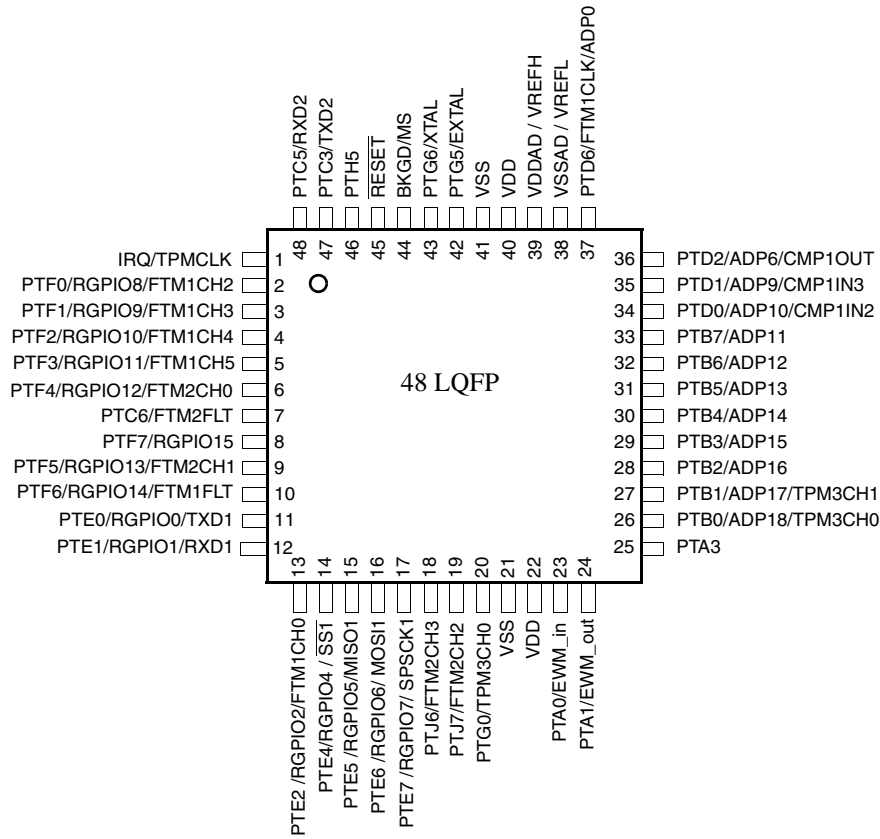


Figure 2-3. 48-Pin LQFP

Table 2-1 shows the package pin assignments.

Table 2-1. Pin Availability by Package Pin-Count

Pin Number			Lowest <-- Priority --> Highest		
80	64	48	Port Pin	Alt 1	Alt 2
1	1	—	PTC0	SCL	
2	2	—	PTC1	SDA	
3	3	1	IRQ	TPMCLK <sup>1</sup>	
4	4	2	PTF0	RGPIO8	FTM1CH2
5	5	3	PTF1	RGPIO9	FTM1CH3
6	6	4	PTF2	RGPIO10	FTM1CH4
7	7	5	PTF3	RGPIO11	FTM1CH5
8	8	6	PTF4	RGPIO12	FTM2CH0
9	9	7	PTC6	FTM2FLT	
10	10	8	PTF7	RGPIO15	
11	11	9	PTF5	RGPIO13	FTM2CH1
12	12	10	PTF6	RGPIO14	FTM1FLT
13	—	—	PTJ0	PST0	
14	—	—	PTJ1	PST1	
15	—	—	PTJ2	PST2	
16	—	—	PTJ3	PST3	
17	13	11	PTE0	RGPIO0	TxD1
18	14	12	PTE1	RGPIO1	RxD1
19	15	13	PTE2	RGPIO2	FTM1CH0
20	16	—	PTE3	RGPIO3	FTM1CH1
21	17	14	PTE4	RGPIO4	$\overline{SS1}$
22	18	15	PTE5	RGPIO5	MISO1
23	19	16	PTE6	RGPIO6	MOSI1
24	20	17	PTE7	RGPIO7	SPSCK1
25	—	—	PTJ4	DDATA0	FTM2CH5
26	—	—	PTJ5	DDATA1	FTM2CH4
27	21	18	PTJ6	DDATA2	FTM2CH3
28	22	19	PTJ7	DDATA3	FTM2CH2
29	23	20	PTG0	PSTCLK0	TPM3CH0
30	24	21	V <sub>SS</sub>		
31	25	22	V <sub>DD</sub>		
32	—	—	PTG1	PSTCLK1	TPM3CH1
33	—	—	PTG2	$\overline{BKPT}$	
34	26	23	PTA0	EWM_in	
35	27	24	PTA1	EWM_out	
36	28	—	PTA2	CIN1	
37	29	25	PTA3	CMP2OUT	
38	30	—	PTA4	C2IN2	
39	31	—	PTA5	C2IN3	
40	32	—	PTA6	MCLK	

Table 2-1. Pin Availability by Package Pin-Count (continued)

Pin Number			Lowest <-- Priority --> Highest		
80	64	48	Port Pin	Alt 1	Alt 2
41	33	—	PTA7	ADP23	
42	—	—	PTH0	ADP22	FTM2CH2
43	—	—	PTH1	ADP21	FTM2CH3
44	—	—	PTH2	ADP20	FTM2CH4
45	—	—	PTH3	ADP19	FTM2CH5
46	34	26	PTB0	ADP18	TPM3CH0
47	35	27	PTB1	ADP17	TPM3CH1
48	36	28	PTB2	ADP16	
49	37	29	PTB3	ADP15	
50	38	30	PTB4	ADP14	
51	39	31	PTB5	ADP13	
52	40	32	PTB6	ADP12	
53	41	33	PTB7	ADP11	
54	42	34	PTD0	ADP10	C1IN2
55	43	35	PTD1	ADP9	C1IN3
56	44	—	PTC2	ADP8	
57	45	—	PTD7	ADP7	
58	46	36	PTD2	ADP6	CMP1OUT
59	47	—	PTD3	ADP5	
60	48	—	PTG3	ADP4	
61	—	—	PTG4	ADP3	
62	49	—	PTD4	FTM2CLK	ADP2
63	50	—	PTD5	ADP1	
64	51	37	PTD6	FTM1CLK	ADP0
65	—	—	PTC4	SS2	
66	52	38	V <sub>SSA</sub>		
67	53	38	V <sub>REFL</sub>		
68	54	39	V <sub>REFH</sub>		
69	55	39	V <sub>DDA</sub>		
70	56	40	V <sub>DD</sub>		
71	57	41	V <sub>SS</sub>		
72	58	42	PTG5	EXTAL	
73	59	43	PTG6	XTAL	
74	60	44	BKGD	MS	
75	61	45	RESET		
76	—	—	PTH4	SPSCK2	
77	62	46	PTH5	MOSI2	
78	—	—	PTH6	MISO2	
79	63	47	PTC3	TxD2	
80	64	48	PTC5	RxD2	

- <sup>1</sup> TPMCLK, FTM1CLK, and FTM2CLK options are configured via software; out of reset, FTM1CLK, FTM2CLK, and TPMCLK are available to FTM1, FTM2, and TPM3 respectively.

## 2.1.4 Recommended System Connections

Figure 2-4 shows pin connections that are common to MCF51AG128 series application systems.

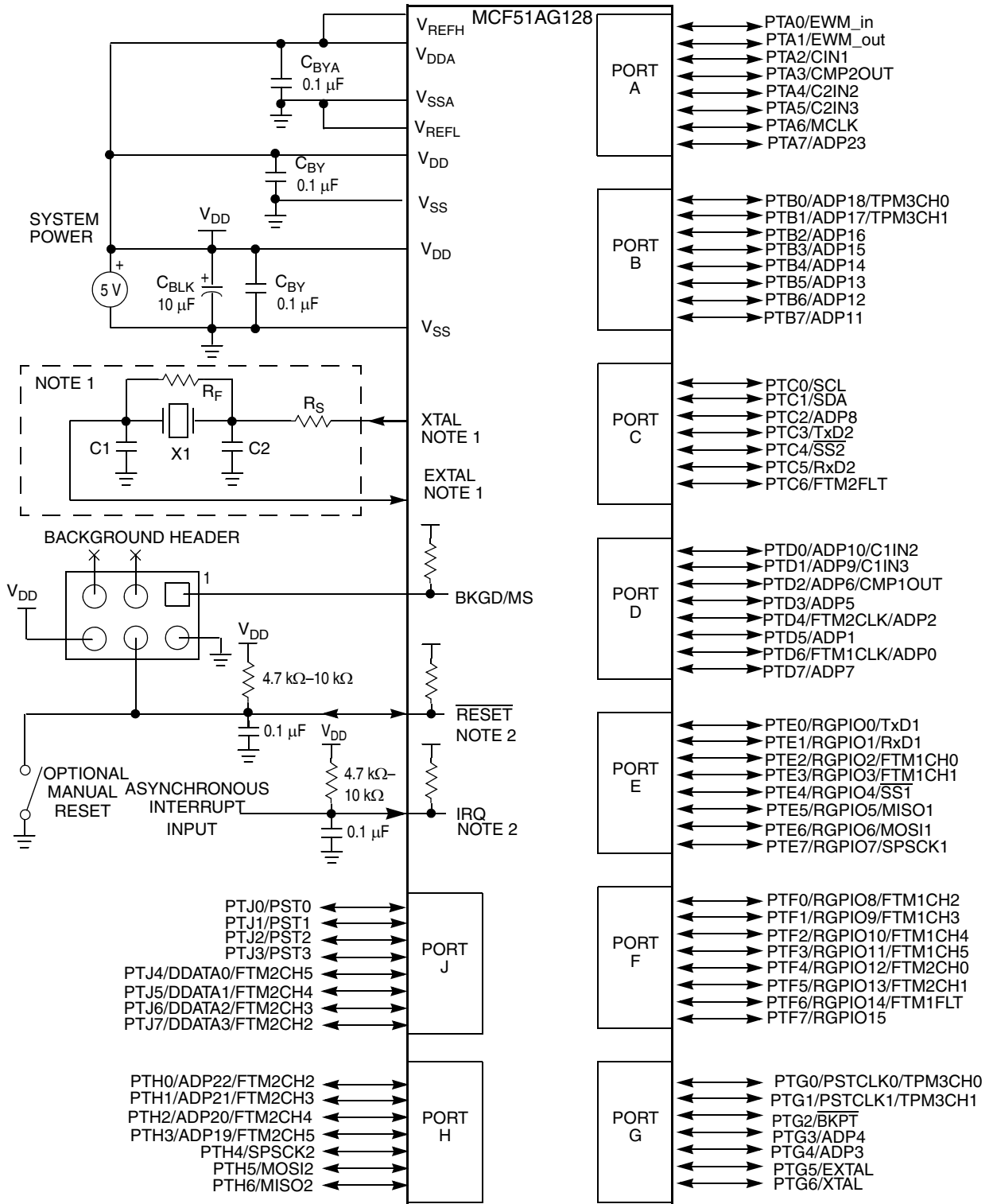


Figure 2-4. Basic System Connections

### 2.1.5 Power ( $V_{DD}$ , $V_{SS}$ , $V_{DDA}$ , and $V_{SSA}$ )

$V_{DD}$  and  $V_{SS}$  are the primary power supply pins for the microcontroller. This voltage source supplies power to all I/O buffer circuitry and to an internal voltage regulator. The internal voltage regulator provides regulated lower-voltage source to the CPU and other internal circuitry of the microcontroller.

Typically, application systems have two separate capacitors across the power pins. In this case, there must be a bulk electrolytic capacitor, such as a 10  $\mu\text{F}$  tantalum capacitor, to provide bulk charge storage for the overall system and a 0.1  $\mu\text{F}$  ceramic bypass capacitor located as close to the microcontroller power pins as practical to suppress high-frequency noise. The MCF51AG128 in 80-pin package has two  $V_{DD}$  pins. Each pin must have a bypass capacitor for best noise suppression.

$V_{DDA}$  and  $V_{SSA}$  are the analog power supply pins for the microcontroller. This voltage source supplies power to the ADC, HSCMP, and DAC modules. A 0.1  $\mu\text{F}$  ceramic bypass capacitor must be located as close to the microcontroller analog power pins as practical to suppress high-frequency noise.

### 2.1.6 Oscillator (XTAL and EXTAL)

Immediately after reset, the microcontroller uses an internally generated clock provided by the internal clock source (ICS) module.

The oscillator (XOSC) in this microcontroller is a pierce oscillator that can accommodate a crystal or ceramic resonator. Optionally, an external clock source can be connected to the EXTAL input pin.

Refer to [Figure 2-4](#) for the following discussion.  $R_S$  (when used) and  $R_F$  must be low-inductance resistors such as carbon composition resistors. Wire-wound resistors and some metal film resistors, have too much inductance. C1 and C2 normally must be high-quality ceramic capacitors that are specifically designed for high-frequency applications.

$R_F$  is used to provide a bias path to keep the EXTAL input in its linear range during crystal startup; its value is not generally critical. Typical systems use 1 M $\Omega$  to 10 M $\Omega$ . Higher values are sensitive to humidity and lower values reduce gain and (in extreme cases) could prevent startup.

C1 and C2 are typically in the 5 pF to 25 pF range and are chosen to match the requirements of a specific crystal or resonator. Be sure to take into account printed circuit board (PCB) capacitance and microcontroller pin capacitance when selecting C1 and C2. The crystal manufacturer typically specifies a load capacitance which is the series combination of C1 and C2 (which are usually the same size). As a first-order approximation, use 10 pF as an estimate of combined pin and PCB capacitance for each oscillator pin (EXTAL and XTAL).



## 2.1.7 $\overline{\text{RESET}}$

The active-low  $\overline{\text{RESET}}$  pin provides the mechanism for off-chip logic to reset the microcontroller. After a power-on reset (POR),  $\overline{\text{RESET}}$  is configured:

- As an open drain to also indicate whether an internal reset (caused by an on-chip mechanism) is in progress
- With its internal pullup resistor enabled

In this configuration, after a reset, the internal pullup resistor pulls  $\overline{\text{RESET}}$  high unless one of the following conditions is true:

- Off-chip logic is asserting  $\overline{\text{RESET}}$ .
- An internal reset is in progress. (A reset takes approximately nn CPU cycles.)

### NOTE

- The  $\overline{\text{RESET}}$  pin does not have a clamp diode to  $V_{DD}$  and must not be driven above  $V_{DD}$ .
- In EMC-sensitive applications, an external RC filter is recommended on the  $\overline{\text{RESET}}$  pin. See [Figure 2-4](#) for an example.

## 2.1.8 IRQ/TPMCLK

The IRQ pin is the input source for the IRQ interrupt. If the IRQ function is not enabled, this pin can be used for TPMCLK. In EMC-sensitive applications, an external RC filter is recommended on the IRQ pin. See [Figure 2-4](#) for an example.

## 2.1.9 Background/Mode Select (BKGD/MS)

During a power-on-reset (POR) or background debug force reset (see bit ENBDM in [Section 26.3.2](#), “[Extended Configuration/Status Register \(XCSR\)](#),” for more information), the BKGD/MS pin functions as a mode select pin. Immediately after any reset, the pin functions as the background pin and can be used for background debug communication.

The BKGD/MS pin has an internal pullup device that is always enabled. If this pin is unconnected, the microcontroller will enter normal operating mode at the rising edge of the internal reset after a POR or forced BDC reset. If a debug system is connected to the 6-pin standard background debug header, it can hold BKGD/MS low during a POR or immediately after issuing a background debug force reset<sup>1</sup>, which forces the microcontroller to halt mode.

The BKGD/MS pin is used primarily for background debug controller (BDC) communications using a custom protocol that uses 16 clock cycles of the target microcontroller’s BDC clock per bit time. The target microcontroller’s BDC clock could be as fast as the bus clock rate, so there must never be any significant capacitance connected to the BKGD/MS pin that could interfere with background serial communications.

Although the BKGD/MS pin is a pseudo open-drain pin, the background debug communication protocol provides brief, actively driven, high speed-up pulses to ensure fast rise times. Small capacitances from

1. Specifically, BKGD must be held low through the first 16 bus cycles after deassertion of the internal reset.

cables and the absolute value of the internal pullup device play almost no role in determining rise and fall times on the BKGD/MS pin.

### 2.1.10 ADC Reference Pins ( $V_{REFH}$ , $V_{REFL}$ )

The  $V_{REFH}$  and  $V_{REFL}$  pins are the voltage reference high and voltage reference low inputs, respectively, for the ADC module.

### 2.1.11 General-Purpose I/O and Peripheral Ports

The MCF51AG128 series microcontrollers support up to 69 general-purpose I/O pins, which are shared with on-chip peripheral functions (timers, serial I/O, ADC, HSCMP, etc.).

When a port pin is configured as a general-purpose output or a peripheral uses the port pin as an output, software can select one of two drive strengths and enable or disable slew rate control. When a port pin is configured as a general-purpose input or a peripheral uses the port pin as an input, software can enable a pull up/pull down device. Immediately after reset, all of these pins are configured as high-impedance general-purpose inputs with internal pull devices disabled.

When a pin is being used as an input, the passive input filter can be enabled using passive filter enable register (PTxPTE) to filter out high frequency transients and prevent system noise from causing an invalid value to be read. When an on-chip peripheral system is controlling a pin, data direction control bits (PTxDDn) still determine what is read from the port data registers, even though the peripheral controls the pin direction via the pin's output buffer enable. The independent pin value register (PTxPV) reads the logic level on the pin directly. For information about controlling these pins as general-purpose I/O pins, see [Chapter 6, "Parallel Input/Output Control."](#)

# Chapter 3

## Modes of Operation

### 3.1 Introduction

This chapter describes the operating modes of the MCF51AG128 series MCUs, and discusses entry, exit, and device functionality pertaining to each mode.

The overall system mode is generally a function of a number of separate, but interrelated, variables: debug mode, security mode, power mode, and clock mode. Clock modes are discussed in [Section 19.1.3, “Modes of Operation.”](#) This chapter covers the other dimensions of the system operating mode.

### 3.2 Summary of Modes

- Debug mode for code development — For devices based on the V1 ColdFire core, such as those in the MCF51AG128 series, debug mode and secure mode are mutually exclusive.
- Secure mode — BDC access to CPU resources is extremely restricted. It is possible to tell that the device has been secured, and to clear security, which involves mass erasing the on-chip flash memory. No other CPU access is allowed. Secure mode can be used in conjunction with each of the power modes below.
- Run mode — CPU clocks can be run at full speed, and the internal supply is fully regulated.
- Wait mode — The CPU shuts down to conserve power; peripheral clocks are running and full regulation is maintained.
- Stop modes — System (CPU and bus) clocks are stopped.
  - Stop4 — All internal circuits are powered (full regulation mode), and the internal clock sources are still at maximum frequency for fast recovery. LVD are enabled or  $XCSR[ENBDM] = 1$ .
  - Stop3 — All internal circuits are loosely regulated and clocks sources are at minimal values, providing a good compromise between power use and speed recovery. LVD in stop should not be enabled, and  $XCSR[ENBDM] = 0$ .
  - Stop2 — Partial power-down of internal circuits; RAM content is retained. The lowest power mode for this device. Upon wake from stop2 mode, the MCU goes through a reset sequence, even if the wakeup source was an interrupt. LVD is not active,  $XCSR[ENBDM] = 0$ , and  $PPDC = 1$ .

On the series MCUs, wait, stop2, stop3, and stop4 are all entered via the CPU STOP instruction. See [Table 3-1](#) and subsequent sections of this chapter for details.

### 3.3 Overview

The ColdFire CPU has two primary modes of operation: run and stop. The STOP instruction is used to invoke both stop and wait modes for this family of devices. The CPU does not differentiate between stop and wait modes.

If the SOPT1[WAITE] bit is set when STOP is executed, the wait mode is entered. Otherwise, if the SOPT1[STOPE] bit is set, the CPU enters one of the stop modes. It is illegal to execute a STOP instruction if neither STOPE or WAITE are set. This results in reset assertion if the instruction-related reset disable bit in the CPU control register (CPUCR[IRD]) is cleared or an illegal instruction exception if CPUCR[IRD] is set.

The MCF51AG128 devices augment stop, wait, and run in a number of ways. The power management controller (PMC) can run the device in fully-regulated mode, standby mode, and partial power-down mode. Standby (loose regulation) or partial power-down can be programmed to occur naturally as a result of a STOP instruction.

During partial power-down mode, the regulator is in standby mode and most of the digital logic on the chip is switched off. These interactions can be seen schematically in Figure 3-1. This figure is for conceptual purposes only. It does not reflect any sequence or time dependencies between the PMC and other parts of the device, nor does it represent any actual design partitioning.

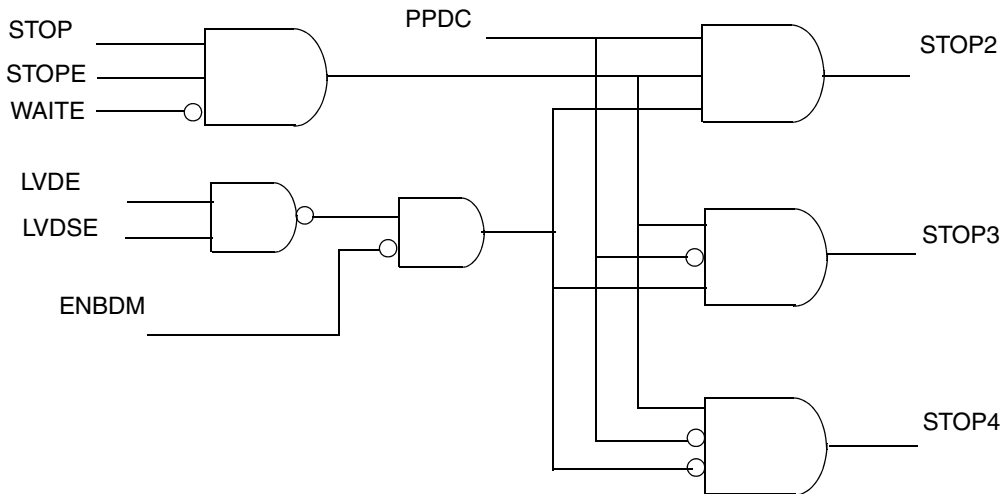


Figure 3-1. MCF51AG128 Stop Modes - Conceptual Drawing

Table 3-1. CPU / Power Mode Selections

Mode of Operation	Register and Bit names						CPU and Peripheral Clocks	Affects on Sub-System
	SOPT1		XCSR	SPMSC1		SPMS C2		
	STOPE	WAITE	ENBDM <sup>1</sup>	LVDE	LVDSE	PPDC		
RUN mode — processor and peripherals clocked normally.	x	x	x	x	x	x	on. ICS in any mode	x
WAIT mode — processor clock nominally inactive, but peripherals are clocked.	x	1	x	x	x	x	Peripherals on and clocked. ICS in any mode	x
Stop modes disabled; illegal opcode reset if STOP instruction executed and CPUCCR[IRD] is cleared, else an illegal instruction exception is generated.	0	0	function of BKGD/MS at reset	x	x	x	on. ICS in any mode	function of BKGD/MS at reset
STOP4 — low voltage detects are enabled or ENBDM = 1.	1	0	x	1	1	x	CPU clock on and peripheral clocks off if XCSR[ENBDM]=1	LVD enabled
			1	x	x			BDC clock enabled only if ENBDM=1 prior to entering stop.
STOP3 — Low voltage detect in STOP is disabled. If BDC is enabled, STOP4 is invoked rather than STOP3.	1	0	0	x	0	0	ICS in stop mode. CPU and bus clocks are off. LPO, internal or external Ref clock can be enabled for RTC wakeup.	LVD disabled in stop only.
				0	x			LVD disabled in all modes.
STOP2 — Low Voltage Detect in stop is disabled. If BDC is enabled, STOP4 is invoked rather than STOP2.	1	0	0	x	0	1	ICS powered off. LPO clock can be enabled for RTC wakeup.	LVD disabled in stop only. Only RAM powered.
				0	x			LVD disabled. Only RAM powered.

<sup>1</sup> ENBDM is located in the upper byte of the XCSR register which is write accessible only through BDC commands, see Debug module [Section 26.3.3, “Configuration/Status Register 2 \(CSR2\)”](#).

### 3.4 Debug Mode

The debug interface is used to program a bootloader or user application program into the flash program memory before the MCU is operated in run mode for the first time. When a MCF51AG128 series MCU is shipped from the Freescale Semiconductor factory, the flash program memory is erased by default unless

specifically noted, so there is no program that could be executed in run mode until the flash memory is initially programmed. The debug interface can also be used to erase and reprogram the flash memory after it has been previously programmed.

For additional information about the debug interface, refer to [Chapter 26, “Version 1 ColdFire Debug \(CF1\\_DEBUG\).”](#)

### 3.5 Secure Mode

While the MCU is in secure mode, there are severe restrictions where debug commands can be used. In this mode, only the upper byte of the CPU's XCSR, CSR2, and CSR3 registers can be accessed. See [Chapter 26, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for details.

### 3.6 Run Mode

Run mode is the normal operating mode for the MCF51AG128 series MCUs. This mode is selected when the BKGD/MS pin is high at the rising edge of the internal  $\overline{\text{RESET}}$  signal. Upon exiting reset, the CPU fetches the supervisor SP and PC from locations 0x(00)00\_0000 and 0x(00)00\_0004 in the memory map and executes code starting at the newly set PC value.

### 3.7 Wait Mode

Wait mode is entered by executing a STOP instruction after configuring the device as per [Table 3-1](#). That is, if the WAITE control bit is set when STOP is executed, the wait mode is entered. Upon execution of the STOP instruction, the CPU enters a low-power state in which it is not clocked.

The V1 ColdFire core does not differentiate between stop and wait modes. The difference between the two is at the device level. In stop mode, most peripheral clocks are shut down; in wait mode, they continue to run.

XCSR[ENBDM] must be set prior to entering wait mode if the device is required to respond to BDM commands once in wait mode.

The low voltage detector, if enabled, can be configured to interrupt the CPU and exit wait mode into run mode.

When an interrupt request occurs, the CPU exits wait mode and resumes processing, beginning with the stacking operations leading to the interrupt service routine.

### 3.8 Stop Modes

One of three stop modes are entered upon execution of a STOP instruction when SOPT1[STOPE] is set. SOPT1[WAITE] must be cleared. In stop3 mode, the bus and CPU clocks are halted. If the ENBDM bit is set prior to entering stop4, only the peripheral clocks are halted. The ICS module can be configured to leave the reference clocks running. See [Chapter 19, “Internal Clock Source \(S08ICSV3\),”](#) for more information.

**NOTE**

If neither the WAITE or STOPE bit is set when the CPU executes a STOP instruction, the MCU does not enter either of the stop modes and either initiates an illegal opcode reset (if CPUCR[IRD]=0) or generates an illegal instruction exception (otherwise), if enabled.

The stop modes are selected by setting the appropriate bits in the SPMSC2 register. [Table 3-1](#) shows all of the control bits that affect mode selection under various conditions. The selected mode is entered after the execution of a STOP instruction.

Most background commands are not available in stop mode. The memory-access-with-status commands do not allow memory access, but they report an error indicating that the MCU is in either stop or wait mode. The BACKGROUND command can be used to wake the MCU from stop4 and enter halt mode if the ENBDM bit was set prior to entering stop mode. After entering halt mode, all background commands are available.

**3.8.1 Stop2 Mode**

Stop2 mode is entered by executing a STOP instruction under the conditions shown in [Table 3-1](#). Most of the MCU internal circuitry turns off in stop2 mode, with the exception of the RAM. After entering stop2 mode, all I/O pin control signals are latched so that the pins retain their states during stop2 mode. Exiting from stop2 mode is performed by asserting either wakeup pin: RESET or IRQ.

**NOTE**

- The IRQ pin enable (IRQPE) control bit in IRQSC must be set, before entering stop2 mode for the IRQ pin to act as the wakeup source from stop2 mode.
- IRQ/TPMCLK always functions as an active-low wakeup input when the MCU is in stop2 mode, regardless of how the pin is configured before entering stop2 mode. The pullup on this pin is always disabled in stop2 mode. This pin must be driven or pulled high externally while in stop2 mode.

In addition, the RTC can wake the MCU from stop2 mode, if enabled and using the low power oscillator (LPO). If the RTC is using the external clock source (EREFSTEN = 1) or internal clock source (IREFSTEN = 1), then the RTC is disabled in stop2 mode.

After waking from stop2 mode, the MCU starts up as from a power-on reset (POR):

- All module control and status registers are reset.
- The LVD reset function is enabled. The MCU remains in the reset state if VDD is below the LVD trip point (low trip point selected due to POR).
- The CPU initiates the reset exception process.

After waking from stop2 mode, SPMSC2[PPDF] is also set. This flag directs the user code to go to a stop2 recovery routine. The PPDF remains set and the I/O pin states remain latched until a 1 is written to SPMSC2[PPDACK]. To maintain I/O states for pins that were configured as general-purpose I/O before

entering stop2 mode, software must restore the contents of the I/O port registers to the port registers before writing to SPMSC2[PPDACK]. If the port registers are not restored from RAM before writing to SPMSC2[PPDACK], the pins then switch to their reset states when SPMSC2[PPDACK] is written.

For pins that were configured as peripheral I/O, software must reconfigure the peripheral module that interfaces to the pin before writing to SPMSC2[PPDACK]. If the peripheral module is not enabled before writing to SPMSC2[PPDACK], the pins are controlled by their associated port control registers when the I/O latches are opened.

### 3.8.2 Stop3 Mode

Stop3 mode is entered by executing a STOP instruction under the conditions shown in [Table 3-1](#). The states of all of the internal registers and logic, RAM contents, and I/O pin states are maintained. Stop3 mode can be exited by asserting  $\overline{\text{RESET}}$ , or by an interrupt from one of the following sources: RTC, SCI edge detect interrupt, IRQ, GPIO, or HSCMP. If stop3 mode is exited by the  $\overline{\text{RESET}}$  pin, the MCU is then reset and operation resumes after processing the reset exception. Exit by means of one of the internal interrupt sources results in the MCU taking the appropriate interrupt vector.

#### NOTE

- For the DMA enabled peripherals, only the interrupt signal can wake the MCU out of stop3 and stop4 modes. Make sure to clear DMAEN bit before entering stop3 and stop4 modes.
- When using the pin interrupt function to bring the MCU out of stop3 and stop4 modes, the PTxIE bit must be set. Make sure the digital filter on the GPIO pin is disabled (bypass mode) or operated by the LPO clock.
- If wakeup from stop3/stop4 happens via an interrupt masked (interrupt priority less than SR[I]), the interrupt causes PMC and clock to wake, but the core remains in stop modes.

### 3.8.3 Stop4 Mode

Stop4 mode is entered by executing a STOP instruction under the conditions shown in [Table 3-1](#). The states of all of the internal registers and logic, RAM contents, and I/O pin states are maintained. Stop4 can be exited by asserting  $\overline{\text{RESET}}$ , or by an interrupt from one of the following sources: RTC, SCI edge detect interrupt, LVD, ADC, IRQ, GPIO, or HSCMP. If stop4 is exited by means of the  $\overline{\text{RESET}}$  pin, the MCU is then reset and operation resumes after processing the reset exception. Exit by means of one of the internal interrupt sources results in the MCU taking the appropriate interrupt vector.

#### 3.8.3.1 LVD Enabled in Stop Mode

The LVD is capable of generating either an interrupt or a reset when the supply voltage drops below the LVD voltage. If the LVD is enabled in stop (SPMSC1[LVDE] && SPMSC1[LVDSE] = 1) at the time the CPU executes a STOP instruction, the voltage regulator then remains active during stop mode. If the user attempts to enter stop2 mode with the LVD enabled for stop mode, the MCU enters stop4 mode instead.



### 3.9 On-Chip Peripheral Modules in Stop and Wait Modes

When the MCU enters any stop mode (WAIT not included), system clocks to the internal peripheral modules stop. Even in the exception case of ENBDM = 1, where clocks to the background debug logic continue to operate, clocks to the peripheral systems are halted to reduce power consumption. Refer to [Section 3.8.1, “Stop2 Mode,”](#) and [Section 3.8.2, “Stop3 Mode,”](#) for specific information on system behavior in stop modes.

When the MCU enters wait mode, system clocks to the internal peripheral modules continue based on the settings of the clock gating control registers (SCGCx).

[Table 3-2](#) defines terms used in [Table 3-3](#) to describe operation of components on the chip in the various low power modes.

**Table 3-2. Abbreviations used in [Table 3-3](#)**

Voltage Regulator	Clocked <sup>1</sup>	Not Clocked
Full Regulation	FullOn	FullNoClk FullADACK <sup>2</sup>
Loose Regulation	SoftOn <sup>3</sup>	SoftNoClk Disabled SoftADACK <sup>4</sup>
Off	N/A	Off

<sup>1</sup> Subject to module enables settings of System Clock Gating Control Registers 1 and 2 (SCGC1 and SCGC2).

<sup>2</sup> This ADC-specific mode defines where the device is fully regulated and the normal peripheral clock is stopped. In this case, the ADC can continue to run using its internally generated asynchronous ADACK clock.

<sup>3</sup> Analog modules must be in their low power mode when the device is operated in this state.

<sup>4</sup> This ADC-specific mode defines where the device is in soft regulation and the normal peripheral clock is stopped. In this case, the ADC can only run using its low power mode and internally generated asynchronous ADACK clock.

**Table 3-3. Low Power Mode Behavior**

Peripheral	Mode			
	STOP2	STOP3	STOP4	WAIT
ADC <sup>1,2</sup>	Off	SoftADACK (Wakeup)	FULLADACK (Wakeup)	FullOn
BDC	Off	SoftOn	On	FullOn
CF1CORE	Off	SoftNoClk	FullNoClk	FullNoClk
CRC	Off	SoftNoClk	FullNoClk	FullOn
Crystal Oscillator (XOSC) <sup>3</sup>	Off	All Modes	All Modes	All Modes
DACx	Off	SoftNoClk	FullNoClk	FullOn

**Table 3-3. Low Power Mode Behavior (continued)**

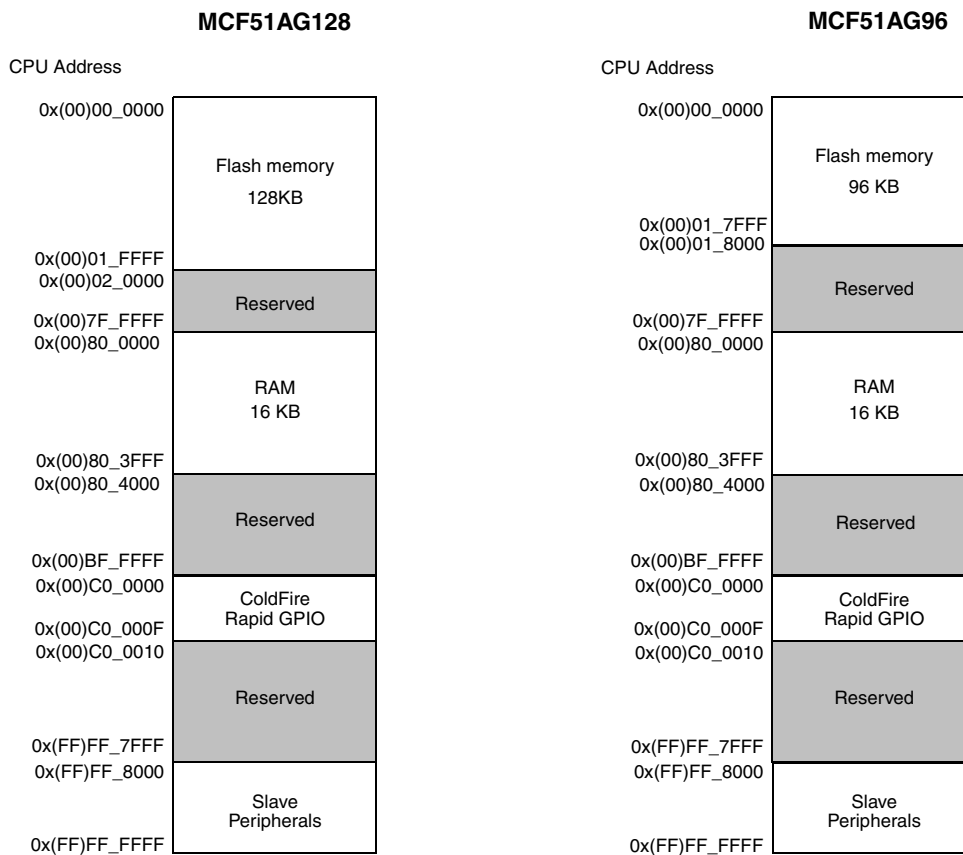
Peripheral	Mode			
	STOP2	STOP3	STOP4	WAIT
<b>DMA</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>EWM</b>	Soft Regulation, LPOCLK	SoftOn LPOCLK	Full Regulation LPOCLK,	FullOn
<b>Flash</b>	Off	SoftNoClk	FullNoClk	FullNoClk
<b>FTMx</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>GPI/O Pins</b>	States Held	SoftNoClk	FullNoClk	FullOn
<b>HSCMPx</b>	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
<b>ICS</b>	Off	STOP or BLPE <sup>4</sup>	STOP or any mode	Any mode
<b>iEvent</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>IICx</b>	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
<b>IRQ</b>	Off (Wakeup via POR) <sup>5</sup>	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
<b>LVD/LVW</b>	Off	Disabled	On (Wakeup)	FullOn
<b>PDB</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>Port I/O Registers</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>RAM</b>	SoftNoClk	SoftNoClk	FullNoClk	FullNoClk
<b>RTC</b>	Soft Regulation, LPOCLK if enabled (Wakeup via POR)	SoftOn, LPOCLK, OSCOUT, or ICSIRCLK only (Wake-up)	Full Regulation, LPOCLK, OSCOUT, or ICSIRCLK only (Wake-up)	FullOn
<b>SCIx</b>	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
<b>SPIx</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>TPM3</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>Voltage Regulator / PMC</b>	Partial Shutdown. 1kHz osc if enabled	Loose Regulation. 1kHz osc if enabled	Full Regulation 1kHz osc on	FullOn 1kHz osc on
<b>WDOG</b>	Off	SoftNoClk	FullNoClk	FullOn

- <sup>1</sup> LP mode for the ADC is invoked by setting ADLPC=1. The ADACK is selected via the ADCCFG[ADICLK] field in the ADC. See [Chapter 10, “Analog-to-Digital Converter \(S08ADC12V3\),”](#) for details.
- <sup>2</sup> LVD must be enabled to run in stop if converting the bandgap channel.
- <sup>3</sup> For crystal oscillator in stop3 mode, RANGE=0 and HGO=0 are the recommended settings to consume less power.
- <sup>4</sup> BLPE refers to the ICS “Bypassed Low Power External” state. See [Chapter 19, “Internal Clock Source \(S08ICSV3\),”](#) for more details.
- <sup>5</sup> The RESET pin also has a direct connection to the on-chip regulator wakeup input. Asserting a low on this pin while in STOP2 triggers the PMC to wake. As a result, the device undergoes a power-on-reset sequence.

# Chapter 4 Memory

## 4.1 MCF51AG128 Series Memory Map

As shown in [Figure 4-1](#), on-chip memory in the MCF51AG128 series microcontrollers consists of RAM and flash program memory for nonvolatile data storage, input/output, and control/status registers.



**Figure 4-1. MCF51AG128 Series Memory Maps**

Regions within the memory map are subject to restrictions with regard to the types of CPU accesses allowed. These are outlined in [Table 4-1](#). Non-supported access types terminate the bus cycle with an error (and would typically generate a system reset in response to the error termination).

Table 4-1. CPU Access Type Allowed by Region

Base Address	Region	Read			Write		
		Byte	Word	Long	Byte	Word	Long
0x(00)00_0000	Flash	•	•	•	—	—	•
0x(00)80_0000	RAM	•	•	•	•	•	•
0x(00)C0_0000	Rapid GPIO	•	•	•	•	•	•
0x(FF)FF_8000	Peripherals	•	•	•	•	•	•

Consistent with past ColdFire devices, flash configuration data is located at 0x(00)00\_0400. The section of memory at 0x(00)C0\_0000 is assigned for use by the ColdFire Rapid GPIO module. See [Table 4-7](#) for the rapid GPIO memory map and [Chapter 9, “Rapid GPIO \(RGPIO\),”](#) for further details on the module.

The MCF51AG128 series microcontrollers use an 8-bit peripheral bus. The bus bridge from the ColdFire system bus to the peripheral bus is capable of serializing 16-bit access into two 8-bit accesses and 32-bit access into four 8-bit accesses. This can be used to speed access to properly aligned peripheral registers.

### NOTE

Not all peripheral registers are aligned to take advantage of this feature.  
ADC registers can be accessed using 16-bit instruction only.

CPU accesses to those parts of the memory map marked as reserved in [Figure 4-1](#) result in an illegal address reset if CPUCR[ARD] is cleared or an address error exception if CPUCR[ARD] is set.

The lower 32 KB of flash memory and slave peripherals section of the memory map are most efficiently accessed using the ColdFire absolute short addressing mode. RAM is most efficiently accessed using the A5-relative addressing mode (address register indirect with displacement mode).

## 4.2 Detailed Register Addresses and Bit Assignments

The slave peripherals section of the memory map is further broken into the following sub-sections:

0x(FF)FF_8000 - 0x(FF)FF_81FF	Peripheral registers
0x(FF)FF_9800 - 0x(FF)FF_99FF	High-page registers
0x(FF)FF_9A00 - 0x(FF)FF_9BFF	iEvent controller
0x(FF)FF_E400 - 0x(FF)FF_E4FF	DMA controller
0x(FF)FF_FFC0 - 0x(FF)FF_FFFF	Interrupt controller

The ColdFire interrupt controller module is mapped in the peripheral space and occupies a 64-byte space at the upper end of memory. Accordingly, its address decode is defined as 0x(FF)FF\_FFC0–0x(FF)FF\_FFFF. This 64-byte space includes the program-visible interrupt controller registers as well as the space used for interrupt acknowledge (IACK) cycles.

There is a nonvolatile register area consisting of a block of 16 bytes in flash memory at 0x(00)00\_0400–0x(00)00\_040F. Nonvolatile register locations include:

- NVPROT and NVOPT are loaded into working registers at reset

- An 8-byte backdoor comparison key that optionally allows you to gain controlled access to secure memory

Because the nonvolatile register locations are flash memory, they must be erased and programmed like other flash memory locations.

In [Table 4-3](#), [Table 4-4](#), [Table 4-5](#), [Table 4-7](#), [Table 4-8](#), and [Table 4-9](#), the register names in column two are shown in bold to set them apart from the bit names to the right. Cells that are not associated with named bits are shaded. A shaded cell with a 0 indicates this unused bit always reads as a 0. Shaded cells with dashes indicate unused or reserved bit locations that could read as 1s or 0s. When writing to these bits, write a 0 unless otherwise specified.

Remember that ColdFire has a big endian byte addressable memory architecture. The most significant byte of each address is the lowest number as shown in [Table 4-2](#). Multi-byte operands (for example, 16-bit words and 32-bit longwords) are referenced using an address pointing to the most significant (first) byte.

**Table 4-2. ColdFire Memory Organization**

31	24	23	16	15	8	7	0
Longword 0x(00)00_0000							
Word 0x(00)00_0000				Word 0x(00)00_0002			
Byte 0x(00)00_0000		Byte 0x(00)00_0001		Byte 0x(00)00_0002		Byte 0x(00)00_0003	
Longword 0x(00)00_0004							
Word 0x(00)00_0004				Word 0x(00)00_0006			
Byte 0x(00)00_0004		Byte 0x(00)00_0005		Byte 0x(00)00_0006		Byte 0x(00)00_0007	
Longword 0x(FF)FF_FFFC							
Word 0x(FF)FF_FFFC				Word 0x(FF)FF_FFFE			
Byte 0x(FF)FF_FFFC		Byte 0x(FF)FF_FFFD		Byte 0x(FF)FF_FFFE		Byte 0x(FF)FF_FFFF	

## 4.2.1 Peripherals I/O and Control Registers

[Table 4-3](#) is a summary of all user-accessible peripherals, registers, and control bits.

Table 4-3. Peripherals Register Summary (Sheet 1 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8000	PTAD	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0xFF_FF_8001	PTADD	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
0xFF_FF_8002	PTAPV	PTAPV7	PTAPV6	PTAPV5	PTAPV4	PTAPV3	PTAPV2	PTAPV1	PTAPV0
0xFF_FF_8003	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8004	PTBD	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
0xFF_FF_8005	PTBDD	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
0xFF_FF_8006	PTBPV	PTBPV7	PTBPV6	PTBPV5	PTBPV4	PTBPV3	PTBPV2	PTBPV1	PTBPV0
0xFF_FF_8007	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8008	PTCD	—	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0xFF_FF_8009	PTCDD	—	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
0xFF_FF_800A	PTCPV	—	PTCPV6	PTCPV5	PTCPV4	PTCPV3	PTCPV2	PTCPV1	PTCPV0
0xFF_FF_800B	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_800C	PTDD	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0xFF_FF_800D	PTDDD	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
0xFF_FF_800E	PTDPV	PTDPV7	PTDPV6	PTDPV5	PTDPV4	PTDPV3	PTDPV2	PTDPV1	PTDPV0
0xFF_FF_800F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8010	PTED	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
0xFF_FF_8011	PTEDD	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
0xFF_FF_8012	PTEPV	PTEPV7	PTEPV6	PTEPV5	PTEPV4	PTEPV3	PTEPV2	PTEPV1	PTEPV0
0xFF_FF_8013	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8014	PTFD	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
0xFF_FF_8015	PTFDD	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
0xFF_FF_8016	PTFPV	PTFPV7	PTFPV6	PTFPV5	PTFPV4	PTFPV3	PTFPV2	PTFPV1	PTFPV0
0xFF_FF_8017	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8018	PTGD	—	PTGD6	PTED5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
0xFF_FF_8019	PTGDD	—	PTGDD6	PTEDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
0xFF_FF_801A	PTGPV	—	PTGPV6	PTEPV5	PTGPV4	PTGPV3	PTGPV2	PTGPV1	PTGPV0
0xFF_FF_801B	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_801C	PTHD	—	PTHD6	PTHD5	PTHD4	PTHD3	PTHD2	PTHD1	PTHD0
0xFF_FF_801D	PTHDD	—	PTHDD6	PTHDD5	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
0xFF_FF_801E	PTHPV	—	PTHPV6	PTHPV5	PTHPV4	PTHPV3	PTHPV2	PTHPV1	PTHPV0
0xFF_FF_801F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8020	PTJD	PTJD7	PTJD6	PTJD5	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0
0xFF_FF_8021	PTJDD	PTJDD7	PTJDD6	PTJDD5	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0
0xFF_FF_8022	PTJPV	PTJPV7	PTJPV6	PTJPV5	PTJPV4	PTJPV3	PTJPV2	PTJPV1	PTJPV0
0xFF_FF_8023	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8024	ICSC1	CLKS		RDIV			IREFS	IRCLKEN	IREFSTEN
0xFF_FF_8025	ICSC2	BDIV		RANGE	HGO	LP	EREFS	ERCLKEN	EREFSSTEN
0xFF_FF_8026	ICSTRM	TRIM							
0xFF_FF_8027	ICSSC	DRS/DRST		DMX32	IREFST	CLKST		OSCINIT	FTRIM
0xFF_FF_8028	RTCSC	RTIF	RTCLKS		RTIE	RTCPS			
0xFF_FF_8029	RTCCNT	RTCCNT							

Table 4-3. Peripherals Register Summary (Sheet 2 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_802A	RTCMOD	RTCMOD							
0x(FF)FF_802B	RTCS1	DMAEN	—	—	—	—	—	—	—
0x(FF)FF_802C	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_802D	DAC1CTRL	DACEN	VRSEL	—	VOSEL				
0x(FF)FF_802E	DAC2CTRL	DACEN	VRSEL	—	VOSEL				
0x(FF)FF_802F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8030	ADCSC1A	0	0	0	0	0	0	0	0
0x(FF)FF_8031	ADCSC1A	COCO	AIEN	ADCO	ADCH				
0x(FF)FF_8032	ADCSC2	0	0	0	0	0	0	0	0
0x(FF)FF_8033	ADCSC2	ADACT	ADTRG	0	BB	DMAEN	0	REFSEL	
0x(FF)FF_8034– 0x(FF)FF_803B	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_803C	ADCCFG	0	0	0	0	0	0	0	0
0x(FF)FF_803D	ADCCFG	ADLPC	ADIV		ADLSMP	MODE		ADICLK	
0x(FF)FF_803E– 0x(FF)FF_8043	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8044	ADCSC1B	0	0	0	0	0	0	0	0
0x(FF)FF_8045	ADCSC1B	COCO	AIEN	ADCO	ADCH				
0x(FF)FF_8046	ADCRA	0	ADR11	ADR10	ADR9	ADR8	ADR7	ADR6	ADR5
0x(FF)FF_8047	ADCRA	ADR4	ADR3	ADR2	ADR1	ADR0	0	0	0
0x(FF)FF_8048	ADCRB	0	ADR11	ADR10	ADR9	ADR8	ADR7	ADR6	ADR5
0x(FF)FF_8049	ADCRB	ADR4	ADR3	ADR2	ADR1	ADR0	0	0	0
0x(FF)FF_804A– 0x(FF)FF_804F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8050	SCI1BDH	LBKDIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8051	SCI1BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8052	SCI1C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8053	SCI1C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_8054	SCI1S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_8055	SCI1S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_8056	SCI1C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_8057	SCI1D	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8058	SCI1MA1	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0x(FF)FF_8059	SCI1MA2	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0x(FF)FF_805A	SCI1C4	MAEN1	MAEN2	0	BRFA				
0x(FF)FF_805B	SCI1C5	TDMAS	0	RDMAS	0	0	0	0	0
0x(FF)FF_805C– 0x(FF)FF_805F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8060	SCI2BDH	LBKDIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8061	SCI2BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8062	SCI2C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8063	SCI2C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_8064	SCI2S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF



Table 4-3. Peripherals Register Summary (Sheet 3 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8065	SCI2S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0xFF_FF_8066	SCI2C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0xFF_FF_8067	SCI2D	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8068	SCI2MA1	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0xFF_FF_8069	SCI2MA2	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0xFF_FF_806A	SCI2C4	MAEN1	MAEN2	0	BRFA				
0xFF_FF_806B	SCI2C5	TDMAS	0	RDMAS	0	0	0	0	0
0xFF_FF_806C– 0xFF_FF_806F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8070	TPM3SC	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0xFF_FF_8071	TPM3CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8072	TPM3CNTL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8073	TPM3MODH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8074	TPM3MODL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8075	TPM3C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0xFF_FF_8076	TPM3C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8077	TPM3C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8078	TPM3C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0xFF_FF_8079	TPM3C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_807A	TPM3C1VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_807B– 0xFF_FF_807F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8080	FTM1SC	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0xFF_FF_8081	FTM1CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8082	FTM1CNTL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8083	FTM1MODH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8084	FTM1MODL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8085	FTM1C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	DMA
0xFF_FF_8086	FTM1C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8087	FTM1C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8088	FTM1C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	DMA
0xFF_FF_8089	FTM1C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_808A	FTM1C1VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_808B	FTM1C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	DMA
0xFF_FF_808C	FTM1C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_808D	FTM1C2VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_808E	FTM1C3SC	CH3F	CH3IE	MS3B	MS3A	ELS3B	ELS3A	0	DMA
0xFF_FF_808F	FTM1C3VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8090	FTM1C3VL	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8091	FTM1C4SC	CH4F	CH4IE	MS4B	MS4A	ELS4B	ELS4A	0	DMA
0xFF_FF_8092	FTM1C4VH	Bit 15	14	13	12	11	10	9	Bit 8
0xFF_FF_8093	FTM1C4VL	Bit 7	6	5	4	3	2	1	Bit 0

Table 4-3. Peripherals Register Summary (Sheet 4 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8094	FTM1C5SC	CH5F	CH5IE	MS5B	MS5A	ELS5B	ELS5A	0	DMA
0x(FF)FF_8095	FTM1C5VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8096	FTM1C5VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8097- 0x(FF)FF_809F	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_80A0	FTM1CNTINH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80A1	FTM1CNTINL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80A2	FTM1STATUS	—	—	CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
0x(FF)FF_80A3	FTM1MODE	FAULTIE	FAULTM		CAPTEST	PWMSYNC	WPDIS	INIT	FTMEN
0x(FF)FF_80A4	FTM1SYNC	SWSYNC	TRIG2	TRIG1	TRIG0	SYNCHOM	REINIT	CNTMAX	CNTMIN
0x(FF)FF_80A5	FTM1OUTINIT	—	—	CH5OI	CH4OI	CH3OI	CH2OI	CH1OI	CH0OI
0x(FF)FF_80A6	FTM1OUTMASK	—	—	CH5OM	CH4OM	CH3OM	CH2OM	CH1OM	CH0OM
0x(FF)FF_80A7	FTM1COMBINE0	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80A8	FTM1COMBINE1	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80A9	FTM1COMBINE2	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80AA	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80AB	FTM1DEADTIME	DTPS			DTVAL				
0x(FF)FF_80AC	FTM1EXTTRIG	TRIGF	INITTRIGEN	CH1TRIG	CH0TRIG	CH5TRIG	CH4TRIG	CH3TRIG	CH2TRIG
0x(FF)FF_80AD	FTM1POL	—	—	POL5	POL4	POL3	POL2	POL1	POL0
0x(FF)FF_80AE	FTM1FMS	FAULTF	WPEN	FAULTIN	0	FAULTF3	FAULTF2	FAULTF1	FAULTF0
0x(FF)FF_80AF	FTM1FILTER0	CH1FVAL				CH0FVAL			
0x(FF)FF_80B0	FTM1FILTER1	CH3FVAL				CH2FVAL			
0x(FF)FF_80B1	FTM1FLTFILTER	0	0	0	0	FFVAL			
0x(FF)FF_80B2	FTM1FLTCTRL	FFLTR3EN	FFLTR2EN	FFLTR1EN	FFLTROEN	FAULT3EN	FAULT2EN	FAULT1EN	FAULT0EN
0x(FF)FF_80B3- 0x(FF)FF_80BF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80C0	FTM2SC	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0x(FF)FF_80C1	FTM2CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80C2	FTM2CNTL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80C3	FTM2MODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80C4	FTM2MODL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80C5	FTM2C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	DMA
0x(FF)FF_80C6	FTM2C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80C7	FTM2C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80C8	FTM2C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	DMA
0x(FF)FF_80C9	FTM2C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80CA	FTM2C1VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80CB	FTM2C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	DMA
0x(FF)FF_80CC	FTM2C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80CD	FTM2C2VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80CE	FTM2C3SC	CH3F	CH3IE	MS3B	MS3A	ELS3B	ELS3A	0	DMA
0x(FF)FF_80CF	FTM2C3VH	Bit 15	14	13	12	11	10	9	Bit 8

Table 4-3. Peripherals Register Summary (Sheet 5 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80D0	FTM2C3VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80D1	FTM2C4SC	CH4F	CH4IE	MS4B	MS4A	ELS4B	ELS4A	0	DMA
0x(FF)FF_80D2	FTM2C4VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80D3	FTM2C4VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80D4	FTM2C5SC	CH5F	CH5IE	MS5B	MS5A	ELS5B	ELS5A	0	DMA
0x(FF)FF_80D5	FTM2C5VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80D6	FTM2C5VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80D7- 0x(FF)FF_80DF	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_80E0	FTM2CNTINH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80E1	FTM2CNTINL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80E2	FTM2STATUS	—	—	CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
0x(FF)FF_80E3	FTM2MODE	FAULTIE	FAULTM		CAPTEST	PWMSYNC	WPDIS	INIT	FTMEN
0x(FF)FF_80E4	FTM2SYNC	SWSYNC	TRIG2	TRIG1	TRIG0	SYNCHOM	REINIT	CNTMAX	CNTMIN
0x(FF)FF_80E5	FTM2OUTINIT	—	—	CH5OI	CH4OI	CH3OI	CH2OI	CH1OI	CH0OI
0x(FF)FF_80E6	FTM2OUTMASK	—	—	CH5OM	CH4OM	CH3OM	CH2OM	CH1OM	CH0OM
0x(FF)FF_80E7	FTM2COMBINE0	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80E8	FTM2COMBINE1	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80E9	FTM2COMBINE2	0	FAULTEN	SYNCEN	DTEN	—	—	COMP	COMBINE
0x(FF)FF_80EA	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80EB	FTM2DEADTIME	DTPS			DTVAL				
0x(FF)FF_80EC	FTM2EXTTRIG	TRIGF	INITTRIGEN	CH1TRIG	CH0TRIG	CH5TRIG	CH4TRIG	CH3TRIG	CH2TRIG
0x(FF)FF_80ED	FTM2POL	—	—	POL5	POL4	POL3	POL2	POL1	POL0
0x(FF)FF_80EE	FTM2FMS	FAULTF	WPEN	FAULTIN	0	FAULTF3	FAULTF2	FAULTF1	FAULTF0
0x(FF)FF_80EF	FTM2FILTER0	CH1FVAL				CH0FVAL			
0x(FF)FF_80F0	FTM2FILTER1	CH3FVAL				CH2FVAL			
0x(FF)FF_80F1	FTM2FLTFILTER	0	0	0	0	FFVAL			
0x(FF)FF_80F2	FTM2FLTCTRL	FFLTR3EN	FFLTR2EN	FFLTR1EN	FFLTR0EN	FAULT3EN	FAULT2EN	FAULT1EN	FAULT0EN
0x(FF)FF_80F3- 0x(FF)FF_80FF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8100	PTAPUE	PTAPUE7	PTAPUE6	PTAPUE5	PTAPUE4	PTAPUE3	PTAPUE2	PTAPUE1	PTAPUE0
0x(FF)FF_8101	PTAPUS	PTAPUS7	PTAPUS6	PTAPUS5	PTAPUS4	PTAPUS3	PTAPUS2	PTAPUS1	PTAPUS0
0x(FF)FF_8102	PTADS	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
0x(FF)FF_8103	PTASRE	PTASRE7	PTASRE6	PTASRE5	PTASRE4	PTASRE3	PTASRE2	PTASRE1	PTASRE0
0x(FF)FF_8104	PTAPFE	PTAPFE7	PTAPFE6	PTAPFE5	PTAPFE4	PTAPFE3	PTAPFE2	PTAPFE1	PTAPFE0
0x(FF)FF_8105	PTAIC	PTADMAEN	—	—	—	—	—	PTAIE	PTAMOD
0x(FF)FF_8106	PTAIBE	PTAIBE7	PTAIBE6	PTAIBE5	PTAIBE4	PTAIBE3	PTAIBE2	PTAIBE1	PTAIBE0
0x(FF)FF_8107	PTAIF	PTAIF7	PTAIF6	PTAIF5	PTAIF4	PTAIF3	PTAIF2	PTAIF1	PTAIF0
0x(FF)FF_8108	PTAIES	PTAEDG7	PTAEDG6	PTAEDG5	PTAEDG4	PTAEDG3	PTAEDG2	PTAEDG1	PTAEDG0
0x(FF)FF_8109	PTADFE	PTADFE7	PTADFE6	PTADFE5	PTADFE4	PTADFE3	PTADFE2	PTADFE1	PTADFE0
0x(FF)FF_810A	PTADFC	PTACLKS	—	—	PTAFF4	PTAFF3	PTAFF2	PTAFF1	PTAFF0

Table 4-3. Peripherals Register Summary (Sheet 6 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_810B- 0x(FF)FF_810F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8110	PTBPUE	PTBPUE7	PTBPUE6	PTBPUE5	PTBPUE4	PTBPUE3	PTBPUE2	PTBPUE1	PTBPUE0
0x(FF)FF_8111	PTBPUS	PTBPUS7	PTBPUS6	PTBPUS5	PTBPUS4	PTBPUS3	PTBPUS2	PTBPUS1	PTBPUS0
0x(FF)FF_8112	PTBDS	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
0x(FF)FF_8113	PTBSRE	PTBSRE7	PTBSRE6	PTBSRE5	PTBSRE4	PTBSRE3	PTBSRE2	PTBSRE1	PTBSRE0
0x(FF)FF_8114	PTBPFE	PTBPFE7	PTBPFE6	PTBPFE5	PTBPFE4	PTBPFE3	PTBPFE2	PTBPFE1	PTBPFE0
0x(FF)FF_8115	PTBIC	PTBDMAEN	—	—	—	—	—	PTBIE	PTBMOD
0x(FF)FF_8116	PTBIPE	PTBIPE7	PTBIPE6	PTBIPE5	PTBIPE4	PTBIPE3	PTBIPE2	PTBIPE1	PTBIPE0
0x(FF)FF_8117	PTBIF	PTBIF7	PTBIF6	PTBIF5	PTBIF4	PTBIF3	PTBIF2	PTBIF1	PTBIF0
0x(FF)FF_8118	PTBIES	PTBEDG7	PTBEDG6	PTBEDG5	PTBEDG4	PTBEDG3	PTBEDG2	PTBEDG1	PTBEDG0
0x(FF)FF_8119	PTBDFE	PTBDFE7	PTBDFE6	PTBDFE5	PTBDFE4	PTBDFE3	PTBDFE2	PTBDFE1	PTBDFE0
0x(FF)FF_8119A	PTBDFC	PTBCLKS	—	—	PTBFF4	PTBFF3	PTBFF2	PTBFF1	PTBFF0
0x(FF)FF_811B- 0x(FF)FF_811F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8120	PTCPUE	—	PTCPUE6	PTCPUE5	PTCPUE4	PTCPUE3	PTCPUE2	PTCPUE1	PTCPUE0
0x(FF)FF_8121	PTCPUS	—	PTCPUS6	PTCPUS5	PTCPUS4	PTCPUS3	PTCPUS2	PTCPUS1	PTCPUS0
0x(FF)FF_8122	PTCDS	—	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
0x(FF)FF_8123	PTCSRE	—	PTCSRE6	PTCSRE5	PTCSRE4	PTCSRE3	PTCSRE2	PTCSRE1	PTCSRE0
0x(FF)FF_8124	PTCPFE	—	PTCPFE6	PTCPFE5	PTCPFE4	PTCPFE3	PTCPFE2	PTCPFE1	PTCPFE0
0x(FF)FF_8125	PTCIC	PTCDMAEN	—	—	—	—	—	PTCIE	PTCMOD
0x(FF)FF_8126	PTCIPE	—	PTCIPE6	PTCIPE5	PTCIPE4	PTCIPE3	PTCIPE2	PTCIPE1	PTCIPE0
0x(FF)FF_8127	PTCIF	—	PTCIF6	PTCIF5	PTCIF4	PTCIF3	PTCIF2	PTCIF1	PTCIF0
0x(FF)FF_8128	PTCIES	—	PTCEDG6	PTCEDG5	PTCEDG4	PTCEDG3	PTCEDG2	PTCEDG1	PTCEDG0
0x(FF)FF_8129	PTCDFE	—	PTCDFE6	PTCDFE5	PTCDFE4	PTCDFE3	PTCDFE2	PTCDFE1	PTCDFE0
0x(FF)FF_812A	PTCDFC	PTCCLKS	—	—	PTCFF4	PTCFF3	PTCFF2	PTCFF1	PTCFF0
0x(FF)FF_812B- 0x(FF)FF_812F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8130	PTDPUE	PTDPUE7	PTDPUE6	PTDPUE5	PTDPUE4	PTDPUE3	PTDPUE2	PTDPUE1	PTDPUE0
0x(FF)FF_8131	PTDPUS	PTDPUS7	PTDPUS6	PTDPUS5	PTDPUS4	PTDPUS3	PTDPUS2	PTDPUS1	PTDPUS0
0x(FF)FF_8132	PTDDS	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
0x(FF)FF_8133	PTDSRE	PTDSRE7	PTDSRE6	PTDSRE5	PTDSRE4	PTDSRE3	PTDSRE2	PTDSRE1	PTDSRE0
0x(FF)FF_8134	PTDPFE	PTDPFE7	PTDPFE6	PTDPFE5	PTDPFE4	PTDPFE3	PTDPFE2	PTDPFE1	PTDPFE0
0x(FF)FF_8135	PTDIC	PTDDMAEN	—	—	—	—	—	PTDIE	PTDMOD
0x(FF)FF_8136	PTDIPE	PTDIPE7	PTDIPE6	PTDIPE5	PTDIPE4	PTDIPE3	PTDIPE2	PTDIPE1	PTDIPE0
0x(FF)FF_8137	PTDIF	PTDIF7	PTDIF6	PTDIF5	PTDIF4	PTDIF3	PTDIF2	PTDIF1	PTDIF0
0x(FF)FF_8138	PTDIES	PTDEDG7	PTDEDG6	PTDEDG5	PTDEDG4	PTDEDG3	PTDEDG2	PTDEDG1	PTDEDG0
0x(FF)FF_8139	PTDDFE	PTDDFE7	PTDDFE6	PTDDFE5	PTDDFE4	PTDDFE3	PTDDFE2	PTDDFE1	PTDDFE0
0x(FF)FF_813A	PTDDFC	PTDCLKS	—	—	PTDFF4	PTDFF3	PTDFF2	PTDFF1	PTDFF0
0x(FF)FF_813B- 0x(FF)FF_813F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8140	PTEPUE	PTEPUE7	PTEPUE6	PTEPUE5	PTEPUE4	PTEPUE3	PTEPUE2	PTEPUE1	PTEPUE0
0x(FF)FF_8141	PTEPUS	PTEPUS7	PTEPUS6	PTEPUS5	PTEPUS4	PTEPUS3	PTEPUS2	PTEPUS1	PTEPUS0

Table 4-3. Peripherals Register Summary (Sheet 7 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0xFF_FF_8142	PTEDS	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
0xFF_FF_8143	PTESRE	PTESRE7	PTESRE6	PTESRE5	PTESRE4	PTESRE3	PTESRE2	PTESRE1	PTESRE0
0xFF_FF_8144	PTEPFE	PTEPFE7	PTEPFE6	PTEPFE5	PTEPFE4	PTEPFE3	PTEPFE2	PTEPFE1	PTEPFE0
0xFF_FF_8145	PTEIC	PTEDMAEN	—	—	—	—	—	PTEIE	PTEMOD
0xFF_FF_8146	PTEIPE	PTEIPE7	PTEIPE6	PTEIPE5	PTEIPE4	PTEIPE3	PTEIPE2	PTEIPE1	PTEIPE0
0xFF_FF_8147	PTEIF	PTEIF7	PTEIF6	PTEIF5	PTEIF4	PTEIF3	PTEIF2	PTEIF1	PTEIF0
0xFF_FF_8148	PTEIES	PTEEDG7	PTEEDG6	PTEEDG5	PTEEDG4	PTEEDG3	PTEEDG2	PTEEDG1	PTEEDG0
0xFF_FF_8149	PTEDFE	PTEDFE7	PTEDFE6	PTEDFE5	PTEDFE4	PTEDFE3	PTEDFE2	PTEDFE1	PTEDFE0
0xFF_FF_814A	PTEDFC	PTECLKS	—	—	PTEFF4	PTEFF3	PTEFF2	PTEFF1	PTEFF0
0xFF_FF_814B- 0xFF_FF_814F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8150	PTFPUE	PTFPUE7	PTFPUE6	PTFPUE5	PTFPUE4	PTFPUE3	PTFPUE2	PTFPUE1	PTFPUE0
0xFF_FF_8151	PTFPUS	PTFPUS7	PTFPUS6	PTFPUS5	PTFPUS4	PTFPUS3	PTFPUS2	PTFPUS1	PTFPUS0
0xFF_FF_8152	PTFDS	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
0xFF_FF_8153	PTFSRE	PTFSRE7	PTFSRE6	PTFSRE5	PTFSRE4	PTFSRE3	PTFSRE2	PTFSRE1	PTFSRE0
0xFF_FF_8154	PTFPFE	PTFPFE7	PTFPFE6	PTFPFE5	PTFPFE4	PTFPFE3	PTFPFE2	PTFPFE1	PTFPFE0
0xFF_FF_8155	PTFIC	PTFDMAEN	—	—	—	—	—	PTFIE	PTFMODE
0xFF_FF_8156	PTFIPE	PTFIPE7	PTFIPE6	PTFIPE5	PTFIPE4	PTFIPE3	PTFIPE2	PTFIPE1	PTFIPE0
0xFF_FF_8157	PTFIF	PTFIF7	PTFIF6	PTFIF5	PTFIF4	PTFIF3	PTFIF2	PTFIF1	PTFIF0
0xFF_FF_8158	PTFIES	PTFEDG7	PTFEDG6	PTFEDG5	PTFEDG4	PTFEDG3	PTFEDG2	PTFEDG1	PTFEDG0
0xFF_FF_8159	PTDFE	PTDFE7	PTDFE6	PTDFE5	PTDFE4	PTDFE3	PTDFE2	PTDFE1	PTDFE0
0xFF_FF_815A	PTDFC	PTFCLKS	—	—	PTFFF4	PTFFF3	PTFFF2	PTFFF1	PTFFF0
0xFF_FF_815B- 0xFF_FF_815F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8160	PTGPUE	—	PTGPUE6	PTGPUE5	PTGPUE4	PTGPUE3	PTGPUE2	PTGPUE1	PTGPUE0
0xFF_FF_8161	PTGPUS	—	PTGPUS6	PTGPUS5	PTGPUS4	PTGPUS3	PTGPUS2	PTGPUS1	PTGPUS0
0xFF_FF_8162	PTGDS	—	PTGDS6	PTEDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
0xFF_FF_8163	PTGSRE	—	PTGSRE6	PTESRE5	PTGSRE4	PTGSRE3	PTGSRE2	PTGSRE1	PTGSRE0
0xFF_FF_8164	PTGPFE	—	PTGPFE6	PTEPFE5	PTGPFE4	PTGPFE3	PTGPFE2	PTGPFE1	PTGPFE0
0xFF_FF_8165	PTGIC	PTGDMAEN	—	—	—	—	—	PTGIE	PTGMODE
0xFF_FF_8166	PTGIPE	—	PTGIPE6	PTEIPE5	PTGIPE4	PTGIPE3	PTGIPE2	PTGIPE1	PTGIPE0
0xFF_FF_8167	PTGIF	—	PTGIF6	PTEIF5	PTGIF4	PTGIF3	PTGIF2	PTGIF1	PTGIF0
0xFF_FF_8168	PTGIES	—	PTGEDG6	PTGEDG5	PTGEDG4	PTGEDG3	PTGEDG2	PTGEDG1	PTGEDG0
0xFF_FF_8169	PTGDFE	—	PTGDFE6	PTEDFE5	PTGDFE4	PTGDFE3	PTGDFE2	PTGDFE1	PTGDFE0
0xFF_FF_816A	PTGDFC	PTGCLKS	—	—	PTGFF4	PTGFF3	PTGFF2	PTGFF1	PTGFF0
0xFF_FF_816B- 0xFF_FF_816F	Reserved	—	—	—	—	—	—	—	—
0xFF_FF_8170	PTHPUE	—	PTHPUE6	PTHPUE5	PTHPUE4	PTHPUE3	PTHPUE2	PTHPUE1	PTHPUE0
0xFF_FF_8171	PTHPUS	—	PTHPUS6	PTHPUS5	PTHPUS4	PTHPUS3	PTHPUS2	PTHPUS1	PTHPUS0
0xFF_FF_8172	PTHDS	—	PTHDS6	PTHDS5	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
0xFF_FF_8173	PTHSRE	—	PTHSRE6	PTHSRE5	PTHSRE4	PTHSRE3	PTHSRE2	PTHSRE1	PTHSRE0
0xFF_FF_8174	PTHPFE	—	PTHPFE6	PTHPFE5	PTHPFE4	PTHPFE3	PTHPFE2	PTHPFE1	PTHPFE0

Table 4-3. Peripherals Register Summary (Sheet 8 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8175	PTHIC	PTHDMAEN	—	—	—	—	—	PTHIE	PTHMOD
0x(FF)FF_8176	PTHIPE	—	PTHIPE6	PTHIPE5	PTHIPE4	PTHIPE3	PTHIPE2	PTHIPE1	PTHIPE0
0x(FF)FF_8177	PTHIF	—	PTHIF6	PTHIF5	PTHIF4	PTHIF3	PTHIF2	PTHIF1	PTHIF0
0x(FF)FF_8178	PTHIES	—	PTHEDG6	PTHEDG5	PTHEDG4	PTHEDG3	PTHEDG2	PTHEDG1	PTHEDG0
0x(FF)FF_8179	PTHDFE	—	PTHDFE6	PTHDFE5	PTHDFE4	PTHDFE3	PTHDFE2	PTHDFE1	PTHDFE0
0x(FF)FF_817A	PTHDFC	PTHCLKS	—	—	PTHFF4	PTHFF3	PTHFF2	PTHFF1	PTHFF0
0x(FF)FF_817B- 0x(FF)FF_817F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8180	PTJPUE	PTJPUE7	PTJPUE6	PTJPUE5	PTJPUE4	PTJPUE3	PTJPUE2	PTJPUE1	PTJPUE0
0x(FF)FF_8181	PTJPUS	PTJPUS7	PTJPUS6	PTJPUS5	PTJPUS4	PTJPUS3	PTJPUS2	PTJPUS1	PTJPUS0
0x(FF)FF_8182	PTJDS	PTJDS7	PTJDS6	PTJDS5	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0
0x(FF)FF_8183	PTJSRE	PTJSRE7	PTJSRE6	PTJSRE5	PTJSRE4	PTJSRE3	PTJSRE2	PTJSRE1	PTJSRE0
0x(FF)FF_8184	PTJPFE	PTJPFE7	PTJPFE6	PTJPFE5	PTJPFE4	PTJPFE3	PTJPFE2	PTJPFE1	PTJPFE0
0x(FF)FF_8185	PTJIC	PTJDMAEN	—	—	—	—	—	PTJIE	PTJMOD
0x(FF)FF_8186	PTJIPE	PTJIPE7	PTJIPE6	PTJIPE5	PTJIPE4	PTJIPE3	PTJIPE2	PTJIPE1	PTJIPE0
0x(FF)FF_8187	PTJIF	PTJIF7	PTJIF6	PTJIF5	PTJIF4	PTJIF3	PTJIF2	PTJIF1	PTJIF0
0x(FF)FF_8188	PTJIES	PTJEDG7	PTJEDG6	PTJEDG5	PTJEDG4	PTJEDG3	PTJEDG2	PTJEDG1	PTJEDG0
0x(FF)FF_8189	PTJDFE	PTJDFE7	PTJDFE6	PTJDFE5	PTJDFE4	PTJDFE3	PTJDFE2	PTJDFE1	PTJDFE0
0x(FF)FF_818A	PTJDFC	PTJCLKS	—	—	PTJFF4	PTJFF3	PTJFF2	PTJFF1	PTJFF0
0x(FF)FF_818B- 0x(FF)FF_818F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8190	IICA1	AD							0
0x(FF)FF_8191	IICF	MULT			ICR				
0x(FF)FF_8192	IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	WUEN	DMAEN
0x(FF)FF_8193	IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
0x(FF)FF_8194	IICD	DATA							
0x(FF)FF_8195	IICC2	GCAEN	ADEXT	HDRS	SBRC	0	AD10	AD9	AD8
0x(FF)FF_8196	IICFLT	0	0	0	FLT				
0x(FF)FF_8197	IICSMB	FACK	ALERTEN	SIICAEN	TCKSEL	SLTF	SHTF1	SHTF2	SHTF2IE
0x(FF)FF_8198	IICA2	SAD							0
0x(FF)FF_8199	IICSLTH	SSLT[15:8]							
0x(FF)FF_819A	IICSLTL	SSLT[7:0]							
0x(FF)FF_819B- 0x(FF)FF_819F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_81A0	SPI1C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_81A1	SPI1C2	SPMIE	SPIMODE	TXDMAE	MODFEN	BIDIROE	RXDMAE	SPISWAI	SPC0
0x(FF)FF_81A2	SPI1BR	0	SPPR2	SPPR1	SPPR0	SPR3	SPR2	SPR1	SPR0
0x(FF)FF_81A3	SPI1S	SPRF	SPMF	SPTEF	MODF	0	0	0	0
0x(FF)FF_81A4	SPI1DH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_81A5	SPI1DL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81A6	SPI1MH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_81A7	SPI1ML	Bit 7	6	5	4	3	2	1	Bit 0

Table 4-3. Peripherals Register Summary (Sheet 9 of 9)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81A8	SPI2C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_81A9	SPI2C2	SPMIE	SPIMODE	TXDMAE	MODFEN	BIDIROE	RXDMAE	SPISWAI	SPC0
0x(FF)FF_81AA	SPI2BR	0	SPPR2	SPPR1	SPPR0	SPR3	SPR2	SPR1	SPR0
0x(FF)FF_81AB	SPI2S	SPRF	SPMF	SPTEF	MODF	0	0	0	0
0x(FF)FF_81AC	SPI2DH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_81AD	SPI2DL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81AE	SPI2MH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_81AF	SPI2ML	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81B0	HSCMP1CR0	0	FILTER_CNT			PMC		MMC	
0x(FF)FF_81B1	HSCMP1CR1	SE	WE	0	PMODE	INV	COS	OPE	EN
0x(FF)FF_81B2	HSCMP1FPR	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81B3	HSCMP1SCR	0	DMAEN	SMLEB	IER	IEF	CFR	CFF	COUT
0x(FF)FF_81B4	HSCMP1PCR	INPPE				INMPE			
0x(FF)FF_81B5– 0x(FF)FF_81B7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_81B8	HSCMP2CR0	0	FILTER_CNT			PMC		MMC	
0x(FF)FF_81B9	HSCMP2CR1	SE	WE	0	PMODE	INV	COS	OPE	EN
0x(FF)FF_81BA	HSCMP2FPR	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_81BB	HSCMP2SCR	0	DMAEN	SMLEB	IER	IEF	CFR	CFF	COUT
0x(FF)FF_81BC	HSCMP2PCR	INPPE				INMPE			
0x(FF)FF_81BD– 0x(FF)FF_81FF	Reserved	—	—	—	—	—	—	—	—

Table 4-4 shows the registers that equate to the high-page register of the MCF51AG128. These registers are typically accessed less often than other I/O and control registers.

Table 4-4. High-Page Equivalent Register Summary

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FE)FF_9800	SRS	POR	PIN	LOC	ILOP	ILAD	WDOG	LVD	0
0x(FE)FF_9801	SIMC	—	—	—	—	—	—	—	—
0x(FE)FF_9802	SOPT1	—	—	STOPE	WAITE	—	—	—	—
0x(FE)FF_9803	SMCLK	0	0	0	MPE	0	MCSEL		
0x(FE)FF_9804 – 0x(FE)FF_9805	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_9806	SDIDH	—	—	—	—	ID11	ID10	ID9	ID8
0x(FE)FF_9807	SDIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0x(FE)FF_9808	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_9809	SPMSC1	LVWF	LVWACK	LVWIE	LVDRE	LVDSE	LVDE	0	BGBE
0x(FE)FF_980A	SPMSC2	0	0	LVDV	LVWV	PPDF	PPDACK	0	PPDC
0x(FE)FF_980B	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_980C	SOPT2	CME	FTM2 SYNC	FTM1 SYNC	ACIC2	TPMCCF G	ACIC1	ADHWTS	
0x(FE)FF_980D	SCGC1	TPM3	FTM2	FTM1	ADC	DMA	IIC	SCI2	SCI1
0x(FE)FF_980E	SCGC2	CRC	FLS	IRQ	PDB	iEVT	RTC	SPI2	SPI1
0x(FE)FF_980F	SCGC3	—	—	—	HSCMP2	HSCMP1	DAC2	DAC1	EWM
0x(FE)FF_9810 – 0x(FE)FF_981F	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_9820	FCDIV	FDIVLD	PRDIV8	FDIV					
0x(FE)FF_9821	FOPT	KEYEN			0	0	0	0	SEC
0x(FE)FF_9822	FRSV0 (Reserved)	0	0	0	0	0	0	0	0
0x(FE)FF_9823	FCNFG	0	0	KEYACC	0	0	0	0	0
0x(FE)FF_9824	FPROT	FPS							FPOPEN
0x(FE)FF_9825	FSTAT	FCBEF	FCCF	FPVIOL	FACCER R	0	FBLANK	0	0
0x(FE)FF_9826	FCMD	0	FCMD						
0x(FE)FF_9827	FRSV1 (Reserved)	0	0	0	0	0	0	0	0
0x(FE)FF_9828	Reserved	0	0	0	0	0	0	0	0
0x(FE)FF_9829	Reserved	0	0	0	0	0	0	0	0
0x(FE)FF_982A	FRSV2 (Reserved)	0	0	0	0	0	0	0	0
0x(FE)FF_982B	FRSV3 (Reserved)	0	0	0	0	0	0	0	0
0x(FE)FF_982C – 0x(FE)FF_9830	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_9831	SAPE1	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
0x(FE)FF_9832	SAPE2	ADPC15	ADPC14	ADPC13	ADPC12	ADPC11	ADPC10	ADPC9	ADPC8
0x(FE)FF_9833	SAPE3	ADPC23	ADPC22	ADPC21	ADPC20	ADPC19	ADPC18	ADPC17	ADPC16
0x(FE)FF_9834	SEIS1	—	—	—	PTE	PTD	PTC	PTB	PTA
0x(FE)FF_9835	SEIS2	—	—	—	—	PTJ	PTH	PTG	PTF



Table 4-4. High-Page Equivalent Register Summary (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9836	SPINPS	—	—	FTM2CH5	FTM2CH4	FTM2CH3	FTM2CH2	TPM3CH1	TPM3CH0
0x(FF)FF_9837	SIMPTA	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9838	SIMPTB	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9839	SIMPTC	—	Bit 6	5	4	3	2	1	Bit 0
0x(FF)FF_983A	SIMPTD	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_983B	SIMPTE	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_983C	SIMPTF	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_983D	SIMPTG	—	Bit 6	5	4	3	2	1	Bit 0
0x(FF)FF_983E	SIMPTH	—	Bit 6	5	4	3	2	1	Bit 0
0x(FF)FF_983F	SIMPTJ	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9840	EWMCTRL	—	—	—	—	—	INEN	ASSIN	EWMMEN
0x(FF)FF_9841	EWMSEV	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9842	EWMCMPL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9843	EWMCMPH	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9844	IRQSC	0	IRQPDD	IRQEDG	IRQPE	IRQF	IRQACK	IRQIE	IRQMOD
0x(FF)FF_9835- 0x(FF)FF_983F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9850	CRCH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9851	CRCL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9852	Transpose	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9853	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9854	CRCL0	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9855	CRCL1	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9856	CRCL2	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9857	CRCL3	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9858- 0x(FF)FF_985F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9860	PDBCTRL1	LDMOD	BOS		ENB	AOS		ENA	LDOK
0x(FF)FF_9861	PDBCTRL2	PRESCALER			TRIGSEL			CONT	SWTRIG
0x(FF)FF_9862	PDBDLYAH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9863	PDBDLYAL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9864	PDBDLYBH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9865	PDBDLYBL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9866	PDBMODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9867	PDBMODL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9868	PDBCNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9869	PDBCNTL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_986A	PDBSCR	—	PDBEN	COF	—	DBF	—	DAF	—
0x(FF)FF_986B- 0x(FF)FF_987F	Reserved	—	—	—	—	—	—	—	—

Table 4-4. High-Page Equivalent Register Summary (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9880	WDOG_ST_CTRL_H	—	DISABLE_TESTWDOG	BYTESEL		TESTSEL	TESTWDOG	—	—
0x(FF)FF_9881		WAIT_EN	STOP_EN	DBG_EN	ALLOW_UPDATE	WIN_EN	IRQ_RST_EN	CLK_SRC	WDOG_EN
0x(FF)FF_9882	WDOG_ST_CTRL_L	INT_FLG	—	—	—	—	—	—	—
0x(FF)FF_9883		—	—	—	—	—	—	—	—
0x(FF)FF_9884	WDOG_TO_VAL_H	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9885		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9886	WDOG_TO_VAL_L	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9887		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9888	WDOG_WIN_H	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9889		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_988A	WDOG_WIN_L	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_988B		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_988C	WDOG_REFRESH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_988D		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_988E	WDOG_UNLOCK	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_988F		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9890	WDOG_TIMEOUT_H	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9891		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9892	WDOG_TIMEOUT_L	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9893		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9894	WDOG_RST_CNT	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9895		Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9896- 0x(FF)FF_99FF	Reserved	—	—	—	—	—	—	—	—

## 4.2.2 Intelligent Event Controller (iEvent) Memory Map

The iEvent is mapped into a 512-byte area starting at location 0x(FF)FF\_9A00. Its memory map is shown below in [Table 4-9](#).

Table 4-5. V1 ColdFire iEvent Memory Map

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9A00	IEVENT_DR0	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A
0x(FF)FF_9A01	IEVENT_DR1	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A
0x(FF)FF_9A02	IEVENT_DR2	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A
0x(FF)FF_9A03	IEVENT_DR3	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A
0x(FF)FF_9A04	IEVENT_DR4	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A
0x(FF)FF_9A05	IEVENT_DR5	Ev_Out	Output_FSM			In_D	In_C	In_B	In_A

Table 4-5. V1 ColdFire iEvent Memory Map (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9A06 – 0x(FF)FF_9A7F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9A80	IEVENT_CR0	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A81	IEVENT_CR1	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A82	IEVENT_CR2	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A83	IEVENT_CR3	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A84	IEVENT_CR4	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A85	IEVENT_CR5	RO	0	0	0	DDB	OSE	Type	
0x(FF)FF_9A86 – 0x(FF)FF_9AFF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9B00	IEVENT_IMXCR0	—				A_Select			
0x(FF)FF_9B01		—				B_Select			
0x(FF)FF_9B02		—				C_Select			
0x(FF)FF_9B03		—				D_Select			
0x(FF)FF_9B04	IEVENT_BFECCR0	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B05		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B06		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B07		PT3_AC		PT3_BC		PT3_CC		PT3_DC	
0x(FF)FF_9B08	IEVENT_IMXCR1	—				A_Select			
0x(FF)FF_9B09		—				B_Select			
0x(FF)FF_9B0A		—				C_Select			
0x(FF)FF_9B0B		—				D_Select			
0x(FF)FF_9B0C	IEVENT_BFECCR1	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B0D		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B0E		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B0F		PT3_AC		PT3_BC		PT3_CC		PT3_DC	
0x(FF)FF_9B10	IEVENT_IMXCR2	—				A_Select			
0x(FF)FF_9B11		—				B_Select			
0x(FF)FF_9B12		—				C_Select			
0x(FF)FF_9B13		—				D_Select			
0x(FF)FF_9B14	IEVENT_BFECCR2	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B15		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B16		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B17		PT3_AC		PT3_BC		PT3_CC		PT3_DC	
0x(FF)FF_9B18	IEVENT_IMXCR3	—				A_Select			
0x(FF)FF_9B19		—				B_Select			
0x(FF)FF_9B1A		—				C_Select			
0x(FF)FF_9B1B		—				D_Select			
0x(FF)FF_9B1C	IEVENT_BFECCR3	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B1D		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B1E		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B1F		PT3_AC		PT3_BC		PT3_CC		PT3_DC	

Table 4-5. V1 ColdFire iEvent Memory Map (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9B20	IEVENT_IMXCR4	—	—	—	—	A_Select			
0x(FF)FF_9B21		—	—	—	—	B_Select			
0x(FF)FF_9B22		—	—	—	—	C_Select			
0x(FF)FF_9B23		—	—	—	—	D_Select			
0x(FF)FF_9B24	IEVENT_BFECCR4	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B25		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B26		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B27		PT3_AC		PT3_BC		PT3_CC		PT3_DC	
0x(FF)FF_9B28	IEVENT_IMXCR5	—	—	—	—	A_Select			
0x(FF)FF_9B29		—	—	—	—	B_Select			
0x(FF)FF_9B2A		—	—	—	—	C_Select			
0x(FF)FF_9B2B		—	—	—	—	D_Select			
0x(FF)FF_9B2C	IEVENT_BFECCR5	PT0_AC		PT0_BC		PT0_CC		PT0_DC	
0x(FF)FF_9B2D		PT1_AC		PT1_BC		PT1_CC		PT1_DC	
0x(FF)FF_9B2E		PT2_AC		PT2_BC		PT2_CC		PT2_DC	
0x(FF)FF_9B2F		PT3_AC		PT3_BC		PT3_CC		PT3_DC	
0x(FF)FF_9B30 – 0x(FF)FF_9BFF	Reserved	—	—	—	—	—	—	—	—

### 4.2.3 Flash Module Reserved Memory Locations

Several reserved flash memory locations, shown in [Table 4-6](#), are used for storing values used by corresponding peripheral registers. These registers include an 8-byte backdoor key which can be used to gain access to secure memory resources. During reset events, the contents of the flash protection (NVPROT) byte and flash nonvolatile (NVOPT) byte in the reserved flash memory are transferred into the corresponding FPROT and FOPT registers in the high-page register area to control security and block protection options.

[Table 4-6](#) reflects the fact that ColdFire uses big endian addressing. See the ColdFire documentation for further details.

Table 4-6. Reserved Flash Memory Addresses <sup>1</sup>

Address	MSB (0x0)	(0x1)	(0x2)	LSB (0x3)
0x(00)00_03FC	Reserved		FTRIM (bit 0)	TRIM (bit 7:0)
0x(00)00_0400	Backdoor comparison key bytes zero through three			
	byte0	byte1	byte2	byte3
0x(00)00_0404	Backdoor comparison key bytes four through seven			
	byte4	byte5	byte6	byte7

**Table 4-6. Reserved Flash Memory Addresses (continued)<sup>1</sup>**

Address	MSB (0x0)	(0x1)	(0x2)	LSB (0x3)
0x(00)00_0408	Reserved			
0x(00)00_040C	Reserved	NVPROT	Reserved	NVOPT

<sup>1</sup> 0x(00)00\_03FC has been reserved for customers to use as a NVM placeholder for the TRIM value of the internal clock of the ICS.

The factory trim values are stored in the flash information row (IFR)<sup>1</sup> and are automatically loaded into the ICSTRM and ICSSC registers after any reset. The oscillator trim values stored in flash IFR are replicated at the TRIM and FTRIM locations of the flash as shown in Table 4-6. The oscillator trim values stored in the reserved flash memory address can be reprogrammed by third party programmers and must be copied into the corresponding ICS registers (ICSTRM and ICSSC) by user code to override the factory trim.

#### NOTE

When the MCU is in an active BDM, the trim value in the IFR is not loaded. Instead, the ICSTRM register resets to 0x80 and ICSSC[FTRIM] resets to zero.

Provided the key enable (KEYEN) bit is set, the 8-byte comparison key can be used to temporarily disengage memory security. This key mechanism can be accessed only through user code running in secure memory. (A security key cannot be entered directly through background debug commands.) This security key can be disabled completely by programming the KEYEN bit to 0. If the security key is disabled, the only way to disengage security is by mass-erasing the flash (normally through the background debug interface) and verifying that flash is blank.

### 4.2.4 ColdFire Rapid GPIO Memory Map

The rapid GPIO module is mapped into a 16-byte area starting at location 0x(00)C0\_0000. Its memory map is shown below in Table 4-7.

**Table 4-7. V1 ColdFire Rapid GPIO Memory Map**

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(00)C0_0000	RGPIO_DIR	Bit 15	14	13	12	11	10	9	8
0x(00)C0_0001		7	6	5	4	3	2	1	Bit 0
0x(00)C0_0002	RGPIO_DATA	Bit 15	14	13	12	11	10	9	8
0x(00)C0_0003		7	6	5	4	3	2	1	Bit 0
0x(00)C0_0004	RGPIO_ENB	Bit 15	14	13	12	11	10	9	8
0x(00)C0_0005		7	6	5	4	3	2	1	Bit 0

- IFR** — Nonvolatile information memory that can only be accessed during production test. During production test, system initialization, configuration, and test information is stored in the IFR. This information cannot be read or modified in normal user or background debug modes.

Table 4-7. V1 ColdFire Rapid GPIO Memory Map (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(00)C0_0006	RGPIO_CLR	Bit 15	14	13	12	11	10	9	8
0x(00)C0_0007		7	6	5	4	3	2	1	Bit 0
0x(00)C0_0008 – 0x(00)C0_0009	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000A	RGPIO_SET	Bit 15	14	13	12	11	10	9	8
0x(00)C0_000B		7	6	5	4	3	2	1	Bit 0
0x(00)C0_000C – 0x(00)C0_000D	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000E	RGPIO_TOG	Bit 15	14	13	12	11	10	9	8
0x(00)C0_000F		7	6	5	4	3	2	1	Bit 0

## 4.2.5 ColdFire Interrupt Controller Memory Map

The V1 ColdFire interrupt controller (CF1\_INTC) register map is sparsely-populated, but retains compatibility with earlier ColdFire interrupt controller definitions. The CF1\_INTC occupies the upper 64 bytes of the system address space and all memory locations are accessed as 8-bit (byte) operands.

Table 4-8. V1 ColdFire Interrupt Controller Memory Map

Address	Register Name	msb Bit Number lsb							
0x(FF)FF_FFC0 – 0x(FF)FF_FFC7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFC8	INTC_IMRH	0	0	0	0	0	0	0	0
0x(FF)FF_FFC9		0	0	0	0	0	0	0	0
0x(FF)FF_FFCA	INTC_IMRL	0	0	0	0	IMR43	IMR42	IMR41	IMR40
0x(FF)FF_FFCB		IMR39	IMR38	IMR37	IMR36	IMR35	IMR34	IMR33	IMR32
0x(FF)FF_FFCC	INTC_IMRL	IMR31	IMR30	IMR29	IMR28	IMR27	IMR26	IMR25	IMR24
0x(FF)FF_FFCD		IMR23	IMR22	IMR21	IMR20	IMR19	IMR18	IMR17	IMR16
0x(FF)FF_FFCE	INTC_FRC	IMR15	IMR14	IMR13	IMR12	IMR11	IMR10	IMR9	IMR8
0x(FF)FF_FFCF		IMR7	IMR6	IMR5	IMR4	IMR3	IMR2	IMR1	IMR0
0x(FF)FF_FFD0	INTC_FRC	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7
0x(FF)FF_FFD1 – 0x(FF)FF_FFD7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFD8	INTC_PL6P7	0	0	REQN					
0x(FF)FF_FFD9	INTC_PL6P6	0	0	REQN					
0x(FF)FF_FFDA	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFDB	INTC_WCR	ENB	0	0	0	0	MASK		
0x(FF)FF_FFDC	INTC_SIMR	0	SALL	SIMR					
0x(FF)FF_FFDD	INTC_CIMR	—	CALL	CIMR					
0x(FF)FF_FFDE	INTC_SFRC	—	—	SET					
0x(FF)FF_FFDF	INTC_CFRC	0	0	CLR					
0x(FF)FF_FFE0	INTC_SWIACK	0	VECN						

Table 4-8. V1 ColdFire Interrupt Controller Memory Map (continued)

Address	Register Name	msb	Bit Number						lsb
0x(FE)FF_FFE1 – 0x(FE)FF_FFE3	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFE4	INTC_LVL1IACK	0	VECN						
0x(FE)FF_FFE5 – 0x(FE)FF_FFE7	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFE8	INTC_LVL2IACK	0	VECN						
0x(FE)FF_FFE9 – 0x(FE)FF_FFEB	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFEc	INTC_LVL3IACK	0	VECN						
0x(FE)FF_FFED – 0x(FE)FF_FFEF	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFF0	INTC_LVL4IACK	0	VECN						
0x(FE)FF_FFF1 – 0x(FE)FF_FFF3	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFF4	INTC_LVL5IACK	0	VECN						
0x(FE)FF_FFF5 – 0x(FE)FF_FFF7	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFF8	INTC_LVL6IACK	0	VECN						
0x(FE)FF_FFF9 – 0x(FE)FF_FFfB	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_FFfC	INTC_LVL7IACK	0	VECN						
0x(FE)FF_FFfD – 0x(FE)FF_FFfF	Reserved	—	—	—	—	—	—	—	—

## 4.2.6 DMA Controller Memory Map

The DMA controller is mapped into a 512-byte area starting at location 0x(FE)FF\_E400. Its memory map is shown below in [Table 4-9](#).

Table 4-9. V1 ColdFire DMA Controller Memory Map

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FE)FF_E400	DMAREQC	—	—	—	—	DMAC0			
0x(FE)FF_E401		—	—	—	—	DMAC1			
0x(FE)FF_E402		—	—	—	—	DMAC2			
0x(FE)FF_E403		—	—	—	—	DMAC3			
0x(FE)FF_E404 – 0x(FE)FF_E4FF	Reserved	—	—	—	—	—	—	—	—
0x(FE)FF_E500	SAR0	—	—	—	—	—	—	—	—
0x(FE)FF_E501		SAR[23:0]							
0x(FE)FF_E502									
0x(FE)FF_E503									

Table 4-9. V1 ColdFire DMA Controller Memory Map (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_E504		—	—	—	—	—	—	—	—
0x(FF)FF_E505	<b>DAR0</b>	DAR[23:0]							
0x(FF)FF_E506									
0x(FF)FF_E507									
0x(FF)FF_E508	<b>DSR0</b>	0	CE	BES	BED	0	REQ	BSY	DONE
0x(FF)FF_E509	<b>BCR0</b>	BCR[23:0]							
0x(FF)FF_E50A									
0x(FF)FF_E50B									
0x(FF)FF_E50C	<b>DCR0</b>	EINT	ERQ	CS	AA	0	0	0	0
0x(FF)FF_E50D		0	SINC	SSIZE		DINC	DSIZE		START
0x(FF)FF_E50E		SMOD				DMOD			
0x(FF)FF_E50F		DREQ	0	LINKCC		LCH1		LCH2	
0x(FF)FF_E510		—	—	—	—	—	—	—	—
0x(FF)FF_E511	<b>SAR1</b>	SAR[23:0]							
0x(FF)FF_E512									
0x(FF)FF_E513									
0x(FF)FF_E514	<b>DAR1</b>	DAR[23:0]							
0x(FF)FF_E515									
0x(FF)FF_E516									
0x(FF)FF_E517	<b>DSR1</b>	0	CE	BES	BED	0	REQ	BSY	DONE
0x(FF)FF_E518	<b>BCR1</b>	BCR[23:0]							
0x(FF)FF_E51A									
0x(FF)FF_E51B									
0x(FF)FF_E51C	<b>DCR1</b>	EINT	ERQ	CS	AA	0	0	0	0
0x(FF)FF_E51D		0	SINC	SSIZE		DINC	DSIZE		START
0x(FF)FF_E51E		SMOD				DMOD			
0x(FF)FF_E51F		DREQ	0	LINKCC		LCH1		LCH2	
0x(FF)FF_E520		—	—	—	—	—	—	—	—
0x(FF)FF_E521	<b>SAR2</b>	SAR[23:0]							
0x(FF)FF_E522									
0x(FF)FF_E523									
0x(FF)FF_E524	<b>DAR2</b>	DAR[23:0]							
0x(FF)FF_E525									
0x(FF)FF_E526									
0x(FF)FF_E527	<b>DSR2</b>	0	CE	BES	BED	0	REQ	BSY	DONE
0x(FF)FF_E528	<b>BCR2</b>	BCR[23:0]							
0x(FF)FF_E529									
0x(FF)FF_E52A									
0x(FF)FF_E52B									



Table 4-9. V1 ColdFire DMA Controller Memory Map (continued)

Register Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_E52C	DCR2	EINT	ERQ	CS	AA	0	0	0	0
0x(FF)FF_E52D		0	SINC	SSIZE		DINC	DSIZE		START
0x(FF)FF_E52E		SMOD				DMOD			
0x(FF)FF_E52F		DREQ	0	LINKCC		LCH1		LCH2	
0x(FF)FF_E530	SAR3	—	—	—	—	—	—	—	—
0x(FF)FF_E531		SAR[23:0]							
0x(FF)FF_E532		SAR[23:0]							
0x(FF)FF_E533		SAR[23:0]							
0x(FF)FF_E534	DAR3	—	—	—	—	—	—	—	—
0x(FF)FF_E535		DAR[23:0]							
0x(FF)FF_E536		DAR[23:0]							
0x(FF)FF_E537		DAR[23:0]							
0x(FF)FF_E538	DSR3	0	CE	BES	BED	0	REQ	BSY	DONE
0x(FF)FF_E539	BCR3	BCR[23:0]							
0x(FF)FF_E53A		BCR[23:0]							
0x(FF)FF_E53B		BCR[23:0]							
0x(FF)FF_E53C		BCR[23:0]							
0x(FF)FF_E53D	DCR3	EINT	ERQ	CS	AA	0	0	0	0
0x(FF)FF_E53E		0	SINC	SSIZE		DINC	DSIZE		START
0x(FF)FF_E53F		SMOD				DMOD			
0x(FF)FF_E53F		DREQ	0	LINKCC		LCH1		LCH2	

### 4.3 RAM

An MCF51AG128 series microcontroller includes up to 16 KB of static RAM. RAM is most efficiently accessed using the A5-relative addressing mode (address register indirect with displacement mode). Any single bit in this area can be accessed with the bit manipulation instructions (BCLR, BSET, etc.).

At power-on, the contents of RAM are uninitialized. RAM data is unaffected by any reset provided that the supply voltage does not drop below the minimum value for RAM retention ( $V_{RAM}$ ).

### 4.4 Flash Memory

The flash memory is intended primarily for program storage and read-only data. In-circuit programming allows the operating program to be loaded into the flash memory after final assembly of the application product. It is possible to program the entire array through the single-wire background debug interface. Because no special voltages are needed for flash erase and programming operations, in-application programming is also possible through other software-controlled communication paths.

Flash memory is ideal for single-supply applications allowing for field reprogramming without requiring external high voltage sources for program or erase operations. The flash module includes a memory controller that executes commands to modify flash memory contents.

Array read access time is one bus cycle for bytes, aligned words, and aligned longwords. Multiple accesses are needed for misaligned word and longword operands. For flash memory, an erased bit reads 1 and a

programmed bit reads 0. It is not possible to read from a flash block while any command is executing on that specific flash block.

### CAUTION

A flash block address must be in the erased state before being programmed. Cumulative programming of bits within a flash block address is not allowed except for status field updates required in EEPROM emulation applications.

Flash memory on MCF51AG128 series MCUs must be programmed 32 bits at a time. The MCF51AG128 flash memory is organized as two 16-bit wide blocks interleaved to yield a 32-bit data path.

## 4.4.1 Features

Flash memory features include:

- Flash size
  - MCF51AG128: 131,072 bytes (128 sectors of 1024 bytes each)
  - MCF51AG96: 98,304 bytes (96 sectors of 1024 bytes each)
- Automated program and erase algorithm
- Fast program and sector erase operation
- Burst program command for faster flash array program times
- Single power supply program and erase
- Command interface for fast program and erase operation
- Up to 100,000 program and erase cycles at typical voltage and temperature
- Flexible block protection (on any 2 KB memory boundary)
- Security feature to prevent unauthorized access to on-chip memory and resources
- Auto power-down for low-frequency read accesses

## 4.4.2 Register Descriptions

The flash module contains a set of 16 control and status registers. Flash registers are byte accessible only. Detailed descriptions of each register bit are provided in the following sections.

### 4.4.2.1 Flash Clock Divider Register (FCDIV)

The FCDIV register controls the length of timed events in program and erase algorithms executed by the flash memory controller.

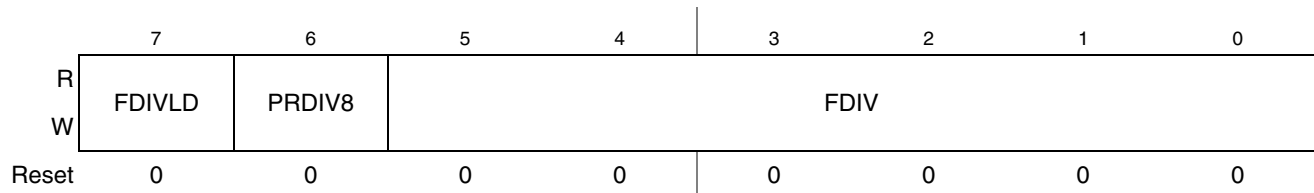


Figure 4-2. Flash Clock Divider Register (FCDIV)

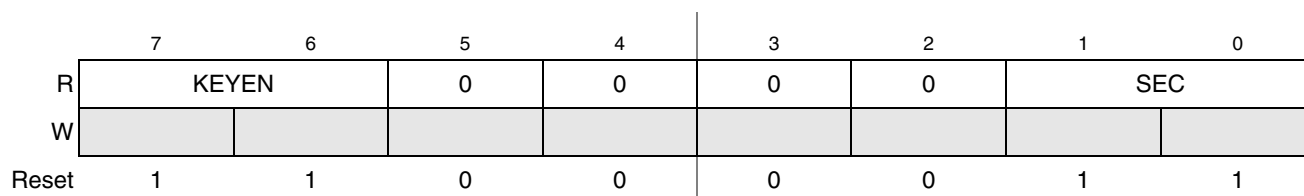
All bits in the FCDIV register are readable and writable with restrictions as determined by the value of FDIVLD when writing to the FCDIV register (see [Table 4-10](#)).

**Table 4-10. FCDIV Field Descriptions**

Field	Description
7 FDIVLD	<b>Clock Divider Load Control</b> — When writing to the FCDIV register for the first time after a reset, the value of the FDIVLD bit written controls the future ability to write to the FCDIV register: 0 Writing 0 to FDIVLD locks the FCDIV register contents; all future writes to FCDIV are ignored. 1 Writing 1 to FDIVLD keeps the FCDIV register writable; next write to FCDIV is allowed. When reading the FCDIV register, the value of the FDIVLD bit read indicates the following: 0 FCDIV register has not been written to since the last reset. 1 FCDIV register has been written to since the last reset.
6 PRDIV8	<b>Enable Prescaler by 8.</b> 0 The bus clock is directly fed into the clock divider. 1 The bus clock is divided by 8 before feeding into the clock divider.
5:0 FDIV[5:0]	<b>Clock Divider Bits</b> — The combination of PRDIV8 and FDIV[5:0] must divide the bus clock down to a frequency of 150 kHz–200 kHz. The minimum divide ratio is 2 (PRDIV8 = 0, FDIV = 0x01) and the maximum divide ratio is 512 (PRDIV8 = 1, FDIV = 0x3F). See <a href="#">Section 4.4.3.1, “Writing the FCDIV Register”</a> for more information.

#### 4.4.2.2 Flash Options Register (FOPT and NVOPT)

The FOPT register holds all bits associated with the security of the MCU and flash module.



**Figure 4-3. Flash Options Register (FOPT)**

All bits in the FOPT register are readable but are not writable.

The FOPT register is loaded from the flash configuration field (see [Section 4.2.3](#)) during the reset sequence.

**Table 4-11. FOPT Field Descriptions**

Field	Description
7–6 KEYEN	<b>Backdoor Key Security Enable Bits</b> — The KEYEN[1:0] bits define the enabling of backdoor key access to the Flash module. 00 Disabled 01 Disabled (Preferred KEYEN state to disable Backdoor Key Access) 10 Enabled 11 Disabled

Table 4-11. FOPT Field Descriptions (continued)

Field	Description
5–2	Reserved, must be cleared.
1–0 SEC	<b>Flash Security Bits</b> — The SEC[1:0] bits define the security state of the MCU. If the Flash module is unsecured using backdoor key access, the SEC[1:0] bits are forced to the unsecured state. 00 Unsecured 01 Unsecured 10 Secured 11 Unsecured

The security feature in the flash module is described in [Section 4.4.7, “Security”](#).

#### 4.4.2.3 Flash Configuration Register (FCNFG)

The FCNFG register gates the security backdoor writes.

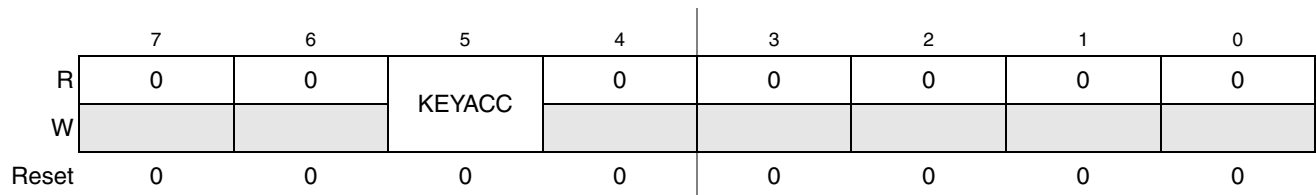


Figure 4-4. Flash Configuration Register (FCNFG)

KEYACC is readable and writable while all remaining bits read 0 and are not writable. KEYACC is only writable if KEYEN is set to the enabled state (see [Section 4.4.2.2, “Flash Options Register \(FOPT and NVOPT\)”](#)).

Table 4-12. FCNFG Field Descriptions

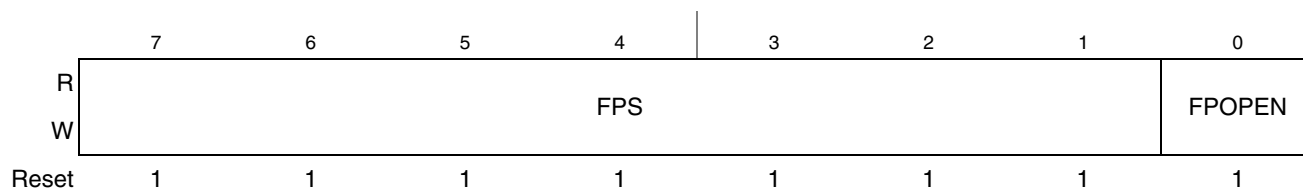
Field	Description
7–6	Reserved, must be cleared.
5 KEYACC	Enable Security Key Writing 0 Writes to the flash block are interpreted as the start of a command write sequence. 1 Writes to the flash block are interpreted as keys to open the backdoor.
4–0	Reserved, must be cleared.

#### NOTE

Flash array reads are allowed while KEYACC is set.

#### 4.4.2.4 Flash Protection Register (FPROT and NVPROT)

The FPROT register defines which flash sectors are protected against program or erase operations.



**Figure 4-5. Flash Protection Register (FPROT)**

FPROT bits are readable and writable as long as the size of the protected flash memory is being increased. Any write to FPROT that attempts to decrease the size of the protected flash memory is ignored.

During the reset sequence, the FPROT register is loaded from the flash protection byte in the flash configuration field (see [Section 4.2.3](#)). To change the flash protection loaded during the reset sequence, the flash sector containing the flash configuration field must be unprotected, then the flash protection byte must be reprogrammed.

Trying to alter data in any protected area in the flash memory results in a protection violation error and the FPVIOL flag is set in the FSTAT register. The mass erase of the flash array is not possible if any of the flash sectors contained in the flash array are protected.

**Table 4-13. FPROT Field Descriptions**

Field	Description
7–1 FPS	<b>Flash Protection Size</b> — With FPOPEN set, the FPS bits determine the size of the protected flash address range as shown in <a href="#">Table 4-14</a> .
0 FPOPEN	Flash Protection Open 0 Flash array fully protected. 1 Flash array protected address range determined by FPS bits.

**Table 4-14. Flash Protection Address Range**

FPS	FPOPEN	Protected Address Range Relative to Flash Array Base	Protected Size
—	0	0x0_0000–0x1_FFFF	128 KB
0x00–0x3F	1	0x0_0000–0x1_FFFF	128 KB
0x40		0x0_0000–0x1_F7FF	126 KB
0x41		0x0_0000–0x1_EFFF	124 KB
0x42		0x0_0000–0x1_E7FF	122 KB
0x43		0x0_0000–0x1_DFFF	120 KB
0x44		0x0_0000–0x1_D7FF	118 KB
0x45		0x0_0000–0x1_CFFF	116 KB
0x46		0x0_0000–0x1_C7FF	114 KB

Table 4-14. Flash Protection Address Range (continued)

FPS	FPOPEN	Protected Address Range Relative to Flash Array Base	Protected Size
0x47	1	0x0_0000–0x1_BFFF	112 KB
...		...	...
0x5B		0x0_0000–0x1_1FFF	72 KB
0x5C		0x0_0000–0x1_17FF	70 KB
0x5D		0x0_0000–0x1_0FFF	68 KB
0x5E		0x0_0000–0x1_07FF	66 KB
0x5F		0x0_0000–0x0_FFFF	64 KB
0x60		0x0_0000–0x0_F7FF	62 KB
0x61		0x0_0000–0x0_EFFF	60 KB
0x62		0x0_0000–0x0_E7FF	58 KB
0x63		0x0_0000–0x0_DFFF	56 KB
...		...	...
0x77		0x0_0000–0x0_3FFF	16 KB
0x78		0x0_0000–0x0_37FF	14 KB
0x79		0x0_0000–0x0_2FFF	12 KB
0x7A		0x0_0000–0x0_27FF	10 KB
0x7B		0x0_0000–0x0_1FFF	8 KB
0x7C		0x0_0000–0x0_17FF	6 KB
0x7D		0x0_0000–0x0_0FFF	4 KB
0x7E		0x0_0000–0x0_07FF	2 KB
0x7F	No Protection	0 KB	

#### 4.4.2.5 Flash Status Register (FSTAT)

The FSTAT register defines the operational status of the flash module.

	7	6	5	4	3	2	1	0
R	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
W								
Reset	1	1	0	0	0	0	0	0

Figure 4-6. Flash Status Register (FSTAT)

FCBEF, FPVIOL, and FACCERR are readable and writable; FBLANK is readable and not writable; remaining bits read 0 and are not writable.

Table 4-15. FSTAT Field Descriptions

Field	Description
7 FCBEF	<p><b>Command Buffer Empty Flag</b> — The FCBEF flag indicates that the command buffer is empty and a new command write sequence can be started when performing burst programming. Writing 0 to the FCBEF flag has no effect on FCBEF. Writing 0 to FCBEF after writing an aligned address to the flash array memory, but before FCBEF is cleared, aborts a command write sequence and cause the FACCERR flag to be set. Writing 0 to FCBEF outside of a command write sequence does not set the FACCERR flag. The FCBEF flag is cleared by writing 1 to FCBEF.</p> <p>0 Command buffers are full. 1 Command buffers are ready to accept a new command.</p>
6 FCCF	<p><b>Command Complete Flag</b> — The FCCF flag indicates that there are no more commands pending. The FCCF flag is cleared when FCBEF is cleared and sets automatically after completion of all active and pending commands. The FCCF flag does not set when an active program command completes and a pending burst program command is fetched from the command buffer. Writing to the FCCF flag has no effect on FCCF.</p> <p>0 Command in progress. 1 All commands are completed.</p>
5 FPVIOL	<p><b>Protection Violation Flag</b> —The FPVIOL flag indicates an attempt was made to program or erase an address in a protected area of the flash memory or flash IFR during a command write sequence. Writing 0 to the FPVIOL flag has no effect on FPVIOL. The FPVIOL flag is cleared by writing 1 to FPVIOL. When FPVIOL is set, it is not possible to launch a command or start a command write sequence.</p> <p>0 No protection violation detected. 1 Protection violation has occurred.</p>
4 FACCERR	<p><b>Access Error Flag</b> — The FACCERR flag indicates an illegal access has occurred to the flash memory or flash IFR caused by either a violation of the command write sequence (see <a href="#">Section 4.4.3.2, “Command Write Sequence”</a>), issuing an illegal flash command (see <a href="#">Section 4.4.2.6, “Flash Command Register (FCMD)”</a>), or the execution of a CPU STOP instruction while a command is executing (FCCF = 0). Writing 0 to the FACCERR flag has no effect on FACCERR. The FACCERR flag is cleared by writing 1 to FACCERR. When FACCERR is set, it is not possible to launch a command or start a command write sequence.</p> <p>0 No access error detected. 1 Access error has occurred.</p>
3	Reserved, must be cleared.
2 FBLANK	<p><b>Flag Indicating the Erase Verify Operation Status</b> — When the FCCF flag is set after completion of an erase verify command, the FBLANK flag indicates the result of the erase verify operation. The FBLANK flag is cleared by the flash module when FCBEF is cleared as part of a new valid command write sequence. Writing to the FBLANK flag has no effect on FBLANK.</p> <p>0 Flash block verified as not erased. 1 Flash block verified as erased.</p>
1–0	Reserved, must be cleared.

#### 4.4.2.6 Flash Command Register (FCMD)

The FCMD register is the flash command register.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	FCMD							
Reset	0	0	0	0	0	0	0	0

Figure 4-7. Flash Command Register (FCMD)

All FCMD bits read 0 and are writable during a command write sequence, while bit 7 reads 0 and is not writable.

**Table 4-16. FCMD Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6–0 FCMD	<p><b>Flash Command</b> — Valid flash commands are shown below. Writing any command other than those listed sets the FACCERR flag in the FSTAT register.</p> <p>0x05 Erase verify            0x20 Program            0x25 Burst program            0x40 Sector erase            0x41 Mass erase</p>

### 4.4.3 Flash Command Operations

Flash command operations execute program, erase, and erase verify algorithms described in this section. The program and erase algorithms are controlled by the flash memory controller whose time base, FCLK, is derived from the bus clock via a programmable divider.

The next sections describe:

1. How to write the FCDIV register to set FCLK
2. Command write sequences to program, erase, and erase verify operations on the flash memory
3. Valid flash commands
4. Effects resulting from illegal flash command write sequences or aborting flash operations

#### 4.4.3.1 Writing the FCDIV Register

Prior to issuing any flash command after a reset, you must write the FCDIV register to divide the bus clock within the 150 kHz to 200 kHz range.

If defined:

- FCLK as the clock of the flash timing control block
- INT(x) as taking the integer part of x (for example, INT(4.323) = 4)

then the FCDIV[PRDIV8, FDIV] bits must be set as described in [Figure 4-8](#).

For example, if the bus clock frequency is 25 MHz, FCDIV[FDIV] must be set to 0x0F (001111) and the FCDIV[PRDIV8] bit set to 1. The resulting FCLK frequency is then 195 kHz. In this case, the flash program and erase algorithm timings are increased over the optimum target by:

$$(200 - 195) \div 200 = 3\%$$

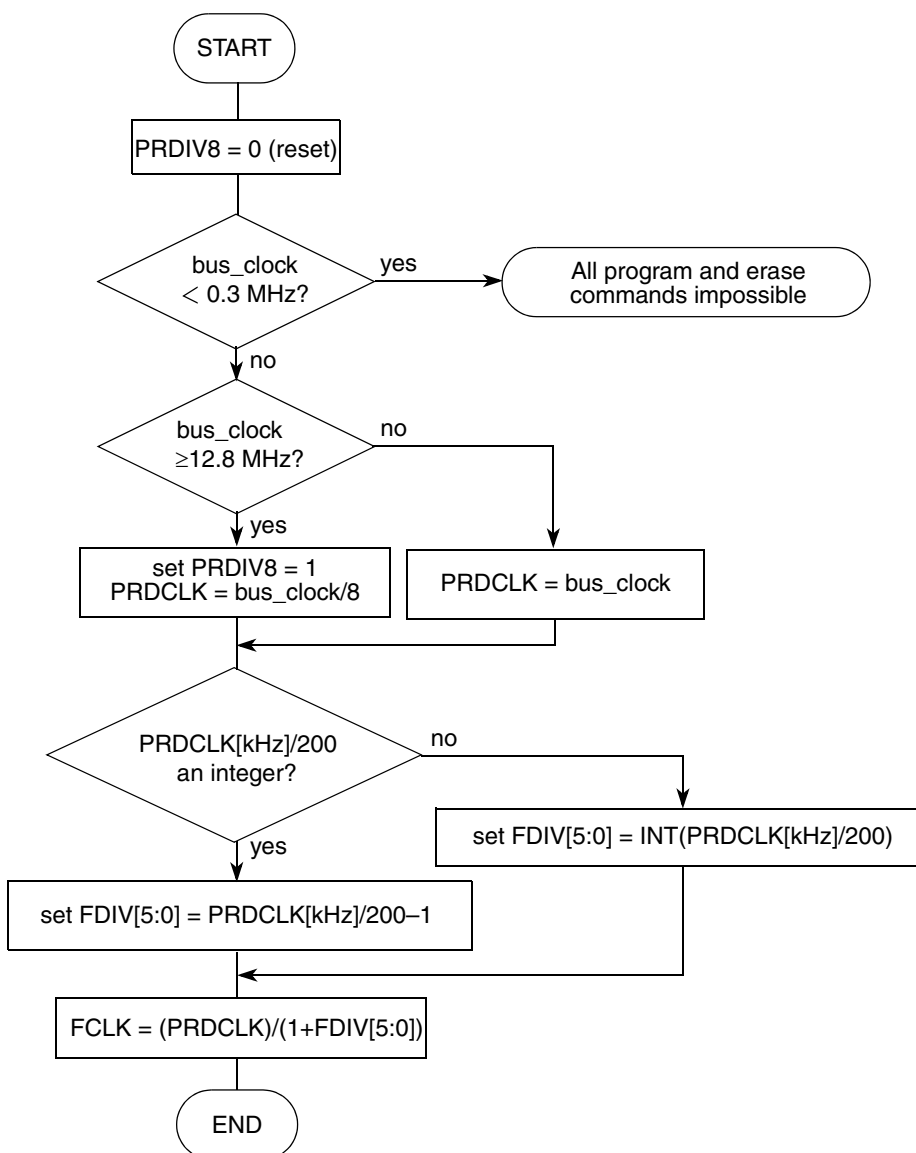
**Eqn. 4-1**



**CAUTION**

Program and erase command execution time increases proportionally with the period of FCLK. Programming or erasing the flash memory with  $FCLK < 150$  kHz must be avoided. Setting FCDIV to a value such that  $FCLK < 150$  kHz can destroy the flash memory due to overstress. Setting FCDIV to a value such that  $FCLK > 200$  kHz can result in incomplete programming or erasure of the flash memory cells.

If the FCDIV register is written, the FDIVLD bit sets automatically. If the FDIVLD bit is 0, the FCDIV register has not been written since the last reset. If the FCDIV register has not been written to, the flash command loaded during a command write sequence does not execute and FSTAT[FACCERR] sets.



**Figure 4-8. Determination Procedure for PRDIV8 and FDIV Bits**

### 4.4.3.2 Command Write Sequence

The flash command controller supervises the command write sequence to execute program, erase, and erase verify algorithms.

Before starting a command write sequence, the FACCERR and FPVIOL flags in the FSTAT register must be clear and the FCBEF flag must be set (see [Section 4.4.2.5](#)).

A command write sequence consists of three steps that must be strictly adhered to with writes to the flash module not permitted between the 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 a valid address in the flash array memory.
2. Write a valid command to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the command.

Once a command is launched, the completion of the command operation is indicated by setting the FCCF flag in the FSTAT register. The FCCF flag sets after completion of all active and buffered burst program commands.

## 4.4.4 Flash Commands

[Table 4-17](#) summarizes the valid flash commands along with the effects of the commands on the flash block.

**Table 4-17. Flash Command Description**

FCMD	NVM Command	Function on Flash Memory
0x05	Erase Verify	Verify all memory bytes in the flash array memory are erased. If the flash array memory is erased, the FBLANK flag in the FSTAT register sets after command completion.
0x20	Program	Program an address in the flash array.
0x25	Burst Program	Program an address in the flash array with the internal address incrementing after the program operation.
0x40	Sector Erase	Erase all memory bytes in a sector of the flash array.
0x41	Mass Erase	Erase all memory bytes in the flash array. A mass erase of the full flash array is only possible when no protection is enabled prior to launching the command.

### CAUTION

A flash block address must be in the erased state before being programmed. Cumulative programming of bits within a flash block address is not allowed except for status field updates required in EEPROM emulation applications.

#### 4.4.4.1 Erase Verify Command

The erase verify operation verifies the entire flash array memory is erased.

An example flow to execute the erase verify operation is shown in [Figure 4-9](#). The erase verify command write sequence is as follows:

1. Write to an aligned flash block address to start the command write sequence for the erase verify command. The address and data written is ignored.
2. Write the erase verify command, 0x05, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the erase verify command.

After launching the erase verify command, the FCCF flag in the FSTAT register sets after the operation has completed. The number of bus cycles required to execute the erase verify operation is equal to the number of addresses in the flash array memory plus several bus cycles as measured from the time the FCBEF flag is cleared until the FCCF flag is set. After erase verify operation completion, the FBLANK flag in the FSTAT register sets if all addresses in the flash array memory are verified to be erased. If any address in the flash array memory is not erased, the erase verify operation terminates and the FBLANK flag in the FSTAT register remains clear.

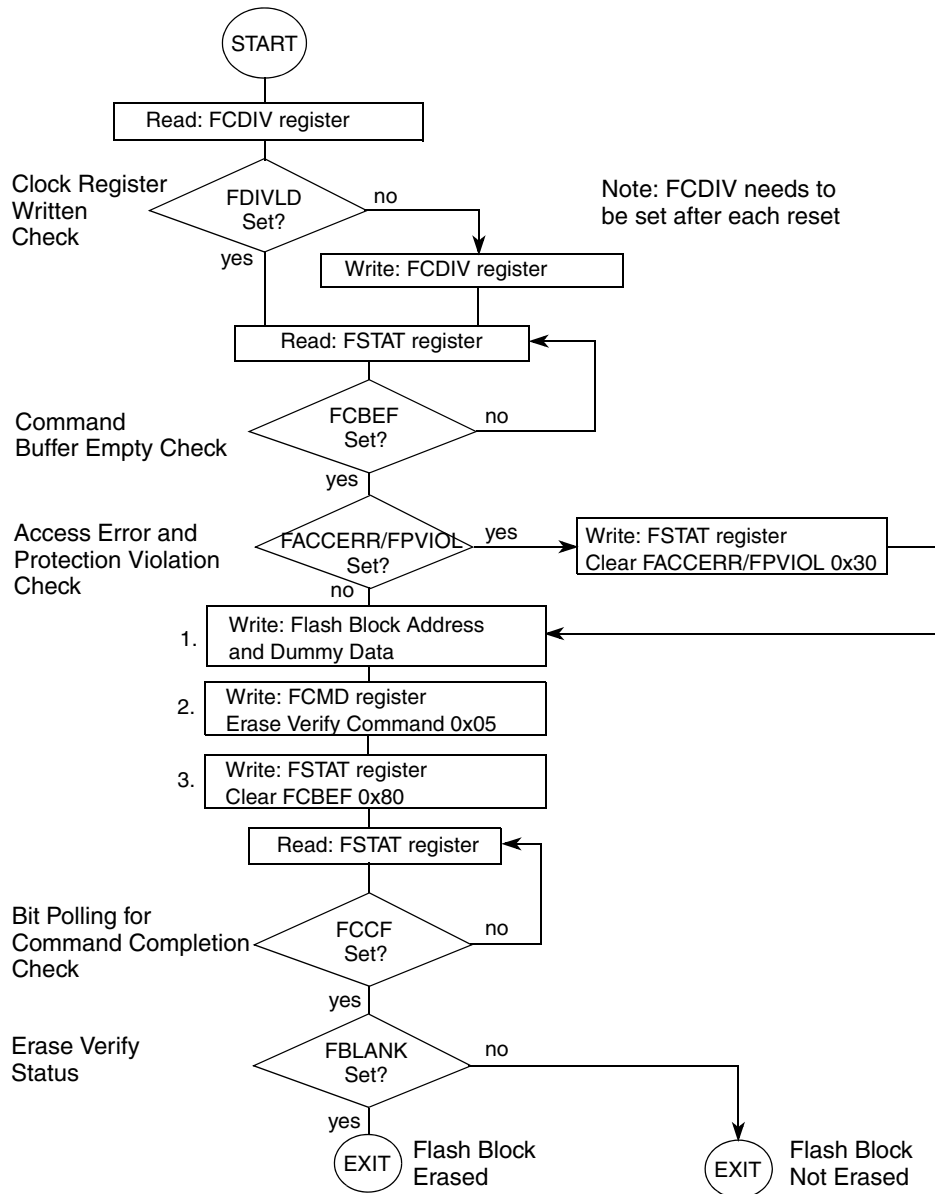


Figure 4-9. Example Erase Verify Command Flow

#### 4.4.4.2 Program Command

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 4-10](#). The program command write sequence is as follows:

1. Write to an aligned flash block address to start the command write sequence for the program command. The data written is programmed to the address written.
2. Write the program command, 0x20, to the FCMD register.

- Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the program command.

If an address to be programmed is in a protected area of the flash block, the FPVIOL flag in the FSTAT register sets and the program command does not launch. Once the program command has successfully launched, the FCCF flag in the FSTAT register sets after the program operation completion.

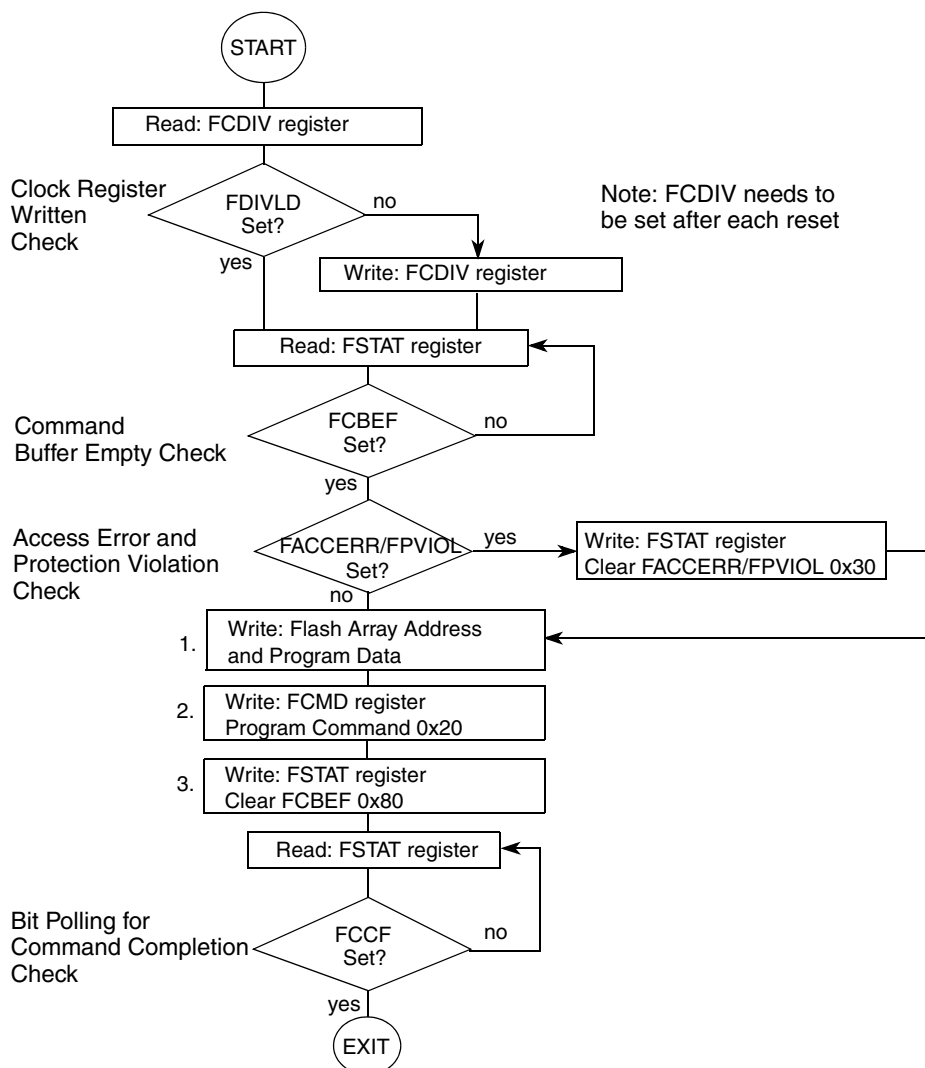


Figure 4-10. Example Program Command Flow

#### 4.4.4.3 Burst Program Command

The burst program operation programs previously erased data in the flash memory using an embedded algorithm.

While burst programming, two internal data registers operate as a buffer and a register (2-stage FIFO) so that a second burst programming command along with the necessary data can be stored to the buffers while the first burst programming command is still in progress. This pipelined operation allows a time optimization when programming more than one consecutive address on a specific row in the flash array. The high voltage generation can be kept active in between two programming commands.

An example flow to execute the burst program operation is shown in [Figure 4-11](#). The burst program command write sequence is as follows:

1. Write to an aligned flash block address to start the command write sequence for the burst program command. The data written is programmed to the address written.
2. Write the program burst command, 0x25, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the program burst command.
4. After the FCBEF flag in the FSTAT register returns to 1, repeat steps 1 through 3. The address written is ignored but is incremented internally.

The burst program procedure can be used to program the entire flash memory even while crossing row boundaries within the flash array. If data to be burst programmed falls within a protected area of the flash array, the FPVIOL flag in the FSTAT register sets and the burst program command does not launch. After the burst program command has successfully launched, the FCCF flag in the FSTAT register sets after the burst program operation has completed unless a new burst program command write sequence has been buffered. By executing a new burst program command write sequence on sequential addresses after the FCBEF flag in the FSTAT register has been set, greater than 50% faster programming time for the entire flash array can be effectively achieved when compared to using the basic program command.

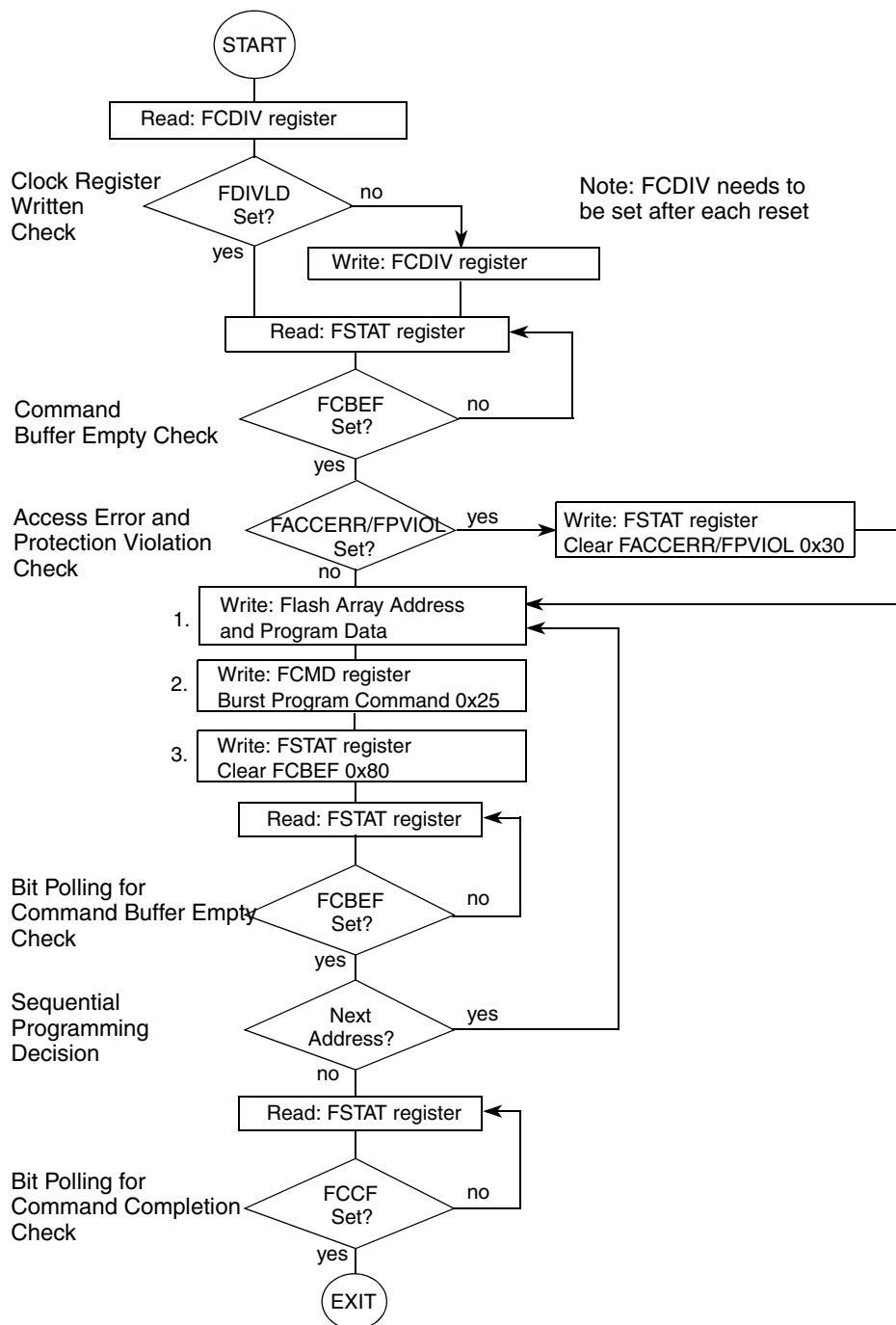


Figure 4-11. Example Burst Program Command Flow

#### 4.4.4.4 Sector Erase Command

The sector erase operation erases all addresses in 1 KB sector of flash memory using an embedded algorithm.

An example flow to execute the sector erase operation is shown in [Figure 4-12](#). The sector erase command write sequence is as follows:

1. Write to an aligned flash block address to start the command write sequence for the sector erase command. The flash address written determines the sector to be erased while the data written is ignored.
2. Write the sector erase command, 0x40, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the sector erase command.

If a flash sector to be erased is in a protected area of the flash block, the FPVIOL flag in the FSTAT register sets and the sector erase command does not launch. Once the sector erase command has successfully launched, the FCCF flag in the FSTAT register sets after the sector erase operation completion.



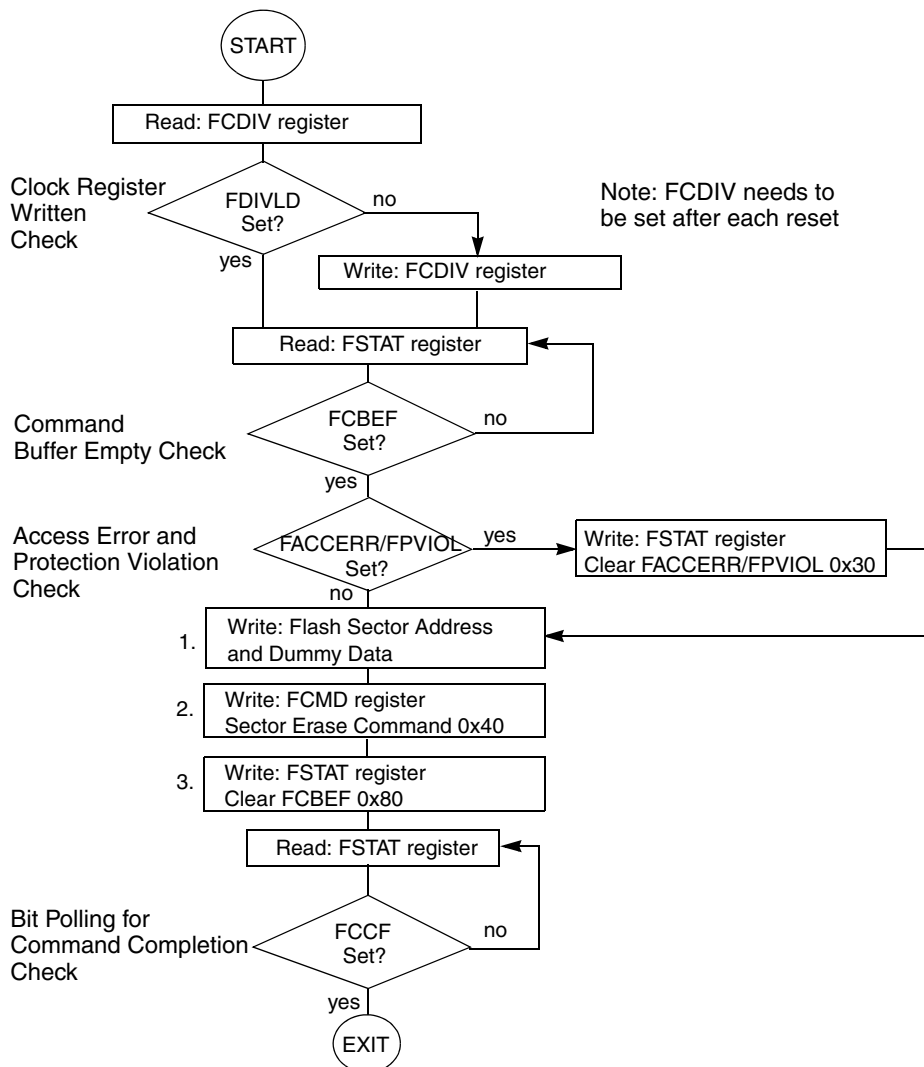


Figure 4-12. Example Sector Erase Command Flow

#### 4.4.4.5 Mass Erase Command

The mass erase operation erases the entire flash array memory using an embedded algorithm.

An example flow to execute the mass erase operation is shown in [Figure 4-13](#). The mass erase command write sequence is as follows:

1. Write to an aligned flash block address to start the command write sequence for the mass erase command. The address and data written are ignored.
2. Write the mass erase command, 0x41, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing 1 to FCBEF to launch the mass erase command.

If the flash array memory to be mass erased contains any protected area, the FPVIOL flag in the FSTAT register sets and the mass erase command does not launch. Once the mass erase command has successfully launched, the FCCF flag in the FSTAT register sets after the mass erase operation completion.

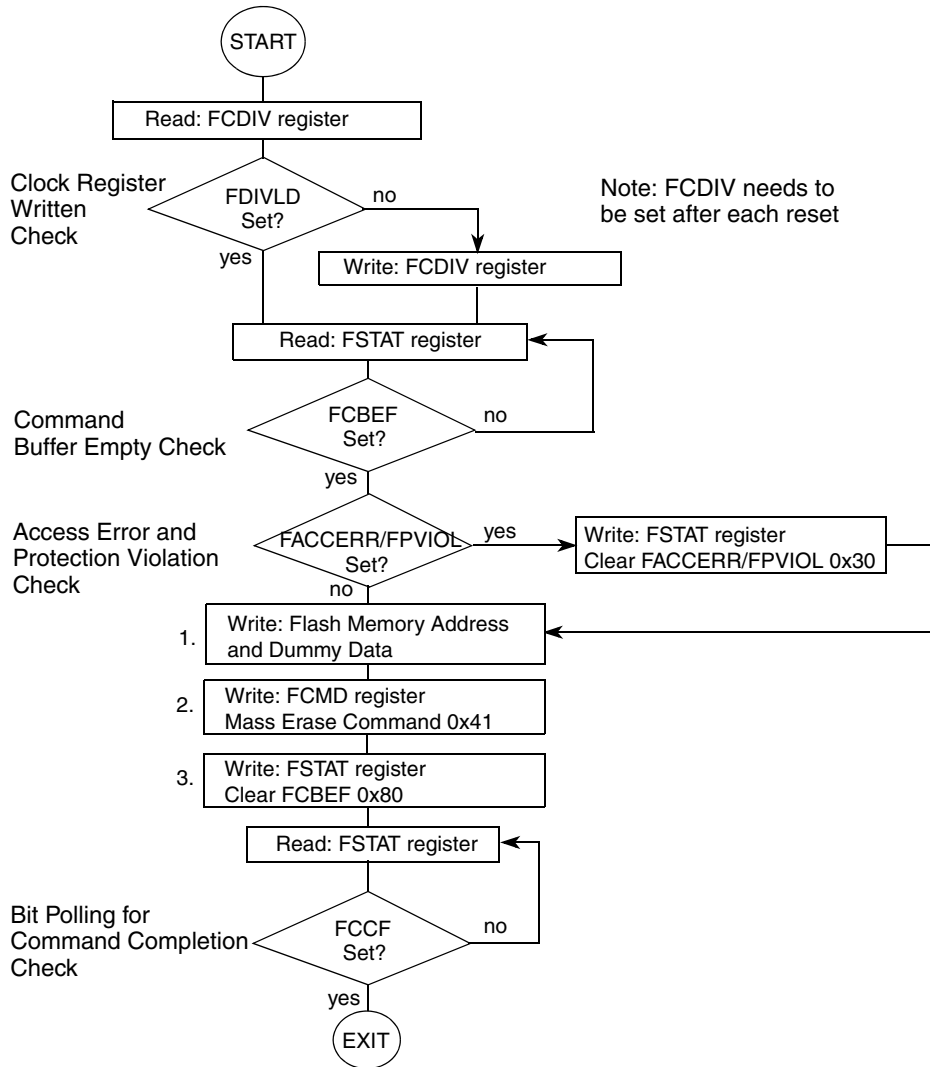


Figure 4-13. Example Mass Erase Command Flow

## 4.4.5 Illegal Flash Operations

### 4.4.5.1 Flash Access Violations

The FACCERR flag sets during the command write sequence if any of the following illegal steps are performed, causing the command write sequence to immediately abort:

1. Writing to a flash address before initializing the FCDIV register.
2. Writing a byte, word, or misaligned longword to a valid flash address.
3. Writing to any flash register other than FCMD after writing to a flash address.
4. Writing to a second flash address in the same command write sequence.
5. Writing an invalid command to the FCMD register unless the address written was in a protected area of the flash array.
6. Writing a command other than burst program while FCBEF is set and FCCF is cleared.
7. When security is enabled, writing a command other than erase verify or mass erase to the FCMD register when the write originates from a non-secure memory location or from the background debug mode.
8. Writing to a flash address after writing to the FCMD register.
9. Writing to any flash register other than FSTAT (to clear FCBEF) after writing to the FCMD register.
10. Writing 0 to the FCBEF flag in the FSTAT register to abort a command write sequence.

The FACCERR flag also sets if the MCU enters stop mode while any command is active ( $FCCF = 0$ ). The operation is aborted immediately and, if burst programming, any pending burst program command is purged (see [Section 4.4.6.2, “Stop Mode”](#)).

The FACCERR flag does not set if any flash register is read during a valid command write sequence.

If the flash memory is read during execution of an algorithm ( $FCCF = 0$ ), the read operation returns invalid data and the FACCERR flag does not set.

If the FACCERR flag is set in the FSTAT register, you must clear the FACCERR flag before starting another command write sequence (see [Section 4.4.2.5, “Flash Status Register \(FSTAT\)”](#)).

### 4.4.5.2 Flash Protection Violations

The FPVIOL flag sets after the command is written to the FCMD register during a command write sequence if any of the following illegal operations are attempted, causing the command write sequence to immediately abort:

1. Writing the program command if the address written in the command write sequence was in a protected area of the flash array.
2. Writing the sector erase command if the address written in the command write sequence was in a protected area of the flash array.
3. Writing the mass erase command while any flash protection is enabled.

4. Writing an invalid command if the address written in the command write sequence was in a protected area of the flash array.

If the FPVIOL flag is set in the FSTAT register, you must clear the FPVIOL flag before starting another command write sequence (see [Section 4.4.2.5, “Flash Status Register \(FSTAT\)”](#)).

## 4.4.6 Operating Modes

### 4.4.6.1 WAIT Mode

If a command is active (FCCF = 0) when the MCU enters wait mode, the active command and any buffered command are completed.

### 4.4.6.2 Stop Mode

If a command is active (FCCF = 0) when the MCU enters stop mode, the operation is halted. If the operation is program or erase, the flash array data being programmed or erased may be corrupted and the FCCF and FACCERR flags are set. If active, the high voltage circuitry to the flash array is immediately switched off when entering stop mode. Upon exit from stop mode, the FCBEF flag sets and any buffered command does not launch. The FACCERR flag must be cleared before starting a command write sequence (see [Section 4.4.3.2, “Command Write Sequence”](#)).

#### NOTE

- As active commands are immediately aborted when the MCU enters stop mode, it is strongly recommended that you do not use the STOP instruction during program or erase operations.
- Active commands continue when the MCU enters wait mode. Use of the STOP instruction when SOPT1[WAITE] = 1 is acceptable.

### 4.4.6.3 Background Debug Mode

In background debug mode, the FPROT register is writable without restrictions. If the MCU is unsecured, then all flash commands listed in [Table 4-17](#) can be executed. If the MCU is secured, only the mass erase and erase verify commands can be executed.

## 4.4.7 Security

The flash module provides the necessary security information to the MCU. During each reset sequence, the flash module determines the security state of the MCU as defined in [Section 4.2.3, “Flash Module Reserved Memory Locations”](#).

The contents of the flash security byte in the flash configuration field (see [Section 4.4.2.3](#)) must be changed directly by programming the flash security byte location when the MCU is unsecured and the sector containing the flash security byte is unprotected. If the flash security byte is left in a secured state, any reset causes the MCU to initialize into a secure operating mode.

### 4.4.7.1 Unsecuring the MCU using Backdoor Key Access

The MCU may be unsecured by using the backdoor key access feature which requires knowledge of the contents of the backdoor keys (see [Section 4.2.3](#)). If the KEYEN[1:0] bits are in the enabled state (see [Section 4.4.2.2](#)) and the KEYACC bit is set, a write to a backdoor key address in the flash memory triggers a comparison between the written data and the backdoor key data stored in the flash memory. If all backdoor keys are written to the correct addresses in the correct order and the data matches the backdoor keys stored in the flash memory, the MCU is then unsecured. The data must be written to the backdoor keys sequentially. Values 0x0000\_0000 and 0xFFFF\_FFFF are not permitted as backdoor keys. While the KEYACC bit is set, the flash memory reads return valid data.

The user code stored in the flash memory must have a method of receiving the backdoor keys from an external stimulus. This external stimulus would typically be through one of the on-chip serial ports.

If the KEYEN[1:0] bits are in the enabled state (see [Section 4.4.2.2](#)), the MCU can be unsecured by the backdoor key access sequence described below:

1. Set the KEYACC bit in the flash configuration register (FCNFG).
2. Sequentially write the correct two longwords to the flash addresses containing the backdoor keys.
3. Clear the KEYACC bit. Depending on the user code used to write the backdoor keys, a wait cycle (NOP) may be required before clearing the KEYACC bit.
4. If all data written match the backdoor keys, the MCU is unsecured and the SEC[1:0] bits in the NVOPT register are forced to an unsecured state.

The backdoor key access sequence is monitored by an internal security state machine. An illegal operation during the backdoor key access sequence causes the security state machine to lock, leaving the MCU in the secured state. A reset of the MCU causes the security state machine to exit the lock state and allow a new backdoor key access sequence to be attempted. The following operations during the backdoor key access sequence locks the security state machine:

1. If any of the keys written does not match the backdoor keys programmed in the flash array.
2. If the keys are written in the wrong sequence.
3. If any of the keys written are all 0's or all 1's.
4. If the KEYACC bit does not remain set while the keys are written.
5. If any of the keys are written on successive MCU clock cycles.
6. Executing a STOP instruction before all keys have been written.

After the backdoor keys have been correctly matched, the MCU is unsecured. Once the MCU is unsecured, the flash security byte can be programmed to the unsecure state, if desired.

In the unsecure state, you have full control of the contents of the backdoor keys by programming the associated addresses in the flash configuration field (see [Section 4.2.3](#)).

The security as defined in the flash security byte is not changed by using the backdoor key access sequence to unsecure. The stored backdoor keys are unaffected by the backdoor key access sequence. After the next reset of the MCU, the security state of the flash module is determined by the flash security byte. The backdoor key access sequence has no effect on the program and erase protections defined in the flash protection register (FPROT).

It is not possible to unsecure the MCU by using the backdoor key access sequence in background debug mode (BDM).

## 4.4.8 Resets

### 4.4.8.1 Flash Reset Sequence

On each reset, the flash module executes a reset sequence to hold CPU activity while reading the following resources from the flash block:

- MCU control parameters (see [Section 4.2.3](#))
- Flash protection byte (see [Section 4.2.3](#) and [Section 4.4.2.4](#))
- Flash nonvolatile byte (see [Section 4.2.3](#))
- Flash security byte (see [Section 4.2.3](#) and [Section 4.4.2.2](#))

### 4.4.8.2 Reset While Flash Command Active

If a reset occurs when any flash command is in progress, the command is immediately halted. The state of the flash array address being programmed or the sector/block being erased is not guaranteed.

### 4.4.8.3 Program and Erase Times

Before any program or erase command can be accepted, the flash clock divider (CSR3[FCDIV]) must be written to set the internal clock for the flash module to a frequency ( $f_{FCLK}$ ) between 150 kHz and 200 kHz. One period of the resulting clock ( $1/f_{FCLK}$ ) is used by the command processor to time program and erase pulses. An integer number of these timing pulses are used by the command processor to complete a program or erase command.

Program and erase times are given in the MCF51AG128 *Data Sheet* (document MCF51AG128).

## 4.5 Security

This device includes circuitry to prevent unauthorized access to the contents of flash and RAM memory. When security is engaged, BDM access is restricted to the upper byte of the ColdFire CSR, XCSR, and CSR2 registers. RAM, flash memory, peripheral registers, and most of the CPU register set are not available via BDM. Programs executing from internal memory have normal access to all MCU memory locations and resources.

Security is engaged or disengaged based on the state of two nonvolatile register bits (SEC01, SEC00) in the FOPT register. During reset, the contents of the nonvolatile location, NVOPT, are copied from flash into the working FOPT register in high-page register space. Enable security by programming the NVOPT location which can be done at the same time the flash memory is programmed. The 1:0 state disengages security and the other three combinations engage security. Note that security is implemented differently than on the pin-compatible MC9S08AC128 family of devices. This is a result of differences inherent in the S08 and ColdFire MCU architectures.

Upon exiting reset, the XCSR[25] bit in the ColdFire CPU is set if the device is secured, cleared otherwise.

You can choose to allow or disallow a security unlocking mechanism through an 8-byte backdoor security key. The security key can be written by the CPU executing from the internal memory. It cannot be entered without the cooperation of a secure user program. The procedure for this is detailed in [Section 4.4.7.1, “Unsecuring the MCU using Backdoor Key Access.”](#)

Development tools unsecure devices via an alternate BDM-based methodology shown in [Figure 4-14](#). Because both  $\overline{\text{RESET}}$  and BKGD pins can be reprogrammed via software, a power-on-reset is required to be absolutely certain of obtaining control of the device via BDM, which is a required prerequisite for clearing security. Other methods (outlined in red in [Figure 4-14](#)) can also be used, but may not work under all circumstances.

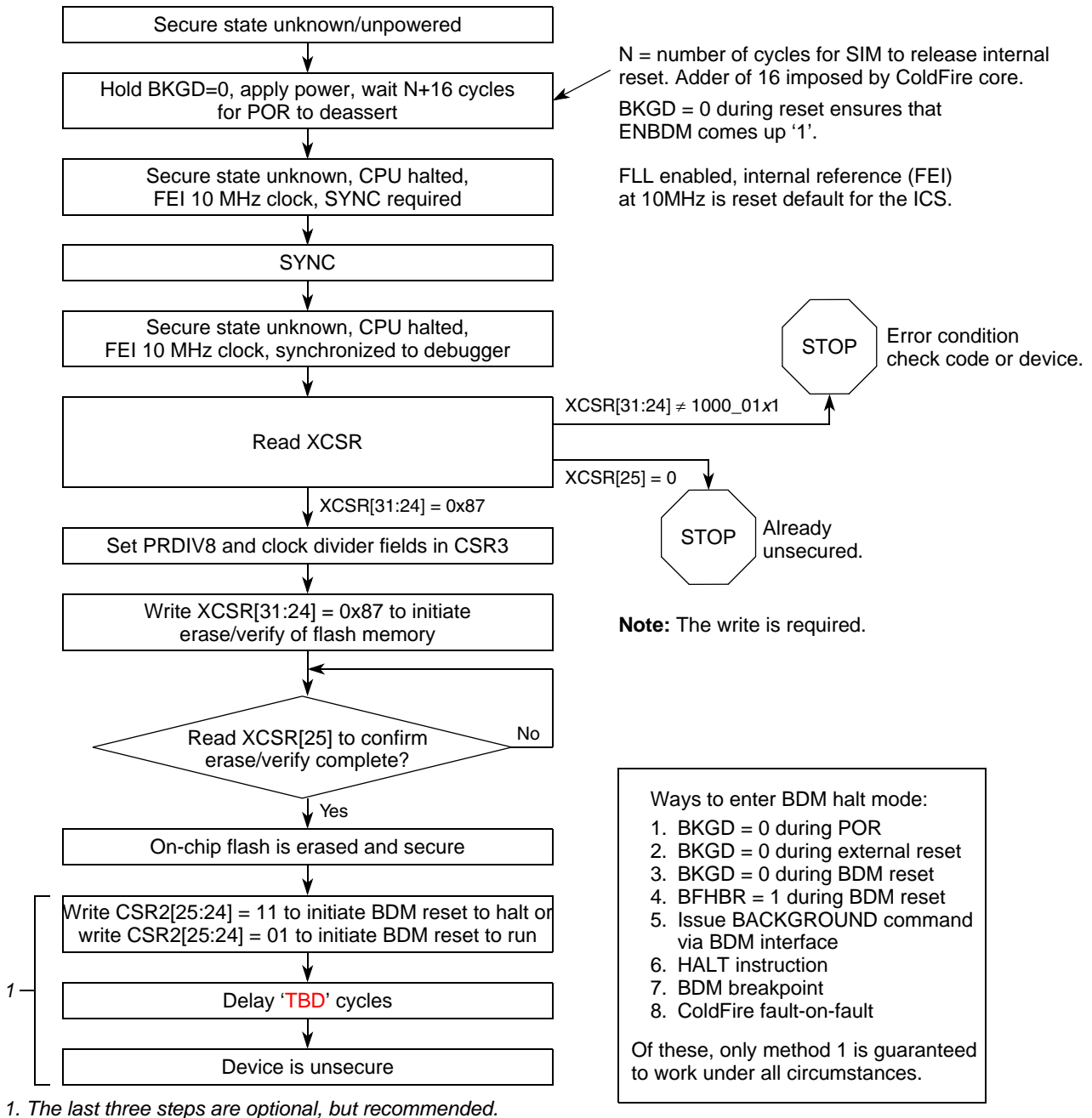


Figure 4-14. Procedure for Clearing Security via the BDM Port



# Chapter 5

## Resets, Interrupts, and General System Control

### 5.1 Introduction

This section discusses basic reset and interrupt mechanisms and the various sources of reset and interrupt on MCF51AG128 series of microcontrollers. Some interrupt sources from peripheral modules are discussed in more detail within other sections of this document. This section gathers basic information about all reset and interrupt sources in one place for easy reference.

### 5.2 Features

Reset and interrupt features include:

- Multiple sources of reset for flexible system configuration and reliable operation
- System reset status (SRS) register to indicate source of the most recent reset
- Separate interrupt vector for most of the modules (reduces polling overhead) (see [Table 5-1](#))

### 5.3 Microcontroller Reset

Resetting the microcontroller provides a way to start processing from a known set of initial conditions. When the ColdFire processor exits reset, it fetches initial 32-bit values for the supervisor stack pointer and program counter from locations 0x(00)00\_0000 and 0x(00)00\_0004, respectively. On-chip peripheral modules are disabled and I/O pins are initially configured as general-purpose high-impedance inputs with pullup devices disabled.

The MCF51AG128 series microcontrollers have the following sources for reset:

- Power-on reset (POR)
- External pin reset (PIN)
- Illegal opcode detect (ILOP)
- Illegal address detect (ILAD)
- Low-voltage detect (LVD)
- Watchdog timer reset (WDOG)
- Background debug forced reset (BDFR)
- Loss of clock reset (LOC)

Each of these sources, with the exception of the BDFR, has an associated bit in the system reset status register (SRS). SRS is cleared after BDFR.

### 5.3.1 Illegal Opcode Detect (ILOP)

By default, the V1 ColdFire core generates a MCU reset when attempting to execute an illegal instruction (except for the ILLEGAL opcode), illegal line-A instruction, illegal line-F instruction, or a supervisor instruction while in user mode (privilege violation). You may set CPUCCR[IRD] to generate the appropriate exception instead of forcing a reset.

#### NOTE

- The attempted execution of the STOP instruction with SOPT[STOPE, WAITE] cleared is treated as an illegal instruction.
- The attempted execution of the HALT instruction with XCSR[ENBDM] cleared is treated as an illegal instruction.

### 5.3.2 Illegal Address Detect (ILAD)

By default, the V1 ColdFire core generates an MCU reset when detecting an address error, bus error termination, RTE format error, or fault-on-fault condition. If CPUCCR[ARD] is set, the processor generates the appropriate exception instead of forcing a reset, or simply halts the processor in response to the fault-on-fault condition.

### 5.3.3 Watchdog Timer (WDOG)

See [Chapter 17, “Watchdog Timer,”](#) for information on the watchdog for MCF51AG128 devices.

### 5.3.4 Loss of Clock Reset (LOC)

The MCF51AG128 series microcontrollers can monitor the external reference clock (ICSERCLK) from ICS when SOPT2[CME] bit is set and generates a MCU reset if this clock is lost. It is useful if the crystal fails during MCU operation in FEE mode. The clock monitor could bring the MCU to FEI mode when SOPT2[CME] bit is set.

## 5.4 Interrupts and Exceptions

The ColdFire interrupt architecture uses a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing seven levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. For more information on exception processing, see [Chapter 8, “Interrupt Controller \(CF1\\_INTC\).”](#)

### 5.4.1 External Interrupt Request (IRQ) Pin

External interrupts are managed by the IRQ status and control register, IRQSC. When the IRQ function is enabled, synchronous logic monitors the pin for edge-only or edge-and-level events. When the microcontroller is in the stop mode and system clocks are shut down, a separate asynchronous path is used, therefore the IRQ pin (if enabled) can wake the microcontroller.

### 5.4.1.1 Pin Configuration Options

The IRQ pin enable (IRQPE) control bit in IRQSC must be set for the IRQ pin to act as the interrupt request (IRQ) input. As an IRQ input, you can choose the polarity of edges or levels detected (IRQEDG), whether the pin detects edges-only or edges and levels (IRQMOD), and whether an event causes an interrupt or only sets the IRQF flag that can be polled by software (IRQIE).

The IRQ pin, when enabled, defaults to use an internal pull device (IRQPDD = 0), configured as a pullup or pulldown depending on the polarity chosen. To use an external pullup or pulldown, the IRQPDD can be set to turn off the internal device.

#### NOTE

- This pin does not contain a clamp diode to  $V_{DD}$  and must not be driven above  $V_{DD}$ .
- The voltage measured on the internally pulled up  $\overline{IRQ}$  pin is not pulled to  $V_{DD}$ . The internal gates connected to this pin are pulled to  $V_{DD}$ . The  $\overline{IRQ}$  pullup must not be used to pull up components external to the microcontroller.

### 5.4.2 Interrupt Vectors, Sources, and Local Masks

Table 5-1 shows address assignments for reset and interrupt vectors. The vector names shown in this table are the labels used in the Freescale Semiconductor-provided equate file for the MCF51AG128 series microcontrollers. The table is sorted by priority of the sources, with higher-priority sources at the top of the table. The force\_lvl entries do not follow the address and vector number order of the surrounding vectors.

Table 5-1. V1 ColdFire Exception Vector Table

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment
0	0x000		—	Initial supervisor stack pointer
1	0x004		—	Initial program counter
2–63	0x008–0x0FC		—	Reserved for internal CPU exceptions
64	0x100	7,mid	Next	IRQ_pin
65	0x104	7,3	Next	low_voltage
66	0x108	7,2	Next	Reserved
67	0x10C	7,1	Next	Reserved
68	0x110	6,5	Next	DMA_ch0
69	0x114	6,4	Next	DMA_ch1
70	0x118	6,3	Next	DMA_ch2
71	0x11C	6,2	Next	DMA_ch3
72	0x120	6,1	Next	iEvent_ch0

Table 5-1. V1 ColdFire Exception Vector Table (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment
73	0x124	5,7	Next	FTM1--(fault+ovfl)
74	0x128	5,6	Next	FTM1_ch0
75	0x12C	5,5	Next	FTM1_ch1
76	0x130	5,4	Next	FTM1_ch2
77	0x134	5,3	Next	FTM1_ch3
78	0x138	5,2	Next	FTM1_ch4
79	0x13C	5,1	Next	FTM1_ch5
80	0x140	4,7	Next	FTM2--(fault+ovfl)
81	0x144	4,6	Next	FTM2_ch0
82	0x148	4,5	Next	FTM2_ch1
83	0x14C	4,4	Next	FTM2_ch2
84	0x150	4,3	Next	FTM2_ch3
85	0x154	4,2	Next	FTM2_ch4
86	0x158	4,1	Next	FTM2_ch5
87	0x15C	3,7	Next	TPM3_ovfl
88	0x160	3,6	Next	TPM3_ch0
89	0x164	3,5	Next	TPM3_ch1
90	0x168	3,4	Next	ADC
91	0x16C	3,3	Next	HSCMP1
92	0x170	3,2	Next	HSCMP2
93	0x174	3,1	Next	iEvent_ch1
94	0x178	2,7	Next	SPI1
95	0x17C	2,6	Next	SPI2
96	0x180	2,5	Next	SCI1_err
97	0x184	2,4	Next	SCI1_rx
98	0x188	2,3	Next	SCI1_tx
99	0x18C	2,2	Next	IIC
100	0x190	2,1	Next	iEvent_ch2
101	0x194	1,7	Next	SCI2_err
102	0x198	1,6	Next	SCI2_rx
103	0x19C	7,0	Next	Level 7 Software Interrupt
104	0x1A0	6,0	Next	Level 6 Software Interrupt
105	0x1A4	5,0	Next	Level 5 Software Interrupt

Table 5-1. V1 ColdFire Exception Vector Table (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment
106	0x1A8	4,0	Next	Level 4 Software Interrupt
107	0x1AC	3,0	Next	Level 3 Software Interrupt
108	0x1B0	2,0	Next	Level 2 Software Interrupt
109	0x1B4	1,0	Next	Level 1 Software Interrupt
110	0x1B8	1,5	Next	SCI2_tx
111	0x1BC	1,4	Next	KBI[A:E]
112	0x1C0	1,mid	Next	KBI[F:J]
113	0x1C4	1,3	Next	RTC + WDOG
114	0x1C8	1,2	Next	iEvent_ch3
115–255	0x1CC–0x3FC	—	—	Reserved; unused for V1

Standard set of ColdFire exceptions are not shown in [Table 5-1](#), many of which apply to this device. These are listed below in [Table 5-2](#).

The CPU configuration register (CPUCR) within the supervisor programming model determines if specific ColdFire exception conditions are to generate a normal exception or a system reset. The default state of the CPUCR forces a system reset for any of the exception types listed in [Table 5-2](#).

Table 5-2. ColdFire Exception Vector Table<sup>1</sup>

Vector	Exception	Reset Disabled via CPUCR	Reported using SRS
64-98	I/O Interrupts	N/A	—
61	Unsupported instruction	N/A	—
47	Trap #15	N/A	—
46	Trap #14	N/A	—
45	Trap #13	N/A	—
44	Trap #12	N/A	—
43	Trap #11	N/A	—
42	Trap #10	N/A	—
41	Trap #9	N/A	—
40	Trap #8	N/A	—
39	Trap #7	N/A	—
38	Trap #6	N/A	—
37	Trap #5	N/A	—
36	Trap #4	N/A	—
35	Trap #3	N/A	—

Table 5-2. ColdFire Exception Vector Table<sup>1</sup> (continued)

Vector	Exception	Reset Disabled via CPUCR	Reported using SRS
34	Trap #2	N/A	—
33	Trap #1	N/A	—
32	Trap #0	N/A	—
24	Spurious IRQ	N/A	—
14	Format error	CPUCR[31]	ilad
12	Debug breakpoint IRQ	N/A	—
11	Illegal LineF	CPUCR[30]	ilop
10	Illegal LineA	CPUCR[30]	ilop
9	Trace	N/A	—
8	Privileged Violation	CPUCR[30]	ilop
4	Illegal instruction	CPUCR[30]	ilop <sup>2</sup>
3	Address error	CPUCR[31]	ilad
2	Access error	CPUCR[31]	ilad
n/a	Flt-on-Flt Halt	CPUCR[31]	ilad

<sup>1</sup> Exception vector numbers not appearing in this table are not applicable to the V1 core and are reserved.

<sup>2</sup> The execution of the ILLEGAL instruction (0x4AFC) always generates an illegal instruction exception, regardless of the state of CPUCR[30].

## 5.5 Low-Voltage Detect (LVD) System

The MCF51AG128 series of microcontrollers include a system to protect against low voltage conditions to protect memory contents and to control microcontroller system states during supply voltage variations. The system is comprised of a power-on reset (POR) circuit and a LVD circuit with a user-selectable trip voltage, high ( $V_{LVDH}$ ) or low ( $V_{LVDL}$ ). The LVD circuit is enabled when the SPMSC1[LVDE] bit is set and the trip voltage is selected by the SPMSC2[LVDV] bit. The LVD is disabled upon entering stop2 or stop3 modes unless the LVDSE bit is set. If LVDE and LVDSE are set when the STOP instruction is processed, the device enters the stop4 mode. The LVD can be left enabled in this mode.

### 5.5.1 Power-On Reset Operation

When power is initially applied to the microcontroller, or when the supply voltage drops below the power-on reset re-arm voltage level,  $V_{POR}$ , the POR circuit causes a reset condition. As the supply voltage rises, the LVD circuit holds the microcontroller in reset until the supply has risen above the LVD low threshold,  $V_{LVDL}$ . The POR and LVD bits in SRS are set following a POR.

## 5.5.2 LVD Reset Operation

The LVD can be configured to generate a reset upon detection of a low voltage condition by setting LVDRE. The low voltage detection threshold is determined by the LVDV bit. After LVD reset has occurred, the LVD system holds the microcontroller in reset until the supply voltage has risen above the low voltage detection threshold. The LVD bit in the SRS register is set following LVD reset or POR.

## 5.5.3 Low-Voltage Warning (LVW) Interrupt Operation

The LVD system has a low voltage warning flag (LVWF) to indicate that the supply voltage is approaching, but is above the LVD voltage. The LVW also has an interrupt associated with it, enabled by setting the SPMSC1[LVWIE] bit. If enabled, LVW interrupt request occurs when the LVWF is set. LVWF is cleared by writing 1 to the SPMSC1[LVWACK] bit. There are two user-selectable trip voltages for the LVW, one high ( $V_{LVWH}$ ) and one low ( $V_{LVWL}$ ). The trip voltage is selected by SPMSC2[LVWV] bit.

## 5.6 MCLK Output

The PTA6 pin is shared with the MCLK clock output. Setting the pin enable bit, MPE, causes the PTA6 pin to output a divided version of the internal MCU bus clock. The divide ratio is determined by the MCSEL bits. When MPE is set, the PTA6 pin is forced to operate as an output pin regardless of the state of the port data direction control bit for the pin. If the MCSEL bits are all 0s, the pin is driven low. The slew rate and drive strength for the pin are controlled by PTASRE6 and PTADS6 bits, respectively. The maximum clock output frequency is limited if the slew rate control is enabled. See MCF51AG128 *Data Sheet* (document MCF51AG128) for pin rise and fall times with slew rate enabled.

## 5.7 Peripheral Clock Gating

The MCF51AG128 series microcontroller includes a clock gating system to manage the bus clock sources to the individual peripherals. Using this system, you can enable or disable the bus clock to each of the peripherals at the clock source, eliminating unnecessary clocks to peripherals which are not in use; thereby reducing the overall run and wait mode currents.

Out of reset, all peripheral clocks are disabled except Flash, iEvent, and DMA controller. For lowest possible run or wait currents, user software must disable the clock source to any peripheral not in use. The actual clock is enabled or disabled immediately following the write to the clock gating control registers (SCGC1, SCGC2, and SCGC3). Any peripheral with a gated clock can not be used unless its clock is enabled. Writing to the registers of a peripheral with a disabled clock has no effect.

### NOTE

User software must disable the peripheral before disabling the clocks to the peripheral. When clocks are re-enabled to a peripheral, the peripheral registers need to be re-initialized by user software.

In stop modes, the bus clock is disabled for all gated peripherals, regardless of the settings in the SCGC1, SCGC2, and SCGC3 registers.

## 5.8 Reset, Interrupt, and System Control Registers and Control Bits

One 8-bit register in the direct page register space and eight 8-bit registers in the high-page register space are related to reset and interrupt systems.

Refer to [Table 4-3](#) and [Table 4-4](#) in [Chapter 4, “Memory,”](#) of this document for the absolute address assignments for all registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

Some control bits in the SOPT1 and SPMSC2 registers are related to modes of operation. Although brief descriptions of these bits are provided here, the related functions are discussed in more detail in [Chapter 3, “Modes of Operation.”](#)

### 5.8.1 Interrupt Pin Request Status and Control Register (IRQSC)

This direct page register includes status and control bits that configure the IRQ function, report status, and acknowledge IRQ events.

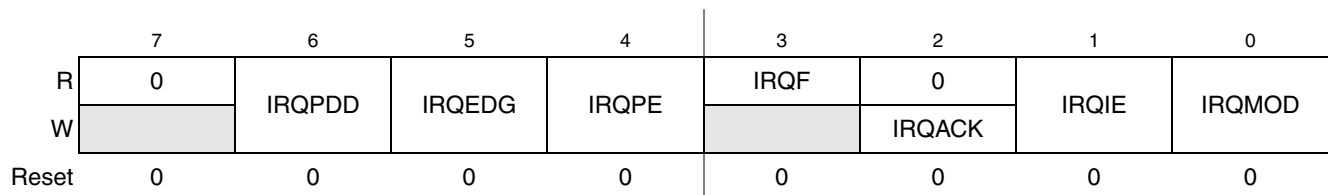


Figure 5-1. Interrupt Request Status and Control Register (IRQSC)

Table 5-3. IRQSC Register Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 IRQPDD	<b>Interrupt Request (IRQ) Pull Device Disable</b> — This read/write control bit disables the internal pullup/pulldown device when the IRQ pin is enabled (IRQPE = 1) allowing for an external device to be used. 0 IRQ pull device enabled if IRQPE is set. 1 IRQ pull device disabled if IRQPE is set.
5 IRQEDG	<b>Interrupt Request (IRQ) Edge Select</b> — This read/write control bit selects the polarity of edges or levels on the IRQ pin that cause IRQF to set. The IRQMOD control bit determines whether the IRQ pin is sensitive to edges and levels or only edges. When IRQEDG is set and the internal pull device is enabled, the pullup device is reconfigured as an optional pulldown device. 0 IRQ is falling edge or falling edge/low-level sensitive. 1 IRQ is rising edge or rising edge/high-level sensitive.
4 IRQPE	<b>IRQ Pin Enable</b> — This read/write control bit enables the IRQ pin function. When this bit is set, the IRQ pin can be used as an external interrupt request. 0 IRQ pin function is disabled. 1 IRQ pin function is enabled.
3 IRQF	<b>IRQ Flag</b> — This read-only status bit indicates when an interrupt request event has occurred. 0 No IRQ request. 1 IRQ event detected.



Table 5-3. IRQSC Register Field Descriptions (continued)

Field	Description
2 IRQACK	<b>IRQ Acknowledge</b> — This write-only bit acknowledges interrupt request events (write 1 to clear IRQF). Writing 0 has no meaning or effect. Reads always return 0. If edge-and-level detection is selected (IRQMOD = 1), IRQF cannot be cleared while the IRQ pin remains at its asserted level.
1 IRQIE	<b>IRQ Interrupt Enable</b> — This read/write control bit determines whether IRQ events generate an interrupt request. 0 Interrupt request when IRQF set is disabled (use polling). 1 Interrupt requested whenever IRQF is set.
0 IRQMOD	<b>IRQ Detection Mode</b> — This read/write control bit selects edge-only detection or edge-and-level detection. The IRQEDG control bit determines the polarity of edges and levels that are detected as interrupt request events. 0 IRQ event on falling edges or rising edges only. 1 IRQ event on falling edges and low levels or on rising edges and high levels.

## 5.8.2 System Reset Status Register (SRS)

This high page register includes read-only status flags to indicate the source of the most recent reset. When a debug host forces reset by setting CSR2[BDFR], none of the status bits in SRS are set. Writing any value to this register address has no effect.

	7	6	5	4	3	2	1	0
R	POR	PIN	LOC	ILOP	ILAD	WDOG	LVD	0
W	Writing any value to SRS has no effect.							
POR:	1	0	0	0	0	0	1	0
LVD:	U <sup>2</sup>	0	0	0	0	0	1	0
Any other reset:	0	Note <sup>1</sup>	Note <sup>1</sup>	Note <sup>1</sup>	Note <sup>1</sup>	Note <sup>1</sup>	0	0

<sup>1</sup> Any of these reset sources that are active at the time of reset entry causes the corresponding bit(s) to be set; bits corresponding to sources that are not active at the time of reset entry are cleared.

<sup>2</sup> When POR reset is followed by LVD reset, this bit remains set else cleared.

Figure 5-2. System Reset Status (SRS)

Table 5-4. SRS Register Field Descriptions

Field	Description
7 POR	<b>Power-On Reset</b> — Reset was caused by the power-on detection logic. Because the internal supply voltage was ramping up at the time, the low-voltage reset (LVD) status bit is also set to indicate that the reset occurred while the internal supply was below the LVD threshold. 0 Reset not caused by POR. 1 POR caused reset.
6 PIN	<b>External Reset Pin</b> — Reset was caused by an active-low level on the external reset pin. 0 Reset not caused by external reset pin. 1 Reset came from external reset pin.

Table 5-4. SRS Register Field Descriptions (continued)

Field	Description
5 LOC	<b>Loss of Clock Reset</b> — Reset was caused by the ICSECLK monitor logic. This status bit is set to indicate that the reset occurred while the ICSECLK is lost. 0 Reset not caused by LOC. 1 LOC caused reset.
4 ILOP	<b>Illegal Opcode</b> — Reset was caused by an attempt to execute an unimplemented or illegal opcode. This includes any illegal instruction (except the ILLEGAL (0x4AFC) opcode) or a privilege violation (execution of a privileged instruction in user mode). The STOP instruction is considered illegal if stop is disabled by ((SOPT[STOPE] = 0) && (SOPT[WAITE] = 0)). The HALT instruction is considered illegal if the BDM interface is disabled by XCSR[ENBDM] = 0. 0 Reset not caused by an illegal opcode. 1 Reset caused by an illegal opcode.
3 ILAD	<b>Illegal Address</b> — Reset was caused by the processor's attempted access of an illegal address in the memory map, an address error, an RTE format error, or the fault-on-fault condition. All the illegal address resets are enabled when CPUCR[ARD] is cleared. When CPUCR[ARD] is set, then the appropriate processor exception is generated instead of the reset, or if a fault-on-fault condition is reached, the processor simply halts. 0 Reset not caused by an illegal access. 1 Reset caused by an illegal access.
2 WDOG	<b>Watchdog timer reset</b> — Reset was caused by the WDOG timer timing out. This reset source can be blocked by WDOG_STAT_CTRL_[WDOG_EN], for more information see <a href="#">Chapter 17, "Watchdog Timer."</a> 0 Reset not caused by WDOG timeout. 1 Reset caused by WDOG timeout.
1 LVD	<b>Low Voltage Detect</b> — If the LVDRE bit is set and the supply drops below the LVD trip voltage, an LVD reset occurs. This bit is also set by POR. 0 Reset not caused by LVD trip or POR. 1 Reset caused by LVD trip or POR.

### 5.8.3 System Options (SOPT1) Register

This high-page register has one write-once bit and one write-anytime bit. For the write-once bits, only the first write after reset is respected. All bits in the register can be read at any time. Any subsequent attempt to write a write-once bit is ignored to avoid accidental changes to these sensitive settings. SOPT1 must be written during the reset initialization program to set the desired controls, even if the desired settings are the same as the reset settings.

	7	6	5	4	3	2	1	0
R	0	0	STOPE <sup>1</sup>	WAITE	0	0	0	0
W								
Reset:	0	0	0	1	0	0	0	0

<sup>1</sup> This bit can be written only one time after reset. Additional writes are ignored.

Figure 5-3. System Options (SOPT1) Register

Table 5-5. SOPT1 Register Field Descriptions

Field	Description
5 STOPE	<b>Stop Mode Enable</b> — This write-once bit enables stop mode. If stop and wait modes are disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset may be generated depending on CPUCCR[IRD].
4 WAITE	<b>WAIT Mode Enable</b> — This write-anytime bit enables wait mode. If stop and wait modes are disabled and a user program attempts to execute a STOP instruction, an illegal opcode reset may be generated depending on CPUCCR[IRD].
3–0	Reserved, must be cleared.

### 5.8.4 System MCLK Control Register (SMCLK)

This register controls the MCLK clock output.

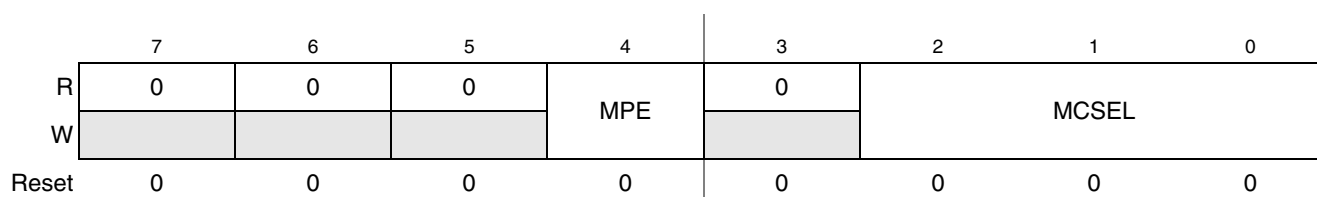


Figure 5-4. System Integration Module MCLK Control Register (SMCLK)

Table 5-6. SMCLK Register Field Descriptions

Field	Description
4 MPE	<b>MCLK Pin Enable</b> — This bit enables the MCLK function. 0 MCLK output disabled. 1 MCLK output enabled on PTA6 pin.
2–0 MCSEL	<b>MCLK Divide Select</b> — These bits select the divide ratio for the MCLK output according to the formula in <a href="#">Equation 5-1</a> when the MCSEL bits are not equal to all zeroes. In the case that the MCSEL bits are all zero and MPE is set, the pin is driven low. See <a href="#">Equation 5-1</a> .

$$\text{MCLK Frequency} = \text{Bus Clock frequency} \div (2 \times \text{MCSEL})$$

Eqn. 5-1

### 5.8.5 System Device Identification Register (SDIDH, SDIDL)

These high page read-only registers identify the ColdFire derivative. This allows the development software to recognize where specific memory blocks, registers, and control bits are located in a target microcontroller.

Additional configuration information about the ColdFire core and memory system is loaded into the 32-bit D0 (core) and D1 (memory) registers at reset. This information can be stored into memory by the system startup code for later use by configuration-sensitive application code.

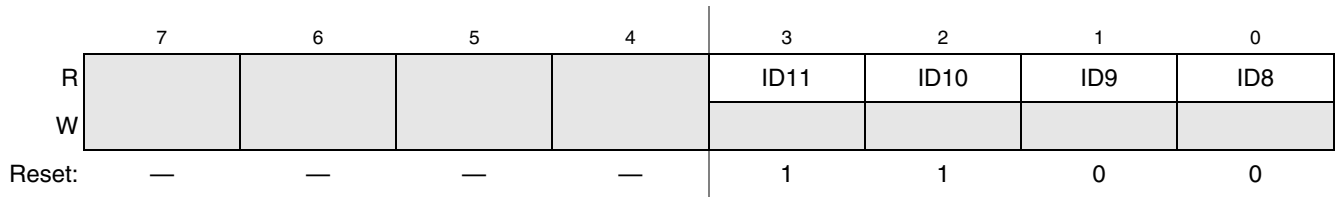


Figure 5-5. System Device Identification Register — High (SDIDH)

Table 5-7. SDIDH Register Field Descriptions

Field	Description
7–4 Reserved	Reserved. Reading these bits result in an indeterminate value; writes have no effect.
3–0 ID[11:8]	<b>Part Identification Number</b> — Each derivative in the ColdFire family has a unique identification number. The MCF51AG128 series microcontrollers are hard coded to the value 0xC05. See also ID bits in <a href="#">Table 5-8</a> .

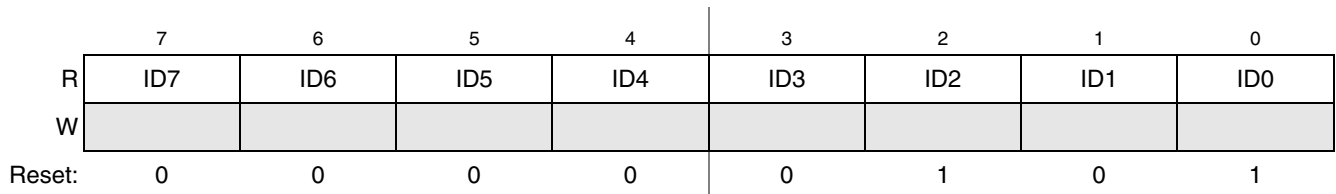


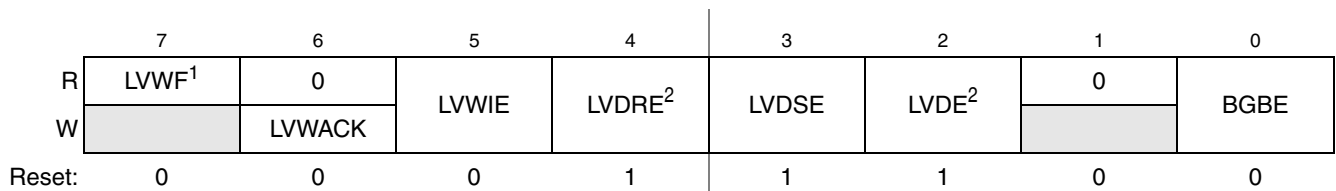
Figure 5-6. System Device Identification Register — Low (SDIDL)

Table 5-8. SDIDL Register Field Descriptions

Field	Description
7–0 ID[7–0]	<b>Part Identification Number</b> — Each derivative in the ColdFire family has a unique identification number. The MCF51AG128 series microcontrollers are hard coded to the value 0xC05. See also ID bits in <a href="#">Table 5-7</a> .

## 5.8.6 System Power Management Status and Control 1 Register (SPMSC1)

This high page register contains status and control bits to support the low voltage detect function, and to enable the bandgap voltage reference for the ADC module. This register should be written during the reset initialization program to set the desired controls even if the desired settings are the same as the reset settings.



<sup>1</sup> LVWF is set when  $V_{Supply}$  transitions below the trip point or after reset and  $V_{Supply}$  is already below  $V_{LVW}$ .

<sup>2</sup> This bit can be written only one time after reset. Additional writes are ignored.

Figure 5-7. System Power Management Status and Control 1 Register (SPMSC1)

Table 5-9. SPMSC1 Register Field Descriptions

Field	Description
7 LVWF	<b>Low-Voltage Warning Flag</b> — The LVWF bit indicates the low-voltage warning status. 0 Low-voltage warning is not present. 1 Low-voltage warning is present or was present.
6 LVWACK	<b>Low-Voltage Warning Acknowledge</b> — If LVWF is set, a low-voltage condition has occurred. To acknowledge this low-voltage warning, write 1 to LVWACK that automatically clears LVWF if the low-voltage warning is no longer present.
5 LVWIE	<b>Low-Voltage Warning Interrupt Enable</b> — This bit enables hardware interrupt requests for LVWF. 0 Hardware interrupt disabled (use polling). 1 Request a hardware interrupt when LVWF is set.
4 LVDRE	<b>Low-Voltage Detect Reset Enable</b> — This write-once bit enables LVDF events to generate a hardware reset (provided LVDE is set). 0 LVDF does not generate hardware resets. 1 Force an MCU reset when an enabled low-voltage detect event occurs.
3 LVDSE	<b>Low-Voltage Detect Stop Enable</b> — If LVDE is set, this read/write bit determines whether the low-voltage detect function operates when the MCU is in stop mode. 0 Low-voltage detect disabled during stop mode. 1 Low-voltage detect enabled during stop mode.
2 LVDE	<b>Low-Voltage Detect Enable</b> — This write-once bit enables low-voltage detect logic and qualifies the operation of other bits in this register. 0 LVD logic disabled. 1 LVD logic enabled.
0 BGBE	<b>Bandgap Buffer Enable</b> — This bit enables an internal buffer for the bandgap voltage reference for use by the ADC and DAC modules on one of its internal channels. 0 Bandgap buffer disabled. 1 Bandgap buffer enabled.

### 5.8.7 System Power Management Status and Control 2 Register (SPMSC2)

This high page register contains status and control bits to configure the low power run and wait modes and configure the stop mode behavior of the microcontroller. See [Section 3.7, “Wait Mode,”](#) and [Section 3.8, “Stop Modes,”](#) for more information.

SPMSC2 is not reset when exiting from STOP2.

	7	6	5	4	3	2	1	0
R	0	0	LVDV <sup>1</sup>	LVWV	PPDF	0	0	PPDC <sup>1</sup>
W						PPDACK		
POR Reset:	0	0	0	0	0	0	0	0
LVD Reset:	0	0	U	U	0	0	0	0
Any other Reset:	0	0	U	U	0	0	0	0

 = Unimplemented or Reserved      U = Unaffected by reset

<sup>1</sup> This bit can be written only one time after power-on reset. Additional writes are ignored.

**Figure 5-8. System Power Management Status and Control 2 Register (SPMSC2)**

**Table 5-10. SPMSC2 Register Field Descriptions**

Field	Description
7-6	Reserved, must be cleared.
5 LVDV	<b>Low-Voltage Detect Voltage Select</b> — The LVDV bit selects the LVD trip point voltage ( $V_{LVD}$ ). 0 Low trip point selected ( $V_{LVD} = V_{LVDDL}$ ). 1 High trip point selected ( $V_{LVD} = V_{LVDH}$ ).
4 LVWV	<b>Low-Voltage Warning Voltage Select</b> — The LVWV bit selects the LVW trip point voltage ( $V_{LVW}$ ). 0 Low trip point selected ( $V_{LVW} = V_{LVWL}$ ). 1 High trip point selected ( $V_{LVW} = V_{LVWH}$ ).
3 PPDF	<b>Partial Power-Down Flag</b> — This read-only status bit indicates that the microcontroller has recovered from the stop2 mode. 0 Microcontroller has not recovered from stop2 mode. 1 Microcontroller recovered from stop2 mode.
2 PPDACK	<b>Partial Power-Down Acknowledge</b> — Writing 1 to PPDACK clears the PPDF bit.
0 PPDC	<b>Partial Power-Down Control</b> — The PPDC bit controls which power-down mode is selected. PPDC is write once only bit. 0 Stop3 low power mode enabled. 1 Stop2 partial power-down mode enabled.

**Table 5-11. LVD and LVW Trip Point Typical Values<sup>1</sup>**

LVDV:LVWV	LVW Trip Point	LVD Trip Point
00	$V_{LVWL} = 2.74 \text{ V}$	$V_{LVDDL} = 2.56 \text{ V}$
01	$V_{LVWL} = 2.92 \text{ V}$	
10	$V_{LVWL} = 4.3 \text{ V}$	$V_{LVDDL} = 4.0 \text{ V}$
11	$V_{LVWL} = 4.6 \text{ V}$	

<sup>1</sup> See the MCF51AG128 *Data Sheet* (document MCF51AG128) for minimum and maximum values.

## 5.8.8 System Options 2 (SOPT2) Register

This high page register contains bits to configure MCU specific features on the MCF51AG128 series devices.

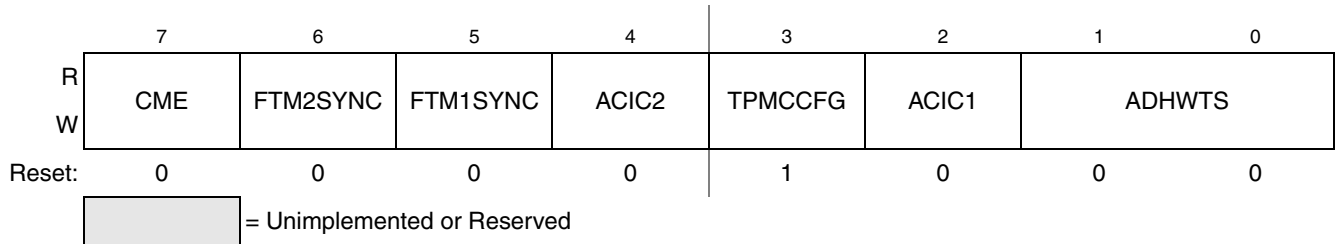


Figure 5-9. System Integration Module Options Register 2 (SOPT2)

Table 5-12. SOPT2 Register Field Descriptions

Field	Description
7 CME	<b>Clock Monitor Enable</b> — Determines if a reset request is made following a loss of external clock indication. The CME bit should only be set to a logic 1 when either the ICS is in an operational mode that uses the external clock (FEE, FBE or BLPE) or the external reference is enabled (ERCLKEN=1 in the ICSC2 register). Also internal reference clock MUST be enabled (IRCLKEN=1 in the ICSC1 register). Whenever the CME bit is set to a logic 1, the value of the RANGE bit in the ICSC2 register should not be changed. 0 Clock monitor is disabled. 1 Generate a reset request on loss of external clock. (EREFSTEN=1 and IREFSTEN=1, if CME is required in the stop modes)
6 FTM2SYNC	<b>FTM2 Synchronization Trigger</b> — Writing a 1 to this bit generates a PWM synchronization trigger to the FTM2 modules. 0 No trigger generated. 1 Generate a PWM synchronization trigger to the FTM2 modules.
5 FTM1SYNC	<b>FTM1 Synchronization Trigger</b> — Writing a 1 to this bit generates a PWM synchronization trigger to the FTM1 modules. 0 No trigger generated. 1 Generate a PWM synchronization trigger to the FTM1 modules.
4 ACIC2	<b>Analog Comparator 2 to Input Capture Enable</b> — This bit connects the output of HSCMP2 to TPM3 input channel 0. See <a href="#">Chapter 9, “High-Speed Comparator (S08HSCMPV2) and Chapter 25, “Timer/PWM Module (S08TPMV3) for more details on this feature.</a> 0 HSCMP2 output not connected to TPM3 input channel 0. 1 HSCMP2 output connected to TPM3 input channel 0.
3 TPMCCFG	<b>TPM Clock Configuration</b> — Configures the timer/pulse-width modulator clock signal. 0 TPMCLK is available to FTM1, FTM2, and TPM3 via the IRQ pin; FTM1CLK and FTM2CLK are not available. 1 FTM1CLK, FTM2CLK, and TPMCLK are available to FTM1, FTM2, and TPM3, respectively.

Table 5-12. SOPT2 Register Field Descriptions (continued)

Field	Description
2 ACIC1	<b>Analog Comparator 1 to Input Capture Enable</b> — This bit connects the output of HSCMP1 to FTM1 input channel 0. See <a href="#">Chapter 9, “High-Speed Comparator (S08HSCMPV2),”</a> and <a href="#">Chapter 13, “FlexTimer Module (S08FTMV3)”</a> for more details on this feature. 0 HSCMP output not connected to FTM1 input channel 0. 1 HSCMP output connected to FTM1 input channel 0.
1:0 ADHWTS	<b>ADC Hardware Trigger Source</b> - These two bits select the input source to the ADHWT (ADC hardware trigger) 00 TriggerA from PDB 01 Output from iEvent_0 10 RTC counter overflow 11 Reserved

### 5.8.9 System Clock Gating Control 1 Register (SCGC1)

This high page register contains control bits to enable or disable the bus clock to the TPM3, FTM<sub>x</sub>, ADC, DAC, IIC, and SCL<sub>x</sub> modules. Gating off the clocks to unused peripherals reduces the microcontroller’s run and wait currents. See [Section 5.7, “Peripheral Clock Gating,”](#) for more information.

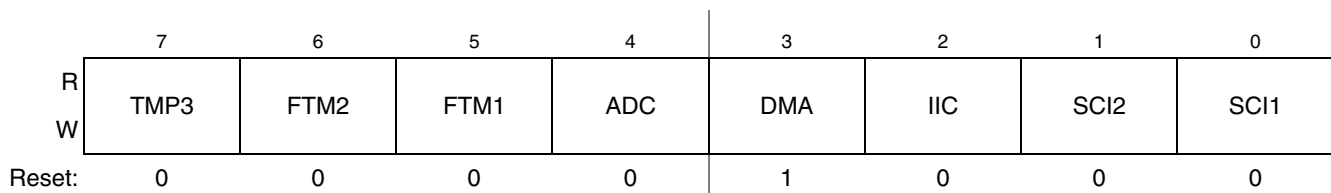


Figure 5-10. System Clock Gating Control 1 Register (SCGC1)

Table 5-13. SCGC1 Register Field Descriptions

Field	Description
7 TPM3	<b>TPM3 Clock Gate Control</b> — This bit controls the clock gate to the TPM3 module. 0 Bus clock to the TPM3 module is disabled. 1 Bus clock to the TPM3 module is enabled.
6 FTM2	<b>FTM2 Clock Gate Control</b> — This bit controls the clock gate to the FTM2 module. 0 Bus clock to the FTM2 module is disabled. 1 Bus clock to the FTM2 module is enabled.
5 FTM1	<b>FTM1 Clock Gate Control</b> — This bit controls the clock gate to the FTM1 module. 0 Bus clock to the FTM1 module is disabled. 1 Bus clock to the FTM1 module is enabled.
4 ADC	<b>ADC Clock Gate Control</b> — This bit controls the clock gate to the ADC module. 0 Bus clock to the ADC module is disabled. 1 Bus clock to the ADC module is enabled.
3 DMA	<b>DMA Clock Gate Control</b> — This bit controls the clock gate to the DMA module. 0 Bus clock to the DMA module is disabled. 1 Bus clock to the DMA module is enabled.



Table 5-13. SCGC1 Register Field Descriptions (continued)

Field	Description
2 IIC	<b>IIC Clock Gate Control</b> — This bit controls the clock gate to the IIC module. 0 Bus clock to the IIC module is disabled. 1 Bus clock to the IIC module is enabled.
1 SCI2	<b>SCI2 Clock Gate Control</b> — This bit controls the clock gate to the SCI2 module. 0 Bus clock to the SCI2 module is disabled. 1 Bus clock to the SCI2 module is enabled.
0 SCI1	<b>SCI1 Clock Gate Control</b> — This bit controls the clock gate to the SCI1 module. 0 Bus clock to the SCI1 module is disabled. 1 Bus clock to the SCI1 module is enabled.

### 5.8.10 System Clock Gating Control 2 Register (SCGC2)

This high page register contains control bits to enable or disable the bus clock to the CRC, FLASH, IRQ, PDB, iEvent, RTC, and SPIx modules. Gating off the clocks to unused peripherals reduces the microcontroller's run and wait currents. See [Section 5.7, "Peripheral Clock Gating,"](#) for more information.

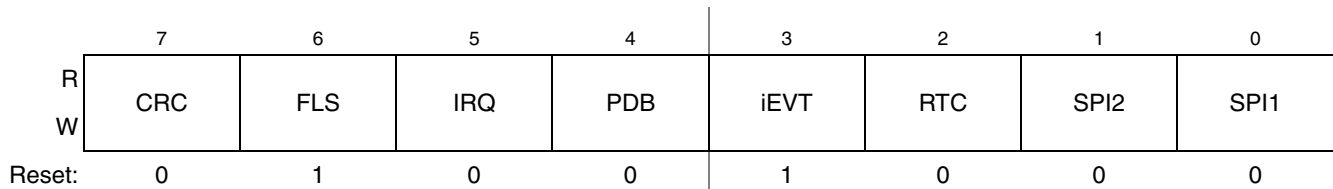


Figure 5-11. System Clock Gating Control 2 Register (SCGC2)

Table 5-14. SCGC2 Register Field Descriptions

Field	Description
7 CRC	<b>CRC Clock Gate Control</b> — This bit controls the clock gate to the CRC module. 0 Bus clock to the CRC module is disabled. 1 Bus clock to the CRC module is enabled.
6 FLS	<b>FTSR Clock Gate Control</b> — This bit controls the bus clock gate to the flash registers. This bit does not affect normal program execution from the flash array. Only the clock to the flash control registers is affected. 0 Bus clock to flash registers is disabled. 1 Bus clock to flash registers is enabled.
5 IRQ	<b>IRQ Clock Gate Control</b> — This bit controls the bus clock gate to the IRQ module. 0 Bus clock to the IRQ module is disabled. 1 Bus clock to the IRQ module is enabled.
4 PDB	<b>PDB Clock Gate Control</b> — This bit controls the clock gate to the PDB modules. 0 Bus clock to the PDB modules is disabled. 1 Bus clock to the PDB modules is enabled.
3 iEVT	<b>iEvent Clock Gate Control</b> — This bit controls the clock gate to the iEvent modules. 0 Bus clock to the iEvent modules is disabled. 1 Bus clock to the iEvent modules is enabled.

Table 5-14. SCGC2 Register Field Descriptions (continued)

Field	Description
2 RTC	<b>RTC Clock Gate Control</b> — This bit controls the bus clock gate to the RTC module. 0 Bus clock to the RTC module is disabled. 1 Bus clock to the RTC module is enabled.
1 SPI2	<b>SPI2 Clock Gate Control</b> — This bit controls the clock gate to the SPI2 module. 0 Bus clock to the SPI2 module is disabled. 1 Bus clock to the SPI2 module is enabled.
0 SPI1	<b>SPI1 Clock Gate Control</b> — This bit controls the clock gate to the SPI1 module. 0 Bus clock to the SPI1 module is disabled. 1 Bus clock to the SPI1 module is enabled.

### 5.8.11 System Clock Gating Control 3 Register (SCGC3)

This high page register contains control bits to enable or disable the bus clock to the HSCMPx, DACx, and EWM modules. Gating off the clocks to unused peripherals reduces the microcontroller's run and wait currents. See [Section 5.7, "Peripheral Clock Gating,"](#) for more information.

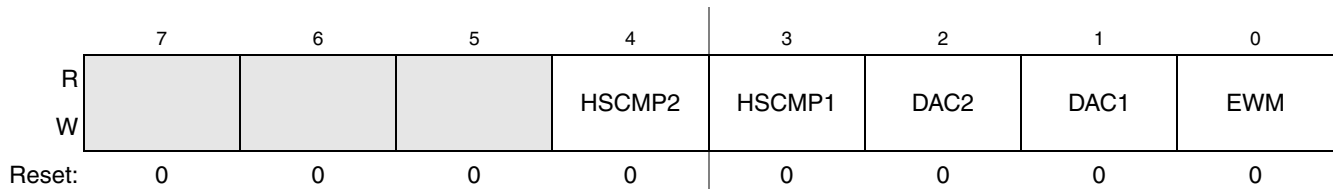


Figure 5-12. System Clock Gating Control 3 Register (SCGC3)

Table 5-15. SCGC2 Register Field Descriptions

Field	Description
4 HSCMP2	<b>HSCMP2 Clock Gate Control</b> — This bit controls the clock gate to the HSCMP2 modules. 0 Bus clock to the HSCMP2 modules is disabled. 1 Bus clock to the HSCMP2 modules is enabled.
3 HSCMP1	<b>HSCMP1 Clock Gate Control</b> — This bit controls the clock gate to the HSCMP1 modules. 0 Bus clock to the HSCMP1 modules is disabled. 1 Bus clock to the HSCMP1 modules is enabled.
2 DAC2	<b>DAC2 Clock Gate Control</b> — This bit controls the bus clock gate to the DAC2 module. Only the bus clock is gated; the clock resource is not controlled by this bit. 0 Bus clock to the DAC2 module is disabled. 1 Bus clock to the DAC2 module is enabled.
1 DAC1	<b>DAC1 Clock Gate Control</b> — This bit controls the clock gate to the DAC1 module. 0 Bus clock to the DAC1 module is disabled. 1 Bus clock to the DAC1 module is enabled.
0 EWM	<b>EWM Clock Gate Control</b> — This bit controls the clock gate to the EWM module. 0 Bus clock to the EWM module is disabled. 1 Bus clock to the EWM module is enabled.

## 5.8.12 System eGPIO Interrupt Status 1 (SEIS1) Register

This high page register includes read-only status flags to indicate the source of the most recent eGPIO Interrupt. Several on these flags exist on MCF51AG128 devices.

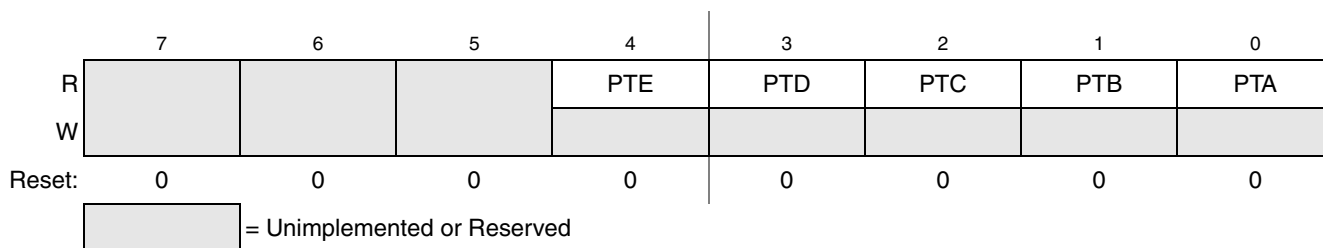


Figure 5-13. System eGPIO Interrupt Status 1 Register (SEIS1)

Table 5-16. SEIS1 Register Field Descriptions

Field	Description
4 PTE	<b>PORT E Interrupt Status</b> — Shows the interrupt status from Port E. 0 No Port E Interrupt generated. 1 Generate an Interrupt from Port E.
3 PTD	<b>PORT D Interrupt Status</b> — Shows the interrupt status from Port D. 0 No Port D Interrupt generated. 1 Generate an Interrupt from Port D.
2 PTC	<b>PORT C Interrupt Status</b> — Shows the interrupt status from Port C. 0 No Port C Interrupt generated. 1 Generate an Interrupt from Port C.
1 PTB	<b>PORT B Interrupt Status</b> — Shows the interrupt status from Port B. 0 No Port B Interrupt generated. 1 Generate an Interrupt from Port B.
0 PTA	<b>PORT A Interrupt Status</b> — Shows the interrupt status from Port A. 0 No Port A Interrupt generated. 1 Generate an Interrupt from Port A.

## 5.8.13 System eGPIO Interrupt Status 2 (SEIS2) Register

This high page register includes read-only status flags to indicate the source of the most recent eGPIO Interrupt. Several on the MCF51AG128 devices.

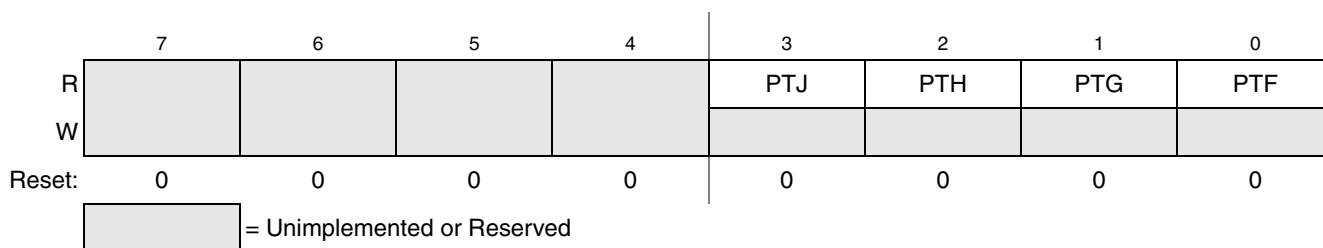


Figure 5-14. System eGPIO Interrupt Status 2 Register (SEIS2)

Table 5-17. SEIS2 Register Field Descriptions

Field	Description
3 PTJ	<b>PORT J Interrupt Status</b> — Shows the interrupt status from Port J. 0 No Port J Interrupt generated. 1 Generate an Interrupt from Port J.
2 PTH	<b>PORT H Interrupt Status</b> — Shows the interrupt status from Port H. 0 No Port H Interrupt generated. 1 Generate an Interrupt from Port H.
1 PTG	<b>PORT G Interrupt Status</b> — Shows the interrupt status from Port G. 0 No Port G Interrupt generated. 1 Generate an Interrupt from Port G.
0 PTF	<b>PORT F Interrupt Status</b> — Shows the interrupt status from Port F. 0 No Port F Interrupt generated. 1 Generate an Interrupt from Port F.

### 5.8.14 System ADC Pin Enable 1 Register (SAPE1)

These pin enable registers disable the I/O port control of MCU pins that are used as analog inputs. SAPE1 controls the pins associated with 0–7 channels of the ADC module.

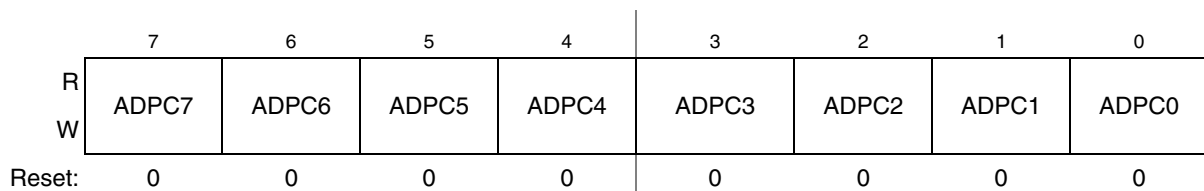


Figure 5-15. System ADC Pin Enable 1 Register (SAPE1)

Table 5-18. SAPE1 Register Field Descriptions

Field	Description
7 ADPC7	<b>ADC Pin Control 7</b> — This bit controls the pin associated with channel AD7. 0 AD7 pin I/O control disabled. 1 AD7 pin I/O control enabled.
6 ADPC6	<b>ADC Pin Control 6</b> — This bit controls the pin associated with channel AD6. 0 AD6 pin I/O control disabled. 1 AD6 pin I/O control enabled.
5 ADPC5	<b>ADC Pin Control 5</b> — This bit controls the pin associated with channel AD5. 0 AD5 pin I/O control disabled. 1 AD5 pin I/O control enabled.
4 ADPC4	<b>ADC Pin Control 4</b> — This bit controls the pin associated with channel AD4. 0 AD4 pin I/O control disabled. 1 AD4 pin I/O control enabled.
3 ADPC3	<b>ADC Pin Control 3</b> — This bit controls the pin associated with channel AD3. 0 AD3 pin I/O control disabled. 1 AD3 pin I/O control enabled.

Table 5-18. SAPE1 Register Field Descriptions (continued)

Field	Description
2 ADPC2	<b>ADC Pin Control 2</b> — This bit controls the pin associated with channel AD2. 0 AD2 pin I/O control disabled. 1 AD2 pin I/O control enabled.
1 ADPC1	<b>ADC Pin Control 1</b> — This bit controls the pin associated with channel AD1. 0 AD1 pin I/O control disabled. 1 AD1 pin I/O control enabled.
0 ADPC0	<b>ADC Pin Control 0</b> — This bit controls the pin associated with channel AD0. 0 AD0 pin I/O control disabled. 1 AD0 pin I/O control enabled.

### 5.8.15 System ADC Pin Enable 2 Register (SAPE2)

This pin enable register controls 8–15 channels of the ADC module.

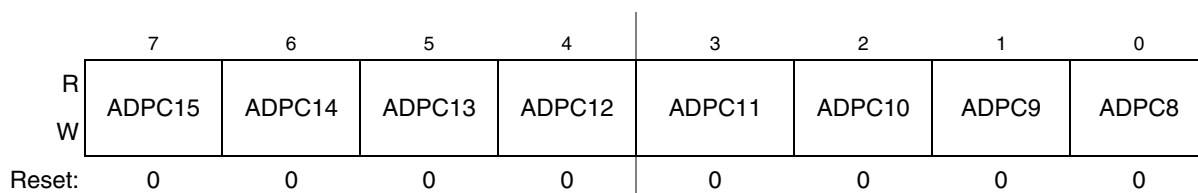


Figure 5-16. System ADC Pin Enable 2 Register (SAPE2)

Table 5-19. SAPE2 Register Field Descriptions

Field	Description
7 ADPC15	<b>ADC Pin Control 15</b> — This bit controls the pin associated with channel AD15. 0 AD15 pin I/O control disabled. 1 AD15 pin I/O control enabled.
6 ADPC14	<b>ADC Pin Control 14</b> — This bit controls the pin associated with channel AD14. 0 AD14 pin I/O control disabled. 1 AD14 pin I/O control enabled.
5 ADPC13	<b>ADC Pin Control 13</b> — This bit controls the pin associated with channel AD13. 0 AD13 pin I/O control disabled. 1 AD13 pin I/O control enabled.
4 ADPC12	<b>ADC Pin Control 12</b> — This bit controls the pin associated with channel AD12. 0 AD12 pin I/O control disabled. 1 AD12 pin I/O control enabled.
3 ADPC11	<b>ADC Pin Control 11</b> — This bit controls the pin associated with channel AD11. 0 AD11 pin I/O control disabled. 1 AD11 pin I/O control enabled.
2 ADPC10	<b>ADC Pin Control 10</b> — This bit controls the pin associated with channel AD10. 0 AD10 pin I/O control disabled. 1 AD10 pin I/O control enabled.

Table 5-19. SAPE2 Register Field Descriptions (continued)

Field	Description
1 ADPC9	<b>ADC Pin Control 9</b> — This bit controls the pin associated with channel AD9. 0 AD9 pin I/O control disabled. 1 AD9 pin I/O control enabled.
0 ADPC8	<b>ADC Pin Control 8</b> — This bit controls the pin associated with channel AD8. 0 AD8 pin I/O control disabled. 1 AD8 pin I/O control enabled.

### 5.8.16 System ADC Pin Enable 3 Register (SAPE3)

This register controls 16–23 channels of the ADC module.

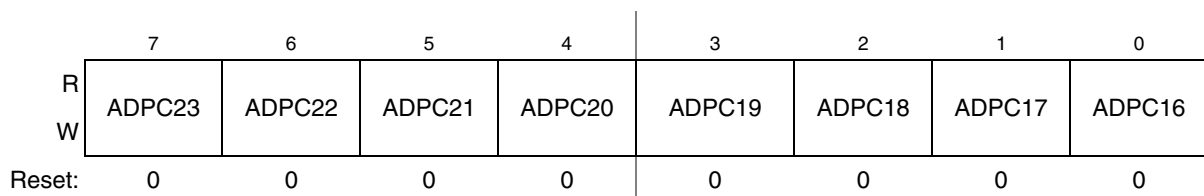


Figure 5-17. System ADC Pin Enable 3 Register (SAPE3)

Table 5-20. SAPE3 Register Field Descriptions

Field	Description
7 ADPC23	<b>ADC Pin Control 23</b> — This bit controls the pin associated with channel AD23. 0 AD23 pin I/O control disabled. 1 AD23 pin I/O control enabled.
6 ADPC22	<b>ADC Pin Control 22</b> — This bit controls the pin associated with channel AD22. 0 AD22 pin I/O control disabled. 1 AD22 pin I/O control enabled.
5 ADPC21	<b>ADC Pin Control 21</b> — This bit controls the pin associated with channel AD21. 0 AD21 pin I/O control disabled. 1 AD21 pin I/O control enabled.
4 ADPC20	<b>ADC Pin Control 20</b> — This bit controls the pin associated with channel AD20. 0 AD20 pin I/O control disabled. 1 AD20 pin I/O control enabled.
3 ADPC19	<b>ADC Pin Control 19</b> — This bit controls the pin associated with channel AD19. 0 AD19 pin I/O control disabled. 1 AD19 pin I/O control enabled.
2 ADPC18	<b>ADC Pin Control 18</b> — This bit controls the pin associated with channel AD18. 0 AD18 pin I/O control disabled. 1 AD18 pin I/O control enabled.

Table 5-20. SAPE3 Register Field Descriptions (continued)

Field	Description
1 ADPC17	<b>ADC Pin Control 17</b> — This bit controls the pin associated with channel AD17. 0 AD17 pin I/O control disabled. 1 AD17 pin I/O control enabled.
0 ADPC16	<b>ADC Pin Control 16</b> — This bit controls the pin associated with channel AD16. 0 AD16 pin I/O control disabled. 1 AD16 pin I/O control enabled.

### 5.8.17 System Pin Positioning Register (SPINPS)

This register controls the pins associated with channels 2–5 of the FTM2 and channels 0–1 of the TPM3.

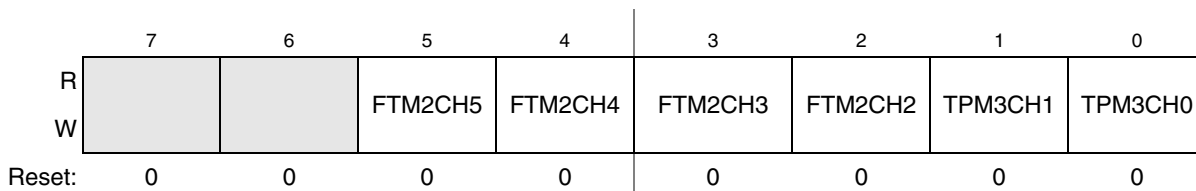


Figure 5-18. System Pin Positioning Register (SPINPS)

Table 5-21. SPINPS Register Field Descriptions

Field	Description
5 FTM2CH5	<b>FTM2 Channel 5 Pin Control</b> — This bit controls the pin associated with channel 5 of FTM2. 0 Channel 5 is connected to PTJ4. 1 Channel 5 is connected to PTH3.
4 FTM2CH4	<b>FTM2 Channel 4 Pin Control</b> — This bit controls the pin associated with channel 4 of FTM2. 0 Channel 4 is connected to PTJ5. 1 Channel 4 is connected to PTH2.
3 FTM2CH3	<b>FTM2 Channel 3 Pin Control</b> — This bit controls the pin associated with channel 3 of FTM2. 0 Channel 3 is connected to PTJ6. 1 Channel 3 is connected to PTH1.
2 FTM2CH2	<b>FTM2 Channel 2 Pin Control</b> — This bit controls the pin associated with channel 2 of FTM2. 0 Channel 2 is connected to PTJ7. 1 Channel 2 is connected to PTH0.
1 TPM3CH1	<b>TPM3 Channel 1 Pin Control</b> — This bit controls the pin associated with channel 1 of TPM3. 0 Channel 1 is connected to PTB1. 1 Channel 1 is connected to PTG1.
0 TPM3CH0	<b>TPM3 Channel 0 Pin Out Control</b> — This bit controls the pin associated with channel 0 of TPM3. 0 Channel 0 is connected to PTB0. 1 Channel 0 is connected to PTG0.

### 5.8.18 System PortA Input Buffer Enable Register (SIMPTA)

When a pin is configured as general purpose I/O pin, disabling its input buffer can save the supply current on low power modes. The SIMPTA register controls the pin input buffers on Port A.

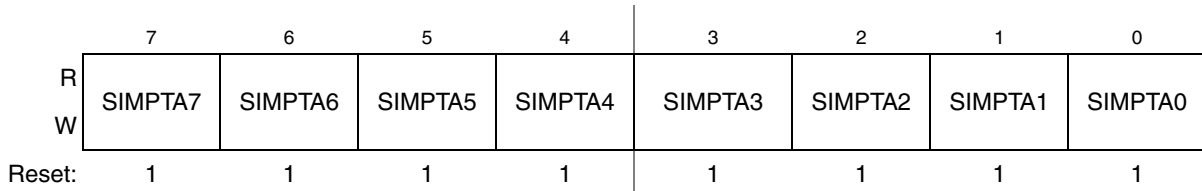


Figure 5-19. PortA input Buffer Enable Register (SIMPTA)

Table 5-22. SIMPTA Register Field Descriptions

Field	Description
7 SIMPTA7	<b>PTA7 Input Buffer Control</b> — This bit controls the input buffer associated with PTA7. 0 PTA7 pin input buffer disabled. 1 PTA7 pin input buffer enabled.
6 SIMPTA6	<b>PTA6 Input Buffer Control</b> — This bit controls the input buffer associated with PTA6. 0 PTA6 pin input buffer disabled. 1 PTA6 pin input buffer enabled.
5 SIMPTA5	<b>PTA5 Input Buffer Control</b> — This bit controls the input buffer associated with PTA5. 0 PTA5 pin input buffer disabled. 1 PTA5 pin input buffer enabled.
4 SIMPTA4	<b>PTA4 Input Buffer Control</b> — This bit controls the input buffer associated with PTA4. 0 PTA4 pin input buffer disabled. 1 PTA4 pin input buffer enabled.
3 SIMPTA3	<b>PTA3 Input Buffer Control</b> — This bit controls the input buffer associated with PTA3. 0 PTA3 pin input buffer disabled. 1 PTA3 pin input buffer enabled.
2 SIMPTA2	<b>PTA2 Input Buffer Control</b> — SIMPTA2 is used to control the input buffer associated with PTA2. 0 PTA2 pin input buffer disabled. 1 PTA2 pin input buffer enabled.
1 SIMPTA1	<b>PTA1 Input Buffer Control</b> — This bit controls the input buffer associated with PTA1. 0 PTA1 pin input buffer disabled. 1 PTA1 pin input buffer enabled.
0 SIMPTA0	<b>PTA0 Input Buffer Control</b> — This bit controls the input buffer associated with PTA0. 0 PTA0 pin input buffer disabled. 1 PTA0 pin input buffer enabled.

### 5.8.19 System PortB Input Buffer Enable Register (SIMPTB)

The SIMPTB register controls the pin input buffers on Port B.

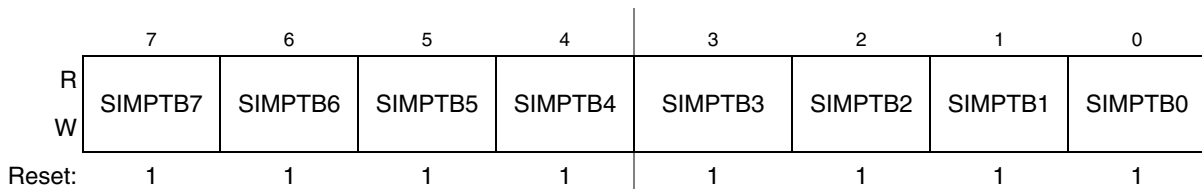


Figure 5-20. PortB input Buffer Enable Register (SIMPTB)



Table 5-23. SIMPTB Register Field Descriptions

Field	Description
7 SIMPTB7	<b>PTB7 Input Buffer Control</b> — This bit controls the input buffer associated with PTB7. 0 PTB7 pin input buffer disabled. 1 PTB7 pin input buffer enabled.
6 SIMPTB6	<b>PTB6 Input Buffer Control</b> — This bit controls the input buffer associated with PTB6. 0 PTB6 pin input buffer disabled. 1 PTB6 pin input buffer enabled.
5 SIMPTB5	<b>PTB5 Input Buffer Control</b> — This bit controls the input buffer associated with PTB5. 0 PTB5 pin input buffer disabled. 1 PTB5 pin input buffer enabled.
4 SIMPTB4	<b>PTB4 Input Buffer Control</b> — This bit controls the input buffer associated with PTB4. 0 PTB4 pin input buffer disabled. 1 PTB4 pin input buffer enabled.
3 SIMPTB3	<b>PTB3 Input Buffer Control</b> — This bit controls the input buffer associated with PTB3. 0 PTB3 pin input buffer disabled. 1 PTB3 pin input buffer enabled.
2 SIMPTB2	<b>PTB2 Input Buffer Control</b> — This bit controls the input buffer associated with PTB2. 0 PTB2 pin input buffer disabled. 1 PTB2 pin input buffer enabled.
1 SIMPTB1	<b>PTB1 Input Buffer Control</b> — This bit controls the input buffer associated with PTB1. 0 PTB1 pin input buffer disabled. 1 PTB1 pin input buffer enabled.
0 SIMPTB0	<b>PTB0 Input Buffer Control</b> — This bit controls the input buffer associated with PTB0. 0 PTB0 pin input buffer disabled. 1 PTB0 pin input buffer enabled.

## 5.8.20 System PortC Input Buffer Enable Register (SIMPTC)

The SIMPTC register controls the pin input buffers on Port C.

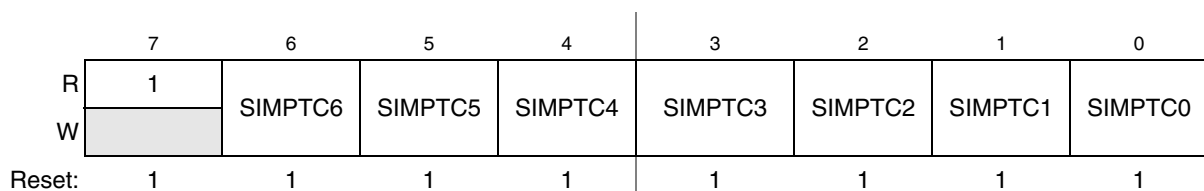


Figure 5-21. PortC input Buffer Enable Register (SIMPTC)

Table 5-24. SIMPTC Register Field Descriptions

Field	Description
7 RSVD	<b>Reserved</b> — Always read 1, write has no effect.
6 SIMPTC6	<b>PTC6 Input Buffer Control</b> — This bit controls the input buffer associated with PTC6. 0 PTC6 pin input buffer disabled. 1 PTC6 pin input buffer enabled.
5 SIMPTC5	<b>PTC5 Input Buffer Control</b> — This bit controls the input buffer associated with PTC5. 0 PTC5 pin input buffer disabled. 1 PTC5 pin input buffer enabled.
4 SIMPTC4	<b>PTC4 Input Buffer Control</b> — This bit controls the input buffer associated with PTC4. 0 PTC4 pin input buffer disabled. 1 PTC4 pin input buffer enabled.
3 SIMPTC3	<b>PTC3 Input Buffer Control</b> — This bit controls the input buffer associated with PTC3. 0 PTC3 pin input buffer disabled. 1 PTC3 pin input buffer enabled.
2 SIMPTC2	<b>PTC2 Input Buffer Control</b> — This bit controls the input buffer associated with PTC2. 0 PTC2 pin input buffer disabled. 1 PTC2 pin input buffer enabled.
1 SIMPTC1	<b>PTC1 Input Buffer Control</b> — This bit controls the input buffer associated with PTC1. 0 PTC1 pin input buffer disabled. 1 PTC1 pin input buffer enabled.
0 SIMPTC0	<b>PTC0 Input Buffer Control</b> — This bit controls the input buffer associated with PTC0. 0 PTC0 pin input buffer disabled. 1 PTC0 pin input buffer enabled.

### 5.8.21 System PortD Input Buffer Enable Register (SIMPTD)

The SIMPTD register controls the pin input buffers on Port D.

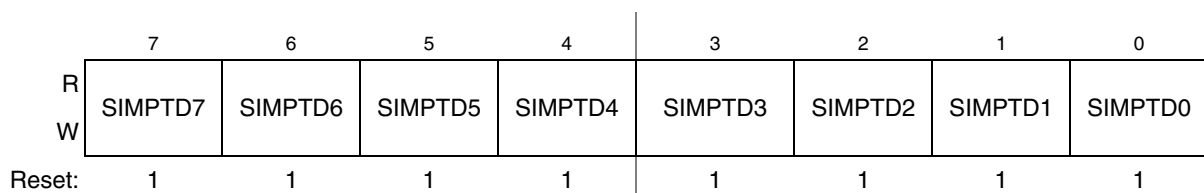


Figure 5-22. PortD input Buffer Enable Register (SIMPTD)

Table 5-25. SIMPTD Register Field Descriptions

Field	Description
7 SIMPTD7	<b>PTD7 Input Buffer Control</b> — This bit controls the input buffer associated with PTD7. 0 PTD7 pin input buffer disabled. 1 PTD7 pin input buffer enabled.
6 SIMPTD6	<b>PTD6 Input Buffer Control</b> — This bit controls the input buffer associated with PTD6. 0 PTD6 pin input buffer disabled. 1 PTD6 pin input buffer enabled.
5 SIMPTD5	<b>PTD5 Input Buffer Control</b> — This bit controls the input buffer associated with PTD5. 0 PTD5 pin input buffer disabled. 1 PTD5 pin input buffer enabled.
4 SIMPTD4	<b>PTD4 Input Buffer Control</b> — This bit controls the input buffer associated with PTD4. 0 PTD4 pin input buffer disabled. 1 PTD4 pin input buffer enabled.
3 SIMPTD3	<b>PTD3 Input Buffer Control</b> — This bit controls the input buffer associated with PTD3. 0 PTD3 pin input buffer disabled. 1 PTD3 pin input buffer enabled.
2 SIMPTD2	<b>PTD2 Input Buffer Control</b> — This bit controls the input buffer associated with PTD2. 0 PTD2 pin input buffer disabled. 1 PTD2 pin input buffer enabled.
1 SIMPTD1	<b>PTD1 Input Buffer Control</b> — This bit controls the input buffer associated with PTD1. 0 PTD1 pin input buffer disabled. 1 PTD1 pin input buffer enabled.
0 SIMPTD0	<b>PTD0 Input Buffer Control</b> — This bit controls the input buffer associated with PTD0. 0 PTD0 pin input buffer disabled. 1 PTD0 pin input buffer enabled.

### 5.8.22 System PortE Input Buffer Enable Register (SIMPTE)

The SIMPTE register controls the pin input buffers on Port E.

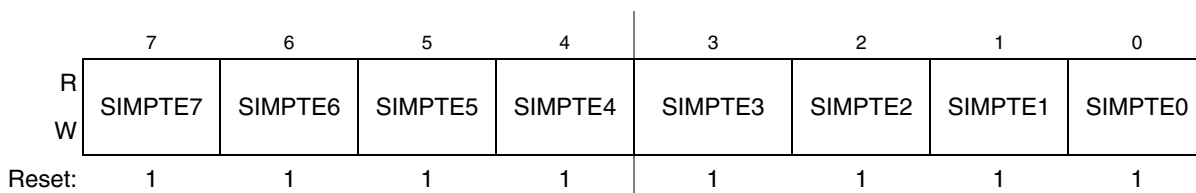


Figure 5-23. PortE Input Buffer Enable Register (SIMPTE)

Table 5-26. SIMPTE Register Field Descriptions

Field	Description
7 SIMPTE7	<b>PTE7 Input Buffer Control</b> — This bit controls the input buffer associated with PTE7. 0 PTE7 pin input buffer disabled. 1 PTE7 pin input buffer enabled.
6 SIMPTE6	<b>PTE6 Input Buffer Control</b> — This bit controls the input buffer associated with PTE6. 0 PTE6 pin input buffer disabled. 1 PTE6 pin input buffer enabled.
5 SIMPTE5	<b>PTE5 Input Buffer Control</b> — This bit controls the input buffer associated with PTE5. 0 PTE5 pin input buffer disabled. 1 PTE5 pin input buffer enabled.
4 SIMPTE4	<b>PTE4 Input Buffer Control</b> — This bit controls the input buffer associated with PTE4. 0 PTE4 pin input buffer disabled. 1 PTE4 pin input buffer enabled.
3 SIMPTE3	<b>PTE3 Input Buffer Control</b> — This bit controls the input buffer associated with PTE3. 0 PTE3 pin input buffer disabled. 1 PTE3 pin input buffer enabled.
2 SIMPTE2	<b>PTE2 Input Buffer Control</b> — This bit controls the input buffer associated with PTE2. 0 PTE2 pin input buffer disabled. 1 PTE2 pin input buffer enabled.
1 SIMPTE1	<b>PTE1 Input Buffer Control</b> — This bit controls the input buffer associated with PTE1. 0 PTE1 pin input buffer disabled. 1 PTE1 pin input buffer enabled.
0 SIMPTE0	<b>PTE0 Input Buffer Control</b> — This bit controls the input buffer associated with PTE0. 0 PTE0 pin input buffer disabled. 1 PTE0 pin input buffer enabled.

### 5.8.23 System PortF Input Buffer Enable Register (SIMPTF)

The SIMPTF register controls the pin input buffers on Port F.

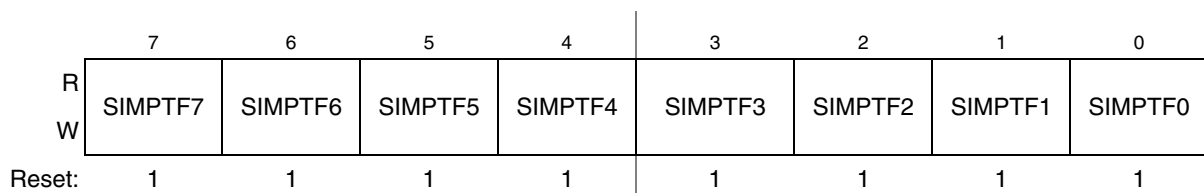


Figure 5-24. PortF input Buffer Enable Register (SIMPTF)

Table 5-27. SIMPTF Register Field Descriptions

Field	Description
7 SIMPTF7	<b>PTF7 Input Buffer Control</b> — This bit controls the input buffer associated with PTF7. 0 PTF7 pin input buffer disabled. 1 PTF7 pin input buffer enabled.
6 SIMPTF6	<b>PTF6 Input Buffer Control</b> — This bit controls the input buffer associated with PTF6. 0 PTF6 pin input buffer disabled. 1 PTF6 pin input buffer enabled.
5 SIMPTF5	<b>PTF5 Input Buffer Control</b> — This bit controls the input buffer associated with PTF5. 0 PTF5 pin input buffer disabled. 1 PTF5 pin input buffer enabled.
4 SIMPTF4	<b>PTF4 Input Buffer Control</b> — This bit controls the input buffer associated with PTF4. 0 PTF4 pin input buffer disabled. 1 PTF4 pin input buffer enabled.
3 SIMPTF3	<b>PTF3 Input Buffer Control</b> — This bit controls the input buffer associated with PTF3. 0 PTF3 pin input buffer disabled. 1 PTF3 pin input buffer enabled.
2 SIMPTF2	<b>PTF2 Input Buffer Control</b> — This bit controls the input buffer associated with PTF2. 0 PTF2 pin input buffer disabled. 1 PTF2 pin input buffer enabled.
1 SIMPTF1	<b>PTF1 Input Buffer Control</b> — This bit controls the input buffer associated with PTF1. 0 PTF1 pin input buffer disabled. 1 PTF1 pin input buffer enabled.
0 SIMPTF0	<b>PTF0 Input Buffer Control</b> — This bit controls the input buffer associated with PTF0. 0 PTF0 pin input buffer disabled. 1 PTF0 pin input buffer enabled.

### 5.8.24 System PortG Input Buffer Enable Register (SIMPTG)

The SIMPTG register controls the pin input buffers on Port G.

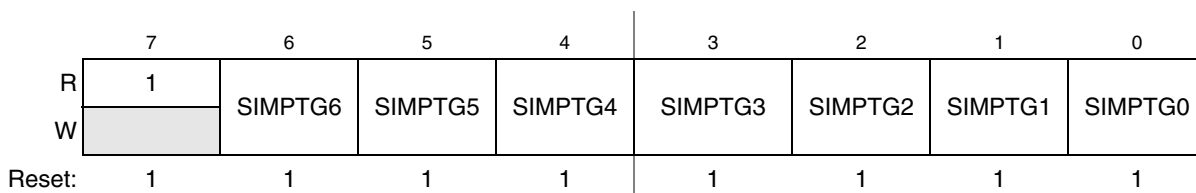


Figure 5-25. PortG input Buffer Enable Register (SIMPTG)

Table 5-28. SIMPTG Register Field Descriptions

Field	Description
7 RSVD	<b>Reserved</b> — Always read 1, write has no effect.
6 SIMPTG6	<b>PTG6 Input Buffer Control</b> — This bit controls the input buffer associated with PTG6. 0 PTG6 pin input buffer disabled. 1 PTG6 pin input buffer enabled.
5 SIMPTG5	<b>PTG5 Input Buffer Control</b> — This bit controls the input buffer associated with PTG5. 0 PTG5 pin input buffer disabled. 1 PTG5 pin input buffer enabled.
4 SIMPTG4	<b>PTG4 Input Buffer Control</b> — This bit controls the input buffer associated with PTG4. 0 PTG4 pin input buffer disabled. 1 PTG4 pin input buffer enabled.
3 SIMPTG3	<b>PTG3 Input Buffer Control</b> — This bit controls the input buffer associated with PTG3. 0 PTG3 pin input buffer disabled. 1 PTG3 pin input buffer enabled.
2 SIMPTG2	<b>PTG2 Input Buffer Control</b> — This bit controls the input buffer associated with PTG2. 0 PTG2 pin input buffer disabled. 1 PTG2 pin input buffer enabled.
1 SIMPTG1	<b>PTG1 Input Buffer Control</b> — This bit controls the input buffer associated with PTG1. 0 PTG1 pin input buffer disabled. 1 PTG1 pin input buffer enabled.
0 SIMPTG0	<b>PTG0 Input Buffer Control</b> — This bit controls the input buffer associated with PTG0. 0 PTG0 pin input buffer disabled. 1 PTG0 pin input buffer enabled.

### 5.8.25 System Porth Input Buffer Enable Register (SIMPTH)

The SIMPTH register controls the pin input buffers on Port H.

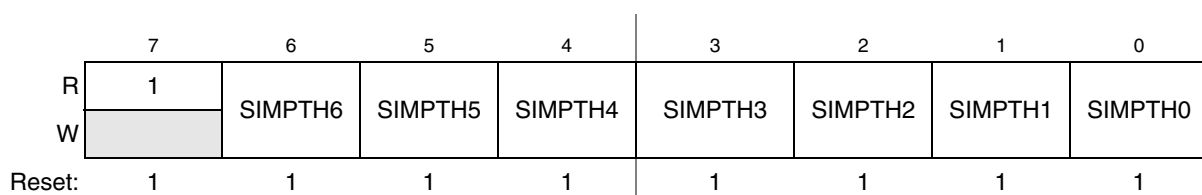


Figure 5-26. Porth Input Buffer Enable Register (SIMPTH)

Table 5-29. SIMPTH Register Field Descriptions

Field	Description
7 RSVD	<b>Reserved</b> — Always read 1, write has no effect.
6 SIMPTH6	<b>PTH6 Input Buffer Control</b> — This bit controls the input buffer associated with PTH6. 0 PTH6 pin input buffer disabled. 1 PTH6 pin input buffer enabled.
5 SIMPTH5	<b>PTH5 Input Buffer Control</b> — This bit controls the input buffer associated with PTH5. 0 PTH5 pin input buffer disabled. 1 PTH5 pin input buffer enabled.
4 SIMPTH4	<b>PTH4 Input Buffer Control</b> — This bit controls the input buffer associated with PTH4. 0 PTH4 pin input buffer disabled. 1 PTH4 pin input buffer enabled.
3 SIMPTH3	<b>PTH3 Input Buffer Control</b> — This bit controls the input buffer associated with PTH3. 0 PTH3 pin input buffer disabled. 1 PTH3 pin input buffer enabled.
2 SIMPTH2	<b>PTH2 Input Buffer Control</b> — This bit controls the input buffer associated with PTH2. 0 PTH2 pin input buffer disabled. 1 PTH2 pin input buffer enabled.
1 SIMPTH1	<b>PTH1 Input Buffer Control</b> — This bit controls the input buffer associated with PTH1. 0 PTH1 pin input buffer disabled. 1 PTH1 pin input buffer enabled.
0 SIMPTH0	<b>PTH0 Input Buffer Control</b> — This bit controls the input buffer associated with PTH0. 0 PTH0 pin input buffer disabled. 1 PTH0 pin input buffer enabled.

### 5.8.26 System PortJ Input Buffer Enable Register (SIMPTJ)

The SIMPTJ register controls the pin input buffers on Port J.

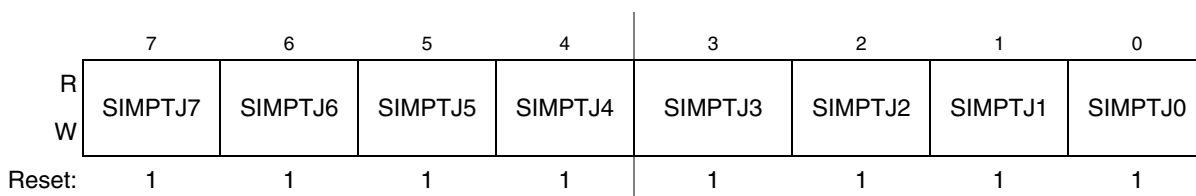


Figure 5-27. PortJ input Buffer Enable Register (SIMPTJ)

Table 5-30. SIMPTJ Register Field Descriptions

Field	Description
7 SIMPTJ7	<b>PTJ7 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ7. 0 PTJ7 pin input buffer disabled. 1 PTJ7 pin input buffer enabled.
6 SIMPTJ6	<b>PTJ6 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ6. 0 PTJ6 pin input buffer disabled. 1 PTJ6 pin input buffer enabled.
5 SIMPTJ5	<b>PTJ5 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ5. 0 PTJ5 pin input buffer disabled. 1 PTJ5 pin input buffer enabled.
4 SIMPTJ4	<b>PTJ4 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ4. 0 PTJ4 pin input buffer disabled. 1 PTJ4 pin input buffer enabled.
3 SIMPTJ3	<b>PTJ3 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ3. 0 PTJ3 pin input buffer disabled. 1 PTJ3 pin input buffer enabled.
2 SIMPTJ2	<b>PTJ2 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ2. 0 PTJ2 pin input buffer disabled. 1 PTJ2 pin input buffer enabled.
1 SIMPTJ1	<b>PTJ1 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ1. 0 PTJ1 pin input buffer disabled. 1 PTJ1 pin input buffer enabled.
0 SIMPTJ0	<b>PTJ0 Input Buffer Control</b> — This bit controls the input buffer associated with PTJ0. 0 PTJ0 pin input buffer disabled. 1 PTJ0 pin input buffer enabled.



## Chapter 6

# Parallel Input/Output Control

This section explains software controls related to parallel input/output (I/O) and pin control. The MCF51AG128 series of MCUs have up to nine parallel I/O ports which include a total of 69 I/O pins and one input-only pin. See [Chapter 2, “Pins and Connections,”](#) for more information about pin assignments and external hardware considerations of these pins.

In addition to standard I/O port functionality, Ports E and F have set, clear, and toggle functions that are integrated as part of the ColdFire core itself to improve edge resolution on those pins. See [Section 6.3, “Functional Description,”](#) and [Chapter 9, “Rapid GPIO \(RGPIO\),”](#) for additional details.

Many port pins are shared with on-chip peripherals such as timer systems, communication systems, or keyboard interrupts as shown in [Figure 6-1](#). The peripheral modules have priority over general-purpose I/O functions. When a peripheral is enabled, the I/O functions associated with the shared pins may be disabled.

After reset, the shared peripheral functions are disabled and the pins are configured as inputs ( $PTxDDn = 0$ ). The pin control functions for each pin are configured as follows: slew rate control disabled ( $PTxSREn = 0$ ), high drive strength selected ( $PTxDSn = 0$ ), and internal pulls disabled ( $PTxPUEn = 0$ ).

### NOTE

Not all general-purpose I/O pins are available on all packages. To avoid extra current drain from floating input pins, the user’s reset initialization routine in the application program must either enable on-chip pullup devices or change the direction of unconnected pins to outputs so the pins do not float.

## 6.1 Overview

The 2nd generation parallel input/output, EGPIO, is a purely digital module (synthesizable for all technologies) present at both sides of the multiplex control module as [Figure 6-1](#).

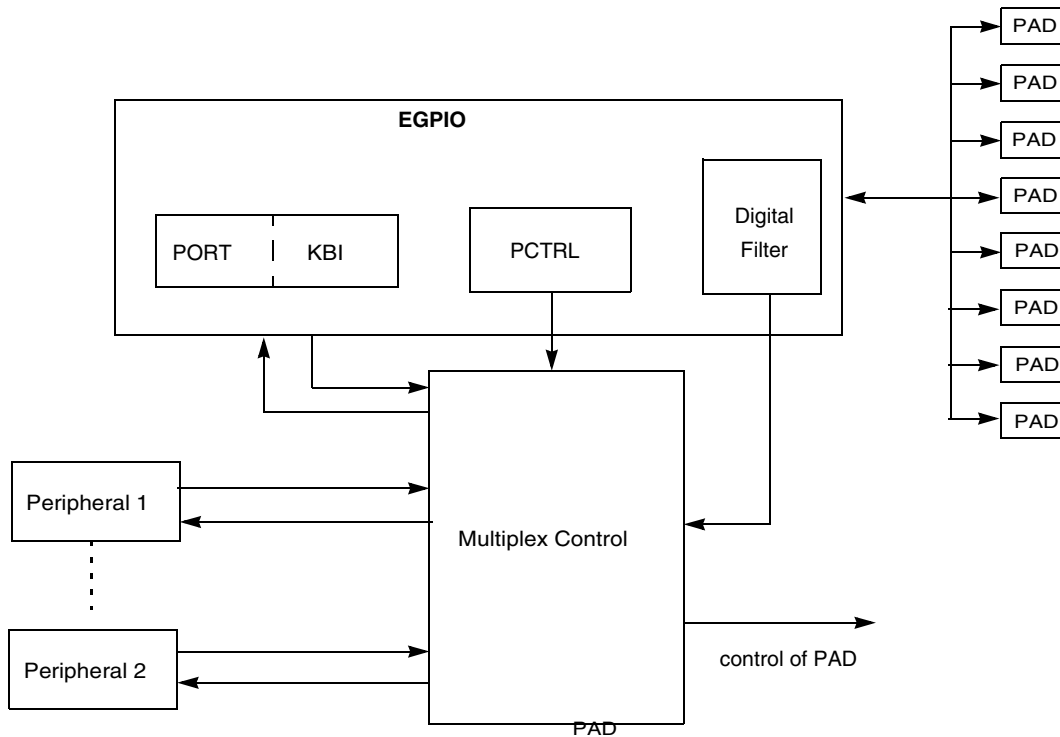


Figure 6-1. Top-level Interface of EGPIO

### 6.1.1 Features

The EGPIO includes four main functions with the below features:

- Port data logic
  - Pin input data mapped to port data register (PTxD) and output data controlled through same
  - Independent directional control per pin through data direction register (PTxDD)
  - Independent pin value register (PTxPV) to read logic level on digital pin
- Independent port control
  - Independent internal input pulling resistor enable per pin through pulling enable register (PTxPUE)
  - Independent internal input pullup/pulldown select per pin through pullup/pulldown select register (PTxPUS)
  - Independent output slew rate control per pin through slew rate enable register (PTxSRE).
  - Independent output drive strength control per pin through drive strength control register (PTxDS).
  - Independent input passive filter per pin through PTxPFE

- Pin interrupt
  - Pin interrupt for each pin with individual flag and individual enable bit
  - Each pin is programmable as falling edge (or rising edge) only, or both falling edge and low level (or both rising edge and high level) interrupt sensitivity
  - One software-enabled interrupt to CPU
  - One asynchronous interrupt to wake the CPU from low-power mode
  - DMA interface to support transfer by DMA
- Digital filters
  - Digital filter for each pin if configured as input with individual enable bit
  - Programmable of the digital filtering frequency through PTxDFC

## 6.1.2 Modes of Operation

The section defines the EGPIO operation in wait, stop, and background debug modes.

### 6.1.2.1 Operation in Wait Mode

All EGPIO functions continue to operate in wait mode if enabled before executing the WAIT instruction. If pin DMA request is not enabled ( $PTxDMAEN = 0$ ), an enabled interrupt pin ( $PTxIPEn = 1$ ) can be used to bring the MCU out of wait mode if the interrupt of EGPIO is enabled ( $PTxIE = 1$ ).

### 6.1.2.2 Operation in Stop Mode

The pin interrupt function operates asynchronously in stop mode if enabled before executing the STOP instruction. Therefore, an enabled interrupt pin ( $PTxIPEn = 1$ ) can be used to bring the MCU out of stop mode if the interrupt of EGPIO is enabled ( $PTxIE = 1$ ) and digital filter on this pin is disabled (bypass mode) or digital filter continues to operate on LPO clock.

The digital filter continues to operate if enabled and LPO clock is selected for digital filtering before entering.

### 6.1.2.3 Operation in Active Background Mode

When the MCU is in active background mode, all EGPIO functions continue to operate normally.

## 6.2 Memory Map and Registers

### 6.2.1 Overview

This section provides a detailed description of all EGPIO registers.

### 6.2.2 Module Memory Map

[Table 6-1](#) shows the registers contained in the EGPIO module.

**Table 6-1. Module Memory Map**

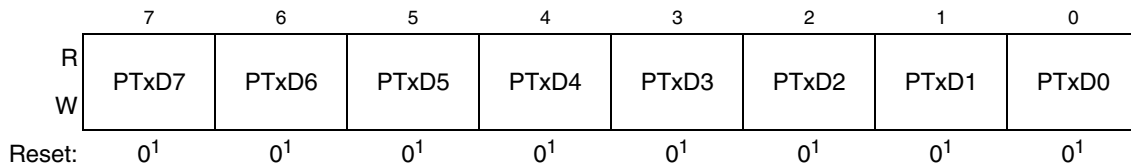
<b>Address</b>	<b>Use</b>	<b>Access</b>
BANK1_Base <sup>1</sup> + 0x00	PORTx Data Register (PTxD)	read/write
BANK1_Base <sup>1</sup> + 0x01	PORTx Data Direction Register (PTxDD)	read/write
BANK1_Base <sup>1</sup> + 0x02	PORTx Pin Value (PTxPV)	read-only
BANK2_BASE <sup>1</sup> + 0x00	PORTx Pulling Enable Register (PTxPUE)	read/write
BANK2_BASE <sup>1</sup> + 0x01	PORTx Pullup/Pulldown Select Register (PTxPUS)	read/write
BANK2_BASE <sup>1</sup> + 0x02	PORTx Drive Strength Enable Register (PTxDS)	read/write
BANK2_BASE <sup>1</sup> + 0x03	PORTx Slew Rate Enable Register (PTxSRE)	read/write
BANK2_BASE <sup>1</sup> + 0x04	PORTx Passive Filter Enable Register (PTxPFE)	read/write
BANK2_BASE <sup>1</sup> + 0x05	PORTx Interrupt Control Register (PTxIC)	read/write
BANK2_BASE <sup>1</sup> + 0x06	PORTx Interrupt Pin Enable Register (PTxIPE)	read/write
BANK2_BASE <sup>1</sup> + 0x07	PORTx Interrupt Flag Register (PTxIF)	read/write
BANK2_BASE <sup>1</sup> + 0x08	PORTx Interrupt Edge Select Register (PTxIES)	read/write
BANK2_BASE <sup>1</sup> + 0x09	PORTx Digiter Filter Enable (PTxDFE)	read/write
BANK2_BASE <sup>1</sup> + 0x0A	PORTx Digital Filter Control Register (PTxDFC)	read/write

<sup>1</sup> The registers of EGPIO are composed of two banks to support architecture of S08. For S08s, registers of EGPIO is located in both direct page (bank1) and high page (bank2). Access of the registers in direct page is much faster. For CFV1, there is no pages called direct page or high page and access of any page has the same speed, so all registers of EGPIO can be put continuously in one page.

### 6.2.3 Register Descriptions

This section describes the EGPIO registers and their operation.

### 6.2.3.1 Port Data Registers (PTxD)



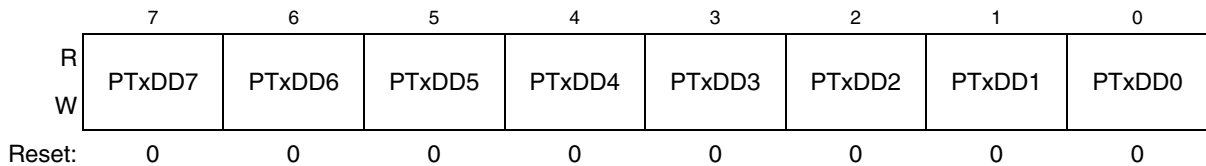
**Figure 6-2. PORTx Data Register (PTxD)**

- <sup>1</sup> The reset values of internal registers for PTxD are zeros. But read of PTxD after reset, returns actual logic level on external pins.

**Table 6-2. PTxD Register Field Descriptions**

Field	Description
7:0 PTxD[7:0]	<b>PORTx Data Register Bits</b> — Writes are latched into all bits of this register. When port data direction bits for port pins are set (PTxDD bit = 1), reads returns last value written to this register. When port pin is controlled by EGPIO with pin interrupt function disabled and associated port data direction bit is set (PTxDD bit =1). The logic level is driven out to the corresponding MCU pin when port data direction bits for port pins are cleared (PTxDD bit =0). For pins that are configured as digital pins, reads return logic level on the pin. For port pins that are controlled by analog functions, reads return zeros (offvalue).

### 6.2.3.2 Data Direction Registers (PTxDD)



**Figure 6-3. PORTx Data Direction Register (PTxDD)**

**Table 6-3. PTxDD Register Field Descriptions**

Field	Description
7:0 PTxDD[7:0]	<b>PORTx Data Direction Register Bits</b> — Each bit of port data direction register controls whether associated port pin is input or output when pin interrupt is disabled and no other module controls the pin. If the DD bit for a port pin is equal to logic one, and Port Data Logic has control of the pin, the port pin is defined as output and the logic value of internal register for PTxD is driven out to the corresponding MCU pin. 1 Port pin defined as output. 0 Port pin defined only as input.

### 6.2.3.3 PORTx Pin Value Register (PTxPV)

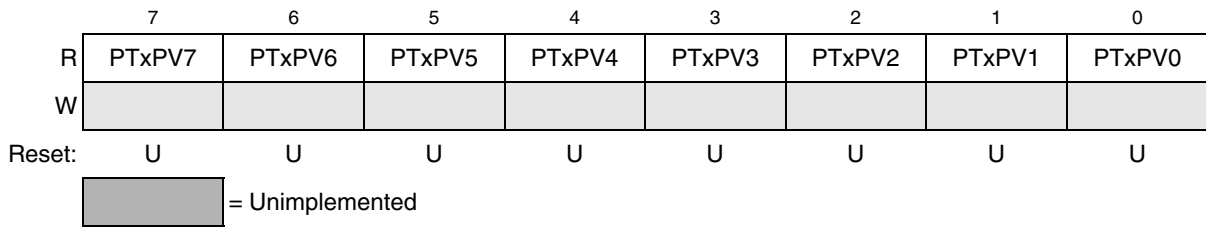


Figure 6-4. PORTx Pin Value Register (PTxPV)

<sup>1</sup> Read of PTxPV after reset returns actual logic level on external pins since reset configures all port pins as high-impedance inputs with pullup/pulldown disabled.

Table 6-4. PTxPV Register Field Descriptions

Field	Description
7:0 PTxPV[7:0]	<b>PORTx Pin Value Bits</b> — Each bit of port pin value register is mapped to one MCU pin. For pins that are configured as digital pins, this register always reflects the digital level on the actual PAD. For pins that are controlled by analog functions, reads return zeros (off-value).

### 6.2.3.4 PORTx Pulling Enable Register (PTxPUE)

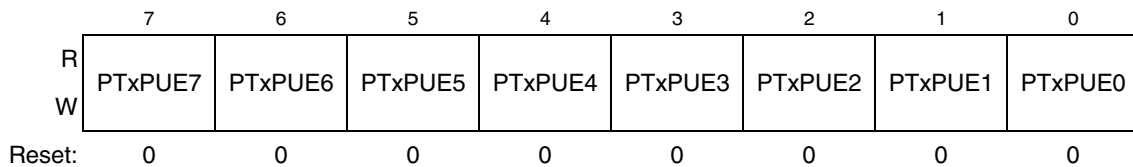


Figure 6-5. PORTx Pulling Enable Register (PTxPUE)

Table 6-5. PTxPUE Register Field Descriptions

Field	Description
7:0 PTxPUE[7:0]	<b>PORTx Internal Pulling Enable Bits<sup>1</sup></b> — Each pin has a pullup and pulldown resistors associated with it. For port pins that are not configured as inputs, these bits have no effect and the internal pull resistors are disabled. 1 Pulling resistor is enabled 0 Pulling resistor is disabled

<sup>1</sup> if there is no special note or description for the module which controls the pin whether a pulling resistor is enabled for the module, is decided by associating PTxPUE bit.

### 6.2.3.5 PORTx Pullup/Pulldown Select Register (PTxPUS)

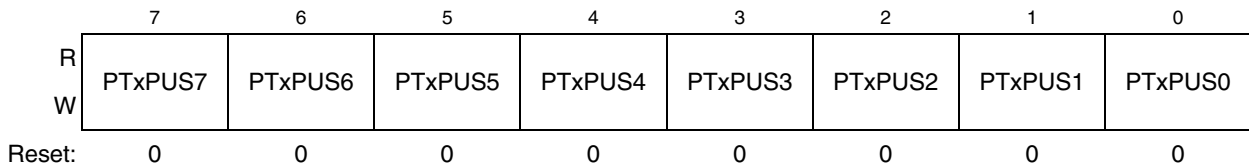


Figure 6-6. PORTx Pullup/Pulldown Register (PTxPUS)

Table 6-6. PTxPUS Register Field Descriptions

Field	Description
7:0 PTxPUS[7:0]	<b>PORTx Pullup/Pulldown Select Bits</b> <sup>1</sup> — Each bit selects pullup or pulldown resistor enabled by the corresponding PTxPUE bit. For port pins that are not configured as inputs, these bits have no effect. 1 Pullup resistor is selected 0 Pulldown resistor is selected

<sup>1</sup> if there is no special note or description for the module which controls the pin, the selection of pullup/pulldown of the module is decided by associated PTxPUS bit.

### 6.2.3.6 PORTx Drive Strength Control Register (PTxDS)

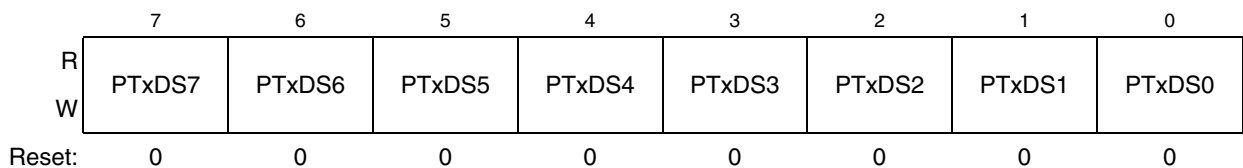


Figure 6-7. PTx Drive Strength Control Register (PTxDS)

Table 6-7. PTxDS Register Field Descriptions

Field	Description
7:0 PTxDS[7:0]	<b>PORTx Output Drive Strength Control Bits</b> — Each of these control bits selects between low and high output drive for the associated PTC pin. For port pins that are configured as inputs, these bits have no effect. 1 High output drive strength selected. 0 Low output drive strength selected.

### 6.2.3.7 PORTx Slew Rate Enable Register (PTxSRE)

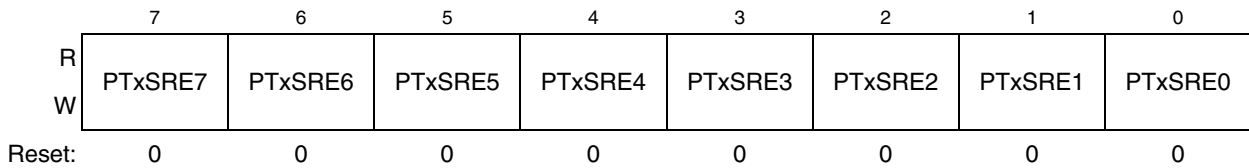


Figure 6-8. PTx Slew Rate Enable Register (PTxSRE)

Table 6-8. PTxSRE Register Field Descriptions

Field	Description
7:0 PTxSRE[7:0]	<b>PORTx Output Slow Rate Enable Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated Port pin. For port pins that are not configured as outputs, these bits have no effect. 1 Slew rate control enabled. 0 Slew rate control disabled.

### 6.2.3.8 PORTx Passive Filter Enable Register (PTxPFE)

PTxPFE enables you to control the input low-pass filter on the pad. These may be enabled by setting the appropriate bit in the input filter enable register (PTxPFE[n]) corresponding to a given pin. When set high, a low pass filter (10 MHz to 30 MHz bandwidth) is enabled in the logic input path. When set low, the filter is bypassed.

The filter is enabled during and after reset by setting the associated PTxPFE bit. The filter is disabled through software control by clearing the associated PTxPFE bit.

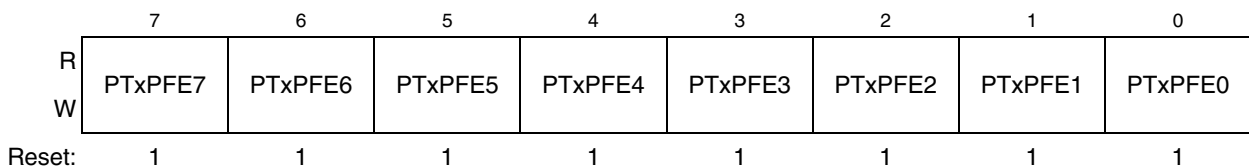


Figure 6-9. PTx Passive Filter Enable Register (PTxPFE)

Table 6-9. PTxPFE Register Field Descriptions

Field	Description
7:0 PTxPFE[7:0]	<b>PORTx Passive Input Filter Enable Bits</b> — Input low-pass filter enables control bits for PTx pins. For port pins not configured as inputs, these bits have no effect. 1 Input low-pass filter on pad enabled. 0 Input low-pass filter on pad disabled.

### 6.2.3.9 PORTx Interrupt Control Register (PTxIC)

PTxIC contains control bits used for pin interrupt function.



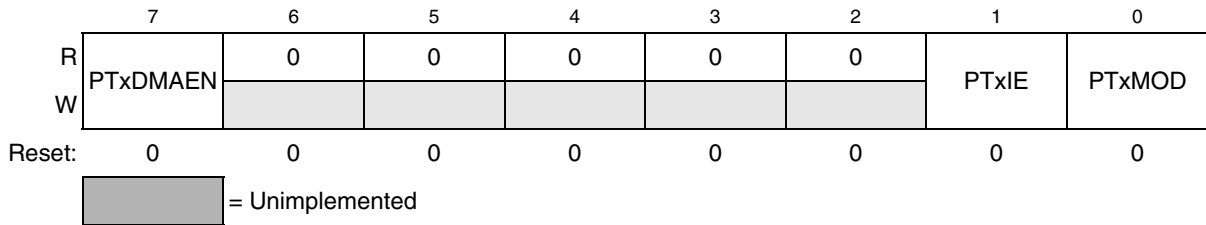


Figure 6-10. PORTx Interrupt Control Register(PTxIC)

Table 6-10. PTxIC Register Field Descriptions

Field	Description
7 PTxDMAEN	<b>DMA Enable bit</b> — This bit determines whether the pin DMA request is enabled. If enabled, pin DMA request is asserted when interrupt flag of any pin is set (at least one bit of PTxIF is set). Meanwhile, synchronous interrupt from PTxIF is disabled if this bit is set. 0 Pin DMA request is disabled and synchronous interrupt from PTxIF is allowed. 1 Pin DMA request is enabled and synchronous interrupt from PTxIF is disabled
1 PTxIE	<b>Interrupt Enable</b> — This bit determines whether a pin interrupt is requested to CPU. 0 Pin interrupt request not enabled. 1 Pin interrupt request enabled.
0 PTxMOD	<b>Detection Mode for Pin Interrupt</b> — MOD (along with the PTxEDG bits) controls the detection mode of the Pin Interrupt for pins. 0 Pin interrupt detects edges only. 1 Pin interrupt detects both edges and levels.

### 6.2.3.10 PORTx Interrupt Pin Enable Register (PTxIPE)

PTxIPE contains the enable control bits of pin interrupt.

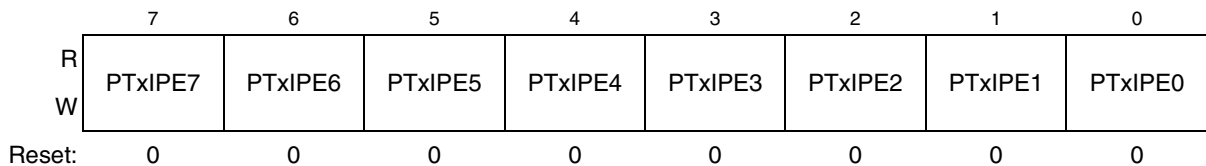


Figure 6-11. PORTx Interrupt Pin Enable Register(PTxIPE)

Table 6-11. PTxIPE Register Field Descriptions

Field	Description
7:0 PTxIPE[7:0]	<b>Interrupt Pin Enables</b> — Each of the PTxIPE bits enables the corresponding pin for pin interrupt. 0 Pin not enabled for pin interrupt. 1 Pin enabled for pin interrupt.

### 6.2.3.11 PORTx Interrupt Flag Register (PTxIF)

PTxIF contains the interrupt flag bits of pin interrupt.

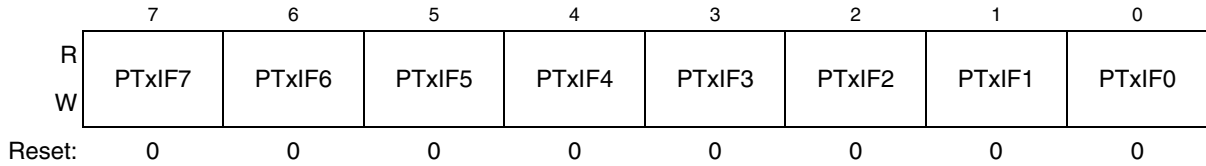


Figure 6-12. PORTx Interrupt Flag Register (PTxIF)

Table 6-12. PTxIF Register Field Descriptions

Field	Description
7:0 PTxIF[7:0]	<b>Interrupt Flags</b> — indicate whether condition of pin interrupt is detected on each input pin if enabled by associated PTxIPE bit. Writing 1 to one bit clears associated PTxIF bit if set. 0 No condition of pin interrupt detected. 1 Condition of pin interrupt detected.

### 6.2.3.12 PORTx Interrupt Edge Select Register (PTxIES)

PTxIES contains the edge select control bits.

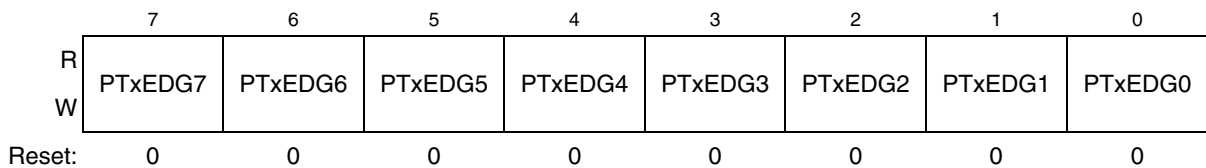


Figure 6-13. PORTx Interrupt Edge Select Register (PTxIES)

Table 6-13. PTxIES Register Field Descriptions

Field	Description
7:0 PTxEDG[7:0]	<b>Edge Selects of Pin Interrupt</b> — Each of the PTxEDG bits selects the falling edge/low level or rising edge/high level function of the corresponding pin. 0 Falling edge/low level. 1 Rising edge/high level.

### 6.2.3.13 PORTx Digital Filter Enable Register (PTxDFE)

PTxDFE contains the enable control bits for digital filters.

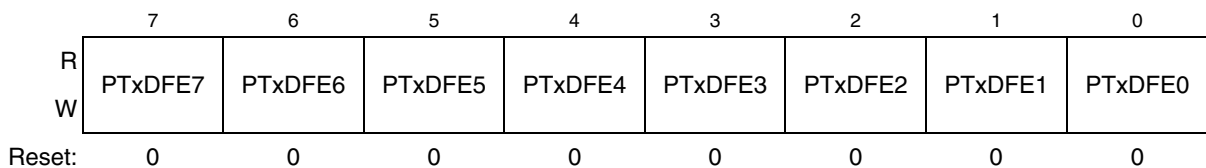


Figure 6-14. PORTx Digital Filter Enable Register (PTxDFE)

Table 6-14. PTxDFE Register Field Descriptions

Field	Description
7:0 PTxDFE[7:0]	<p><b>Digital Filter Enables</b> — The PTxDFE register enables the digital filter on the pin when configured as an input. If the pin is not configured as an input, the digital filter circuit is not used. The digital filter when enabled is included in the signal path to EGPIO and any module who gets control of the pin and configure it as an input.</p> <p>0 Digital filter disabled (bypass mode). 1 Digital filter enabled.</p>

### 6.2.3.14 PORTx Digital Filter Control Register (PTxDFC)

PTxDFC contains the control bits to select clock and filter factor for all digital filters of port.

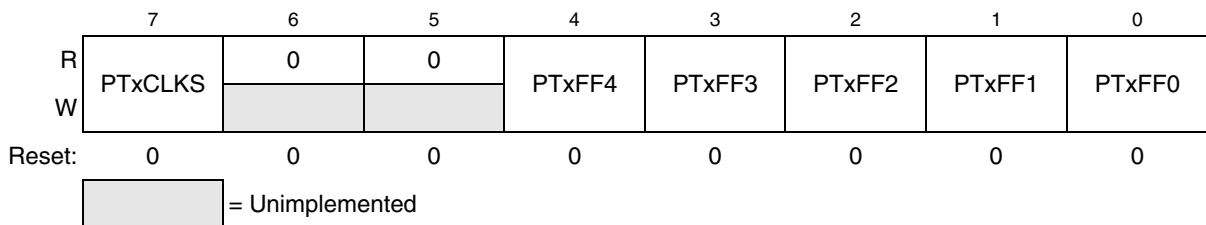


Figure 6-15. PORTx Digital Filter Control Register (PTxDFC)

Table 6-15. PTxDFC Register Field Descriptions

Field	Description
7 PTxCLKS	<p><b>CLok Select Bit</b> — Selects counting clock for digital filters.</p> <p>0 Digital filters count on bus clock. 1 Digital filters count on LPO clock.</p>
4:0 PTxFF[4:0]	<p><b>Filter Factor Bits</b> — Contains the programmable controls for the width of glitch (in terms of clock cycles) the filter should absorb; in other words, the filter does not let glitches less than or equal to this width setting pass.</p>

Table 6-16. PTxFF Descriptions

PTxFF[4:0] Field					Width of Clock Cycles
0	0	0	0	0	1
0	0	0	0	1	2
0	0	0	1	0	3
0	0	0	1	1	4
0	0	1	0	0	5
0	0	1	0	1	6
0	0	1	1	0	7
0	0	1	1	1	8
.....					

**Table 6-16. PTxFF Descriptions (continued)**

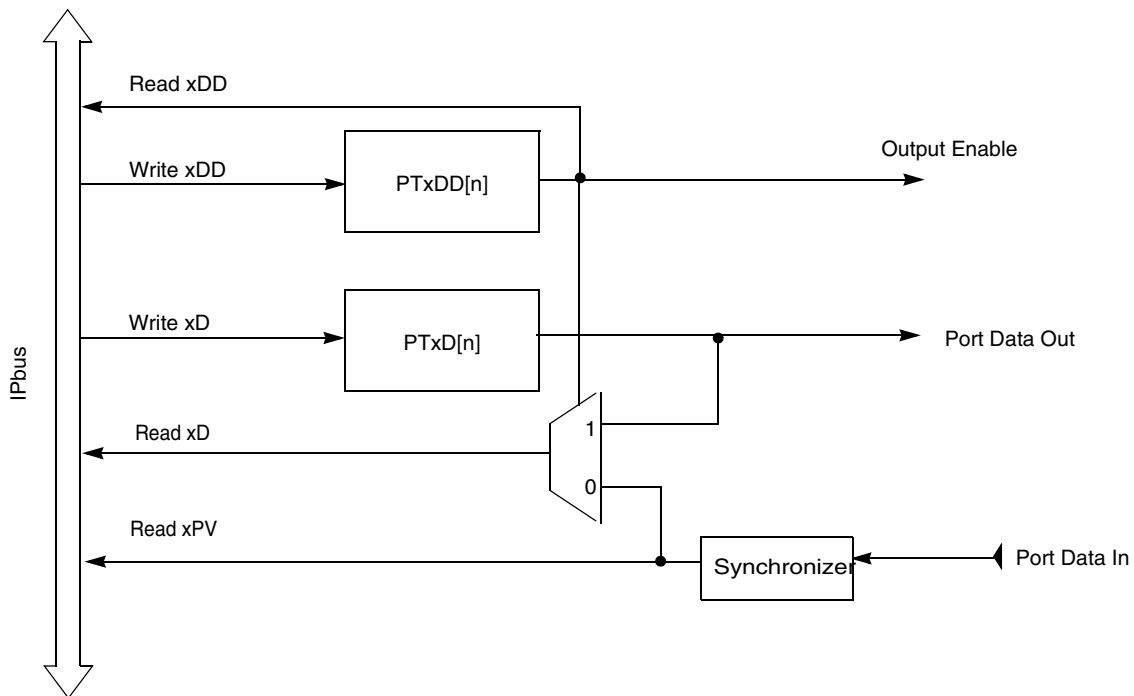
PTxFF[4:0] Field					Width of Clock Cycles
1	1	0	0	0	25
1	1	0	0	1	26
1	1	0	1	0	27
1	1	0	1	1	28
1	1	1	0	0	29
1	1	1	0	1	30
1	1	1	1	0	31
1	1	1	1	1	32

### 6.3 Functional Description

This section provides full descriptions for all EGPIO functions.

#### 6.3.1 Port Data Logic

Figure 6-16 shows port data logic diagram.



**Figure 6-16. Diagram of Port Data Logic Function**

Port data logic is for general purpose input and output function which is the default function for port pin when no shared on-chip module is enabled and the pin interrupt function is disabled.

Each port has a data register, a data direction register, and a pin value register. For S08s, these registers can be mapped to a direct page for quick access. Besides the EGPIO pin interrupt function, an I/O port bit may or may not share an I/O pin with another on-chip module. Consult the chip documentation for information on pin assignments.

### 6.3.1.1 Operation When EGPIO Controls Pin

If pin interrupt for this pin is not enabled (PTxIPE bit = 0), the direction of the pin depends on associated port DD bit. When port DD bit is one, data written to the port data register is driven out to the corresponding pad.

If pin interrupt for this pin is enabled (PTxIPE bit = 1), the direction of the pin is configured as input and the associated port DD bit has no effect. In this case, data written to port data register is not driven out to the corresponding pad. When associated port DD bit is cleared, a read of port data register returns the logic level of associated pin. When port DD bit is set, a read of port data register returns the last value written to associated bit of port data register.

When port pin is controlled by EGPIO, the port is configured for digital function so the pin value register read always returns actual logic level on pads.

### 6.3.1.2 Operation When Another On-Chip Module Controls Pin

When a pin is controlled by another on-chip module, the port DD bit associated with the pin does not affect the direction of the pin, either input or output. However, when port DD bit is set, the port data register read still returns the last value written to the port data register associated bit. When port DD bit is cleared and port pin is configured for digital function, read of port data register still returns actual logic level on pads.

### 6.3.1.3 Pin Value Register

For port pins that are configured as digital pins, no matter as input or output, pin value register read always reflects logic level on pads. For port pins that are controlled by analog functions, pin value register read returns zeros (off-value).

### 6.3.2 Port Control

Figure 6-17 shows PCTRL function diagram.

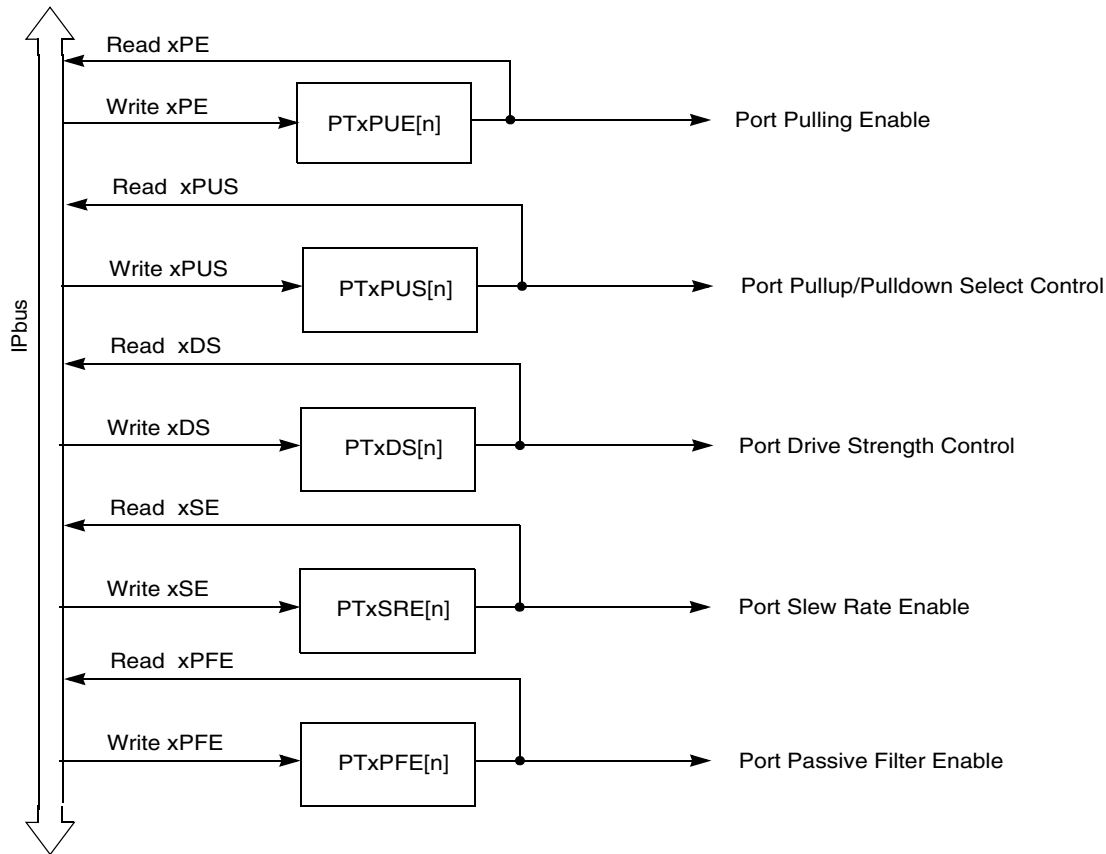


Figure 6-17. Diagram of PCTRL function

The pin control registers are CPU-accessible and operate as described in [Section 6.2, “Memory Map and Registers.”](#)

### 6.3.3 Pin Interrupt

The pin interrupt function provides up to eight independently enabled external interrupt sources. The block diagram for the pin interrupt function is shown [Figure 6-18](#).

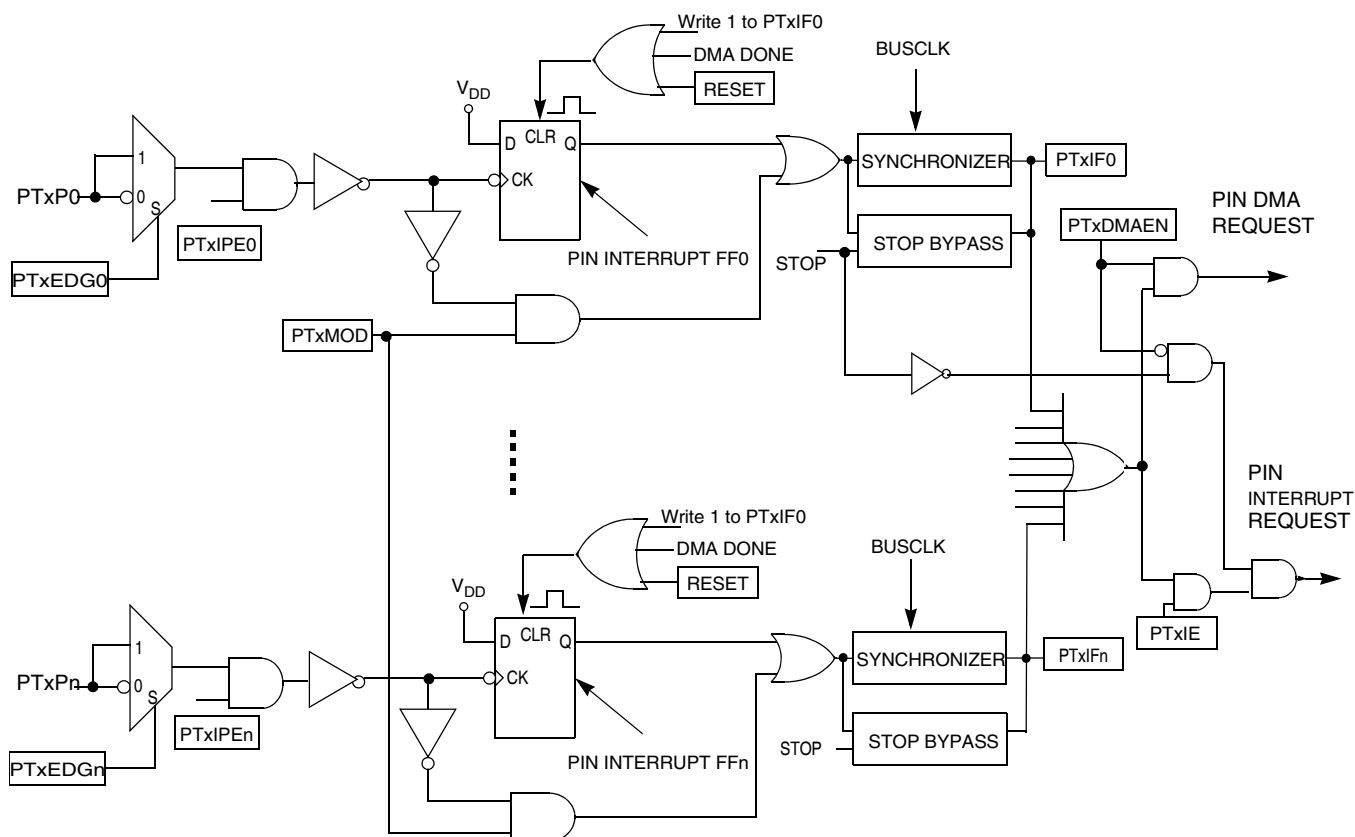


Figure 6-18. Diagram of Pin Interrupt function

The pin interrupt function is included in EGPIO to supersede existing KBI module. The main difference is that the pin interrupt function is still able to operate only if a port pin is configured as a digital pin even when EGPIO does not get control of the pin that means pin interrupt if enabled can operate with an other on-chip module at the same time.

When a port pin is controlled by EGPIO, if the pin interrupt is enabled for this pin (PTxIPE bit =1), the pin is configured as input. When a port pin is controlled by digital function of other on-chip module, the pin interrupt if enabled (PTxIPE bit =1) still operates because the EGPIO always receives the actual logic level on external Pads when the pin is configured for digital use. But special care must be taken when pin interrupt operates at the same time with other on-chip module and it is strongly recommended to disable pin interrupt function when other on-chip module controls the pin for output function. When a port pin is configured for analog function, input for pin interrupt function is zero and it is recommended to disable the pin interrupt function as well.

### 6.3.3.1 Edge Only Sensitivity

A valid edge on an enabled pin interrupt sets an associated PTxIF bit. If PTxDMAEN in PTxIC is set, a DMA request is asserted. If PTxDMAEN is not set and PTxIE is set, an interrupt request is presented to the CPU. Clearing of a PTxIF bit is accomplished by writing 1 to the same bit or when the DMA DONE signal for Pin DMA request is asserted. A valid edge or level on an enabled pin of pin interrupt sets an

associated PTxIF bit. If PTxDMAEN in PTxIC is set, a DMA request is asserted. If PTxDMAEN is not set and PTxIE is set, an interrupt request is presented to the CPU. Clearing of a PTxIF bit is accomplished by writing 1 to the same bit or when the DMA DONE signal for Pin DMA request is asserted provided associated enabled input of pin interrupt is at its deasserted level. A PTxIF bit remains set if associated enabled pin of pin interrupt is asserted while attempting to clear it. Since the DMA process is automatic and it is very hard to deassert all enabled inputs before the DMA DONE signal is done, it is strongly recommended not to enable pin DMA request when both edge and level are selected as valid conditions for pin interrupt (PTxMOD = 1).

### 6.3.3.2 Edge and Level Sensitivity

A valid edge or level on an enabled pin interrupt sets an associated PTxIF bit. If PTxDMAEN in PTxIC is set, a DMA request is asserted. If PTxDMAEN is not set and PTxIE is set, an interrupt request is presented to the CPU. Clearing of a PTxIF bit is accomplished by writing 1 to the same bit or when the DMA DONE signal for Pin DMA request is asserted, provided associated enabled input of pin interrupt is at its deasserted level. A PTxIF bit remains set if associated enabled pin of pin interrupt is asserted while attempting to clear it. Since the DMA process is automatic and it is very hard to deassert all enabled inputs before the DMA DONE signal is done; it is strongly recommended not to enable pin DMA request when both edge and level are selected as valid conditions for pin interrupt (PTxMOD = 1).

### 6.3.3.3 Control of Pullup/Pulldown Resistors

The enabled pin interrupt can be configured to use an internal pullup/pulldown resistor with the associated I/O port pulling enable register. When an internal pulling resistor is enabled (pin configured as input and PTxPUE bit = 1), there are three situations as below:

1. If the EGPIO gets control of the pin and pin is enabled for pin interrupt, an associated PTxEDG bit in PTxIES selects whether the resistor is a pullup (PTxEDG bit = 0) or a pulldown (PTxEDG bit = 1).
2. If the EGPIO does not get control of the pin or pin is not enabled for pin interrupt, the PTxEDG bits have no effect.
3. If the EGPIO does not get control of the pin and pin is still enabled for pin interrupt, the PTxEDG bits have no effect on selection of pullup/pulldown resistors. However, care must be taken and associated PTxEDG bit must be set according to actual selection of internal pullup (PTxEDG bit must be cleared) or pulldown (PTxEDG bit must be set) on Pad by the module which gets control of the pin, otherwise false conditions when there is no drive on Pad outside may be detected as valid conditions for pin interrupt.

### 6.3.3.4 Asynchronous Interrupt in Stop Mode

When the MCU enters stop mode, the synchronous edge-detection logic is bypassed (because clocks are stopped). In stop mode, enabled inputs of pin interrupt act as asynchronous level-sensitive inputs so they can wake the MCU from stop mode. From the diagram of pin interrupt above, in stop mode, the pin interrupt requests are not blocked by PTxDMAEN=1.



### 6.3.3.5 Pin Interrupt Initialization

When an interrupt pin is first enabled or reconfigured, it is possible to get a false interrupt flag. To prevent a false interrupt request during pin interrupt initialization, you must perform the following steps:

1. Mask port interrupts by clearing PTxIE in PTxIC.
2. Enable the polarity for pin interrupt by setting the appropriate PTxEDGn bits in PTxIES. If pin interrupt function operates at the same time with other on-chip module which gets control of the pin, the associated PTxEDG bit must be set according to actual selection of internal pullup (PTxEDG bit must be cleared) or pulldown (PTxEDG bit must be set) on pad by the module which gets control of the pin.
3. If using internal pullup/pulldown device, configure the associated pulling enable bits in PTxPUE.
4. Enable the pins for pin interrupt by setting the appropriate bits in PTxIPE.
5. Execute three NOP instructions before the next step to avoid timing conflict between setting PTxIPE and clearing flag.
6. Write 0xFF to PTxIF to clear any false interrupt flags.
7. Enable DMA request or interrupt request by configuration of PTxIC.

### 6.3.4 Digital Filters

The passive input low-pass filter only can filter >10 MHz signals. EGPIO provides programmable digital filters to filter signals much less than 10 MHz for low-speed applications.

The digital filters absorb glitches on digital pins. For the port pin configured as a digital pin, the digital filter is enabled for the pin if associated PTxDFE bit is set. For port pin that is not configured for analog function, the digital filter for the pin is disabled and associated PTxDFE bit has no effect.

The width of the glitch to absorb can be specified in terms of number of filter clock cycles. The bus clock or LPO clock can be selected as a filter clock that is configured by the CLKS bit in PTxDFC. The width of the glitch to absorb depends on the FF (Filter Factor) bits in PTxDFC. Effectively, any down-up-down or up-down-up transition on the digital input line that occurs within the number of clock cycles programmed by filter factor is ignored by on-chip modules. See [Section 6.2.3.14, “PORTx Digital Filter Control Register \(PTxDFC\)”](#) for details.

Since the configuration of PTxDFC is for all digital filters of port, change of PTxDFC affects all digital filters of port if enabled. Configuration of PTxDFC must be done when no digital filter is active (PTxDFE = 0x00).

The LPO clock can operate in stop mode, if the digital filter works in stop mode, the CLKS bit must be set before entering stop mode.

#### 6.3.4.1 Initialization of Digital Filters

When a digital filter for a port pin is enabled from disabled or pin control is changed from one module to another module while the digital filter is active, it is possible for some enabled modules to get a false input.

To prevent a false input to on-chip modules and related errors during initialization of digital filters, you must do the following to enable the digital filter for input of target module:

1. Write 1'b0 to associate the PTxDFE bit to disable the digital filter for the pin if active
2. Write 1'b0 to associate the PTxIPE bit to disable the pin interrupt function if enabled
3. Disable other on-chip modules that have the highest priority of pin control than the target module
4. Write 1'b1 to associate pin enable bit of the target module if not enabled.
5. Follow the flow for initialization of pin interrupt if you still want the pin interrupt function working with the target module.
6. Write 1'b1 to associate DFE bit to enable the digital filter for the pin.

The above flow assumes that selection of filter clock and filter factor do not change and the target module is other on-chip module than EGPIO.

If you enable pin interrupt function of EGPIO on a port pin when the digital filter for the pin has been active for the input of another on-chip module, you must follow the initialization of pin interrupt.

If you enable only a digital filter for EGPIO input, you must perform the following steps:

1. Write 1'b0 to associate the PTxDFE bit to disable the digital filter for the pin if active
2. Disable other on-chip modules that have the highest priority of pin control than EGPIO
3. Follow initialization of pin interrupt if you enable pin interrupt function on this port pin.
4. Write 1'b1 to associate DFE bit to enable the digital filter for the pin.

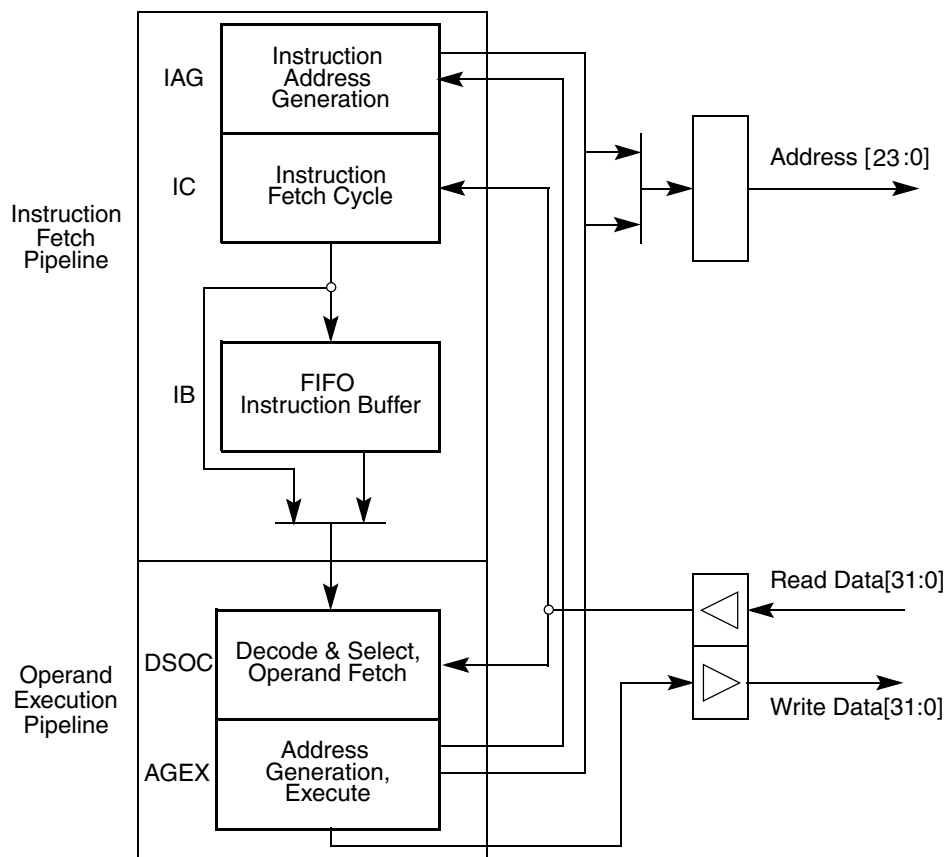
# Chapter 7 ColdFire Core

## 7.1 Introduction

This section describes the organization of the Version 1 (V1) ColdFire<sup>®</sup> processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA\_C definition in the *ColdFire Family Programmer's Reference Manual*.

### 7.1.1 Overview

As with all ColdFire cores, the V1 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.



**Figure 7-1. V1 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 V1 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. The instruction buffer on the V1 core contains three longwords of storage.

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 V1 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

## 7.2 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 7-1](#) 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)

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)
- 32-bit CPU configuration register (CPUCR)

**Table 7-1. ColdFire Core Programming Model**

BDM Command <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written with MOVEC <sup>2</sup>	Section/Page
<b>Supervisor/User Access Registers</b>						
Load: 0x60 Store: 0x40	Data Register 0 (D0)	32	R/W	See <a href="#">7.3.3.14/7-19</a>	No	<a href="#">7.2.1/7-3</a>
Load: 0x61 Store: 0x41	Data Register 1 (D1)	32	R/W	See <a href="#">7.3.3.14/7-19</a>	No	<a href="#">7.2.1/7-3</a>
Load: 0x6–7 Store: 0x4–7	Data Register –7 (D–D7)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.1/7-3</a>
Load: 0x68–E Store: 0x48–E	Address Register 0–6 (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.2/7-4</a>
Load: 0x6F Store: 0x4F	User A7 Stack Pointer (A7)	32	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.3/7-4</a>
Load: 0xEE Store: 0xCE	Condition Code Register (CCR)	8	R/W	POR: Undefined Else: Unaffected	No	<a href="#">7.2.4/7-5</a>
Load: 0xEF Store: 0xCF	Program Counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No	<a href="#">7.2.5/7-6</a>
<b>Supervisor Access Only Registers</b>						
Load: 0xE0 Store: 0xC0	Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x(00)00_0000	No	<a href="#">7.2.3/7-4</a>
Load: 0xE1 Store: 0xC1	Vector Base Register (VBR)	32	R/W	See section	Yes; Rc = 0x801	<a href="#">7.2.6/7-6</a>
Load: 0xE2 Store: 0xC2	CPU Configuration Register (CPUCR)	32	W	See section	Yes; Rc = 0x802	<a href="#">7.2.7/7-7</a>
Load: 0xEE Store: 0xCE	Status Register (SR)	16	R/W	0x27--	No	<a href="#">7.2.8/7-8</a>

<sup>1</sup> The values listed in this column represent the 8-bit BDM command code used when accessing the core registers via the 1-pin BDM port. For more information see [Chapter 26, “Version 1 ColdFire Debug \(CF1\\_DEBUG\)”](#). (These BDM commands are not similar to other ColdFire processors.)

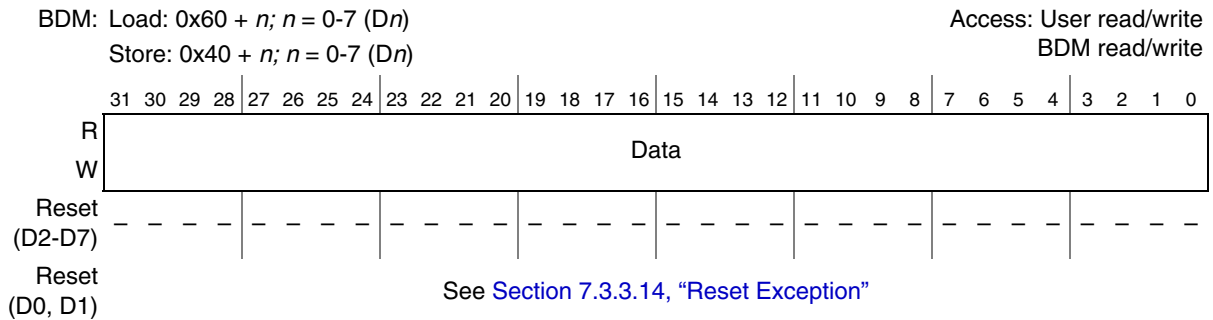
<sup>2</sup> If the given register is written using the MOVEC instruction, the 12-bit control register address (Rc) is also specified.

## 7.2.1 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**

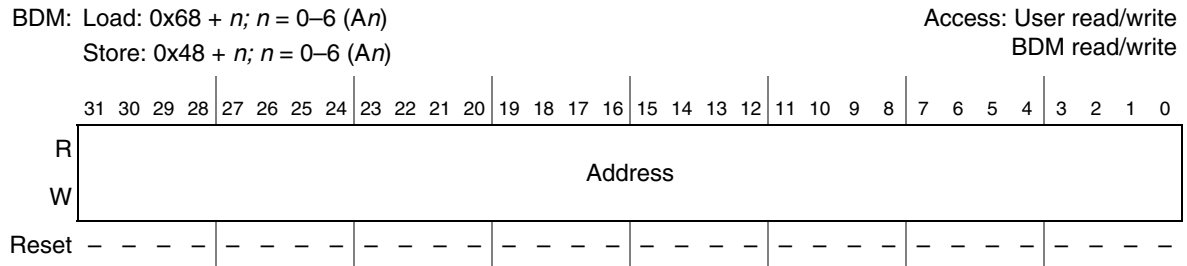
Registers D0 and D1 contain hardware configuration details after reset. See [Section 7.3.3.14, “Reset Exception”](#) for more details.



**Figure 7-2. Data Registers (D0–D7)**

## 7.2.2 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.



**Figure 7-3. Address Registers (A0–A6)**

## 7.2.3 Supervisor/User Stack Pointers (A7 and OTHER\_A7)

This 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 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 `0x(00)00_0000`.

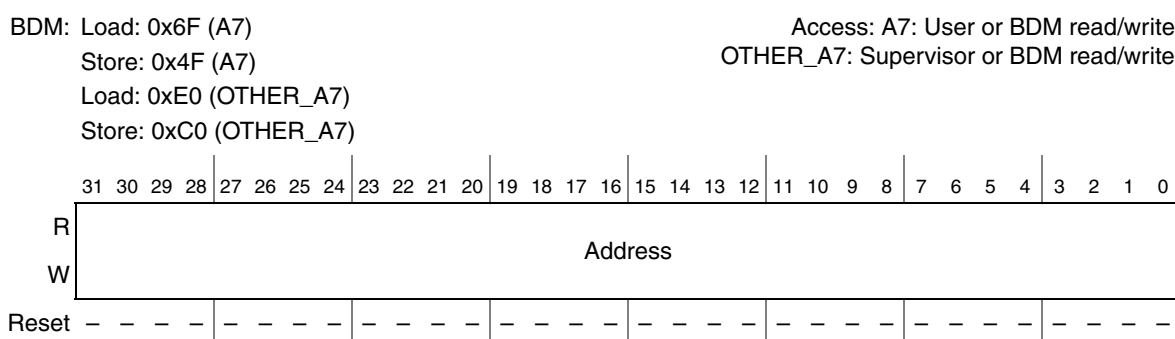


Figure 7-4. Stack Pointer Registers (A7 and OTHER\_A7)

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

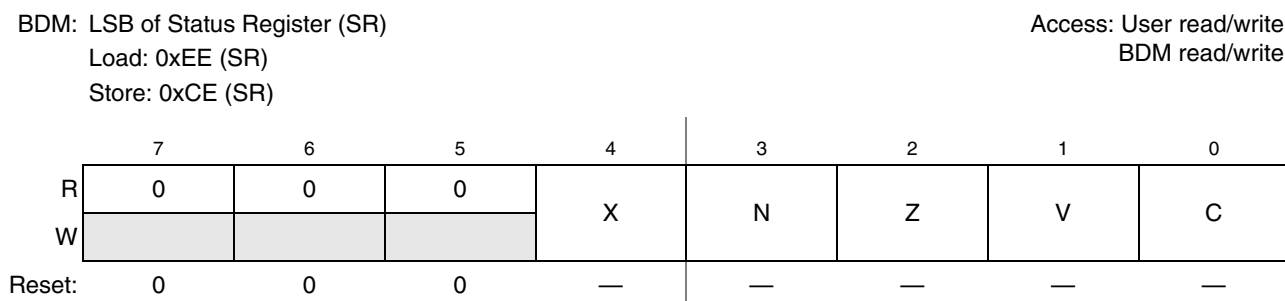


Figure 7-5. Condition Code Register (CCR)

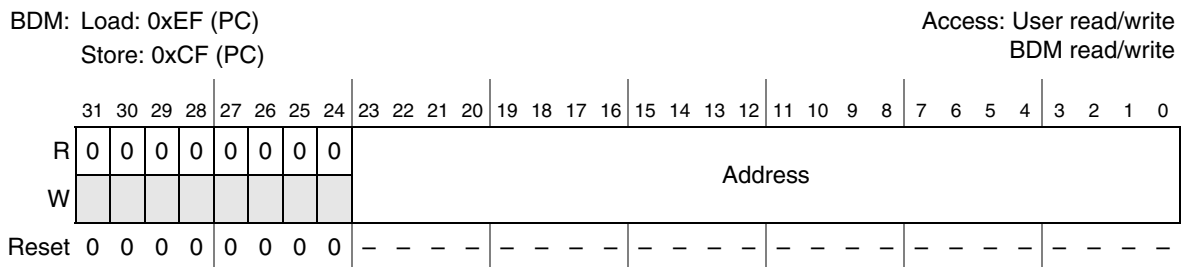
**Table 7-2. CCR Field Descriptions**

<b>Field</b>	<b>Description</b>
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.

## 7.2.5 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 0x(00)00\_0004.

**Figure 7-6. Program Counter Register (PC)**

## 7.2.6 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.

In addition, because the V1 ColdFire core supports a 16 MB address space, the upper byte of the VBR is also forced to zero. The VBR can be used to relocate the exception vector table from its default position in the flash memory (address 0x(00)00\_0000) to the base of the RAM (address 0x(00)80\_0000) if needed.



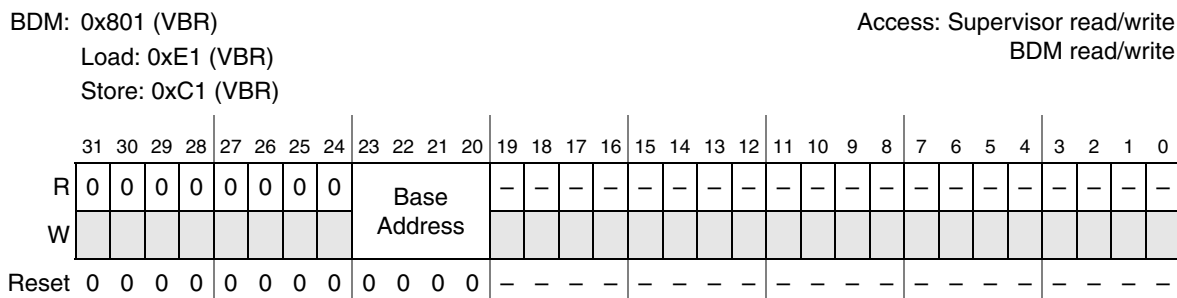


Figure 7-7. Vector Base Register (VBR)

## 7.2.7 CPU Configuration Register (CPUCR)

The CPUCR provides supervisor mode configurability of specific core functionality. Certain hardware features can be enabled/disabled individually based on the state of the CPUCR.

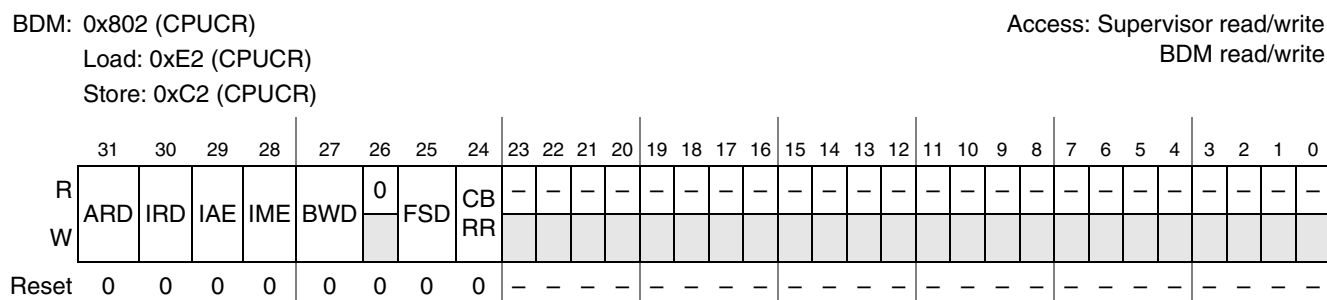


Figure 7-8. CPU Configuration Register (CPUCR)

Table 7-3. CPUCR Field Descriptions

Field	Description
31 ARD	Address-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by an address error, a bus error, an RTE format error, or a fault-on-fault halt condition. 0 The detection of these types of exception conditions or the fault-on-fault halt condition generate a reset event. 1 No reset is generated in response to these exception conditions.
30 IRD	Instruction-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by the attempted execution of an illegal instruction (except for the ILLEGAL opcode), illegal line A, illegal line F instructions, or a privilege violation. 0 The detection of these types of exception conditions generate a reset event. 1 No reset is generated in response to these exception conditions.
29 IAE	Interrupt acknowledge (IACK) enable. Forces the processor to generate an IACK read cycle from the interrupt controller during exception processing to retrieve the vector number of the interrupt request being acknowledged. The processor's execution time for an interrupt exception is slightly improved when this bit is cleared. 0 The processor uses the vector number provided by the interrupt controller at the time the request is signaled. 1 IACK read cycle from the interrupt controller is generated.
28 IME	Interrupt mask enable. Forces the processor to raise the interrupt level mask (SR[I]) to 7 during every interrupt exception. 0 As part of an interrupt exception, the processor sets SR[I] to the level of the interrupt being serviced. 1 As part of an interrupt exception, the processor sets SR[I] to 7. This disables all level 1-6 interrupt requests but allows recognition of the edge-sensitive level 7 requests.

Table 7-3. CPUCR Field Descriptions (continued)

Field	Description
27 BWD	Buffered write disable. The ColdFire core is capable of marking processor memory writes as bufferable or non-bufferable. 0 Writes are buffered and the bus cycle is terminated immediately with zero wait states. 1 Disable the buffering of writes. In this configuration, the write transfer is terminated based on the response time of the addressed destination memory device. <b>Note:</b> If buffered writes are enabled (BWD = 0), any error status is lost as the immediate termination of the data transfer assumes an error-free completion.
26	Reserved, must be cleared.
25 FSD	Flash speculation disabled. Disables certain performance-enhancing features related to address speculation in the flash memory controller. 0 The flash controller tries to speculate on read accesses to improve processor performance by minimizing the exposed flash memory access time. Recall the basic flash access time is two processor cycles. 1 Certain flash address speculation is disabled.
24 CBRR	Crossbar round-robin arbitration enable. Configures the crossbar slave ports to fixed-priority or round-robin arbitration. 0 Fixed-priority arbitration 1 Round robin arbitration
23–0	Reserved, must be cleared.

## 7.2.8 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. 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: Load: 0xEE (SR)

Store: 0xCE (SR)

Access: Supervisor read/write

BDM read/write

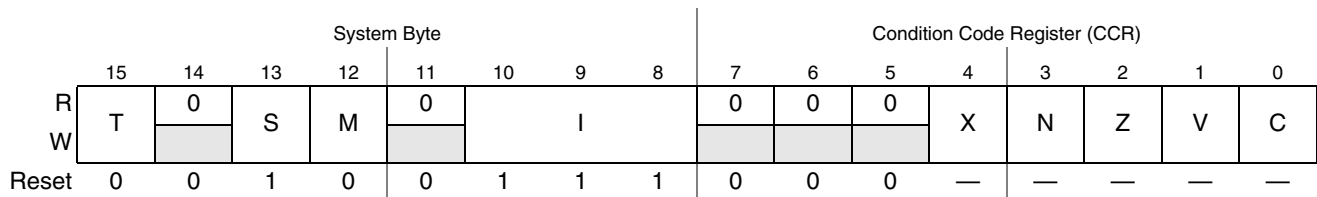


Figure 7-9. Status Register (SR)

Table 7-4. 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.

Table 7-4. SR Field Descriptions (continued)

Field	Description
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 <a href="#">Section 7.2.4, “Condition Code Register (CCR)”</a> .

## 7.3 Functional Description

### 7.3.1 Instruction Set Architecture (ISA\_C)

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 new 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 7-5](#) summarizes the instructions added to revision ISA\_A to form revision ISA\_C. For more details see the *ColdFire Family Programmer's Reference Manual*.

**Table 7-5. Instruction Enhancements over Revision ISA\_A**

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; that is, 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; that is, 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.
MOV3Q.L	Moves 3-bit immediate data to the destination location.
Move from USP	User Stack Pointer → Destination register
Move to USP	Source register → User Stack Pointer
MVS.{B,W}	Sign-extends source operand and moves it to destination register.
MVZ.{B,W}	Zero-fills source operand and moves it to destination register.
SATS.L	Performs saturation operation for signed arithmetic and updates destination register, depending on CCR[V] and bit 31 of the register.
TAS.B	Performs indivisible read-modify-write cycle to test and set addressed memory byte.
Bcc.L	Branch conditionally, longword
BSR.L	Branch to sub-routine, longword
CMP.{B,W}	Compare, byte and word
CMPA.W	Compare address, word
CMPI.{B,W}	Compare immediate, byte and word
MOVEI	Move immediate, byte and word to memory using Ax with displacement
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

### 7.3.2 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
- Use of separate system stack pointers for user and supervisor modes.

All ColdFire processors use an instruction restart exception model. 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 if CPUCR[IAE] is set. The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address. If CPUCR[IAE] is cleared, the processor uses the vector number supplied by the interrupt controller at the time the request was signaled for improved performance.
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 7-10](#), 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  $(4 \times \text{vector number})$ . 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 MB address boundary (see [Table 7-6](#)). For the V1 ColdFire core, the only practical locations for the vector table are based at 0x(00)00\_0000 in the flash or 0x(00)80\_0000 in the internal SRAM.

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 8, "Interrupt Controller \(CF1\\_INTC\)"](#) for details on the device-specific interrupt sources.

For the V1 ColdFire core, the table is partially populated with the first 64 reserved for internal processor exceptions, while vectors 64-102 are reserved for the peripheral I/O requests and the seven software interrupts. Vectors 103–255 are unused and reserved.

**Table 7-6. Exception Vector Assignments**

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter

Table 7-6. Exception Vector Assignments (continued)

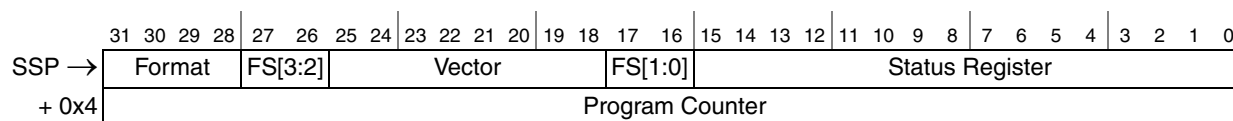
Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5–7	0x014–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented 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
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–60	0x0C0–0x0F0	—	Reserved
61	0x0F4	Fault	Unsupported instruction
62–63	0x0F8–0x0FC	—	Reserved
64–102	0x100–0x198	Next	Device-specific interrupts
103–255	0x19C–0x3FC	—	Reserved

<sup>1</sup> 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\_C 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. Finally, the V1 ColdFire core includes the CPUCR[IME] bit that forces the processor to automatically raise the mask level to 7 during the interrupt exception, removing the need for any explicit instruction in the service routine to perform this function. For more details, see *ColdFire Family Programmer's Reference Manual*.

### 7.3.2.1 Exception Stack Frame Definition

Figure 7-10 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 7-10. 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 7-7](#).

**Table 7-7. 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 7-8](#).

**Table 7-8. Fault Status Encodings**

FS[3:0]	Definition
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Reserved
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 7-6](#).

## 7.3.3 Processor Exceptions

### 7.3.3.1 Access Error Exception

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an access error (also known as a bus error) is detected. If CPUCCR[ARD] is set, the reset is disabled and a processor exception is generated as detailed below.

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 V1 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.

### 7.3.3.2 Address Error Exception

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an address error is detected. If CPUCCR[ARD] equals 1, then the reset is disabled and a processor exception is generated as detailed below.

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 an RTS instruction, the Version 1 ColdFire processor overwrites the faulting return PC with the address error stack frame.



### 7.3.3.3 Illegal Instruction Exception

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an illegal instruction is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below. There is one special case involving the ILLEGAL opcode (0x4AFC); attempted execution of this instruction always generates an illegal instruction exception, regardless of the state of the CPUCR[IRD] bit.

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 7-11. The opword line definition is shown in Table 7-9.

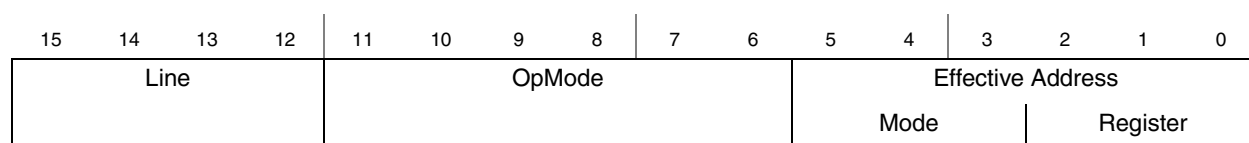


Figure 7-11. ColdFire Instruction Operation Word (Opword) Format

Table 7-9. 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)
0xA	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	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 opcodes 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 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.

The V1 ColdFire processor also detects two special cases involving illegal instruction conditions:

1. If execution of the stop instruction is attempted and neither low-power stop nor wait modes are enabled, the processor signals an illegal instruction.
2. If execution of the halt instruction is attempted and BDM is not enabled (XCSR[ENBDM] equals 0), the processor signals an illegal instruction.

In both cases, the processor response is then dependent on the state of CPUCR[IRD]— a reset event or a processor exception.

### 7.3.3.4 Privilege Violation

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if a privilege violation is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### 7.3.3.5 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.

### 7.3.3.6 Unimplemented Line-A Opcode

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an unimplemented line-A opcode is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### 7.3.3.7 Unimplemented Line-F Opcode

The default operation of the V1 ColdFire processor is the generation of an illegal opcode reset event if an unimplemented line-F opcode is detected. If CPUCR[IRD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### 7.3.3.8 Debug Interrupt

See [Chapter 26, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) 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.

### 7.3.3.9 RTE and Format Error Exception

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if an RTE format error is detected. If CPUCR[ARD] is set, the reset is disabled and a processor exception is generated as detailed below.

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.

### 7.3.3.10 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.

This set of 16 instructions provides a similar but expanded functionality compared to the S08's SWI (software interrupt) instruction. Do not confuse these instructions and their functionality with the software-scheduled interrupt requests, which are handled like normal I/O interrupt requests by the interrupt controller. The processing of the software-scheduled IRQs can be masked, based on the interrupt priority level defined by the SR[I] field.

### 7.3.3.11 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 7.3.3.14, "Reset Exception,"](#) for details.

For this device, attempted execution of valid integer divide opcodes, CAU, and all MAC and EMAC instructions result in the unsupported instruction exception.

### 7.3.3.12 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle or using the previously-supplied vector number, under control of CPUCCR[IAE]. See [Chapter 8, "Interrupt Controller \(CF1\\_INTC\),"](#) for details on the interrupt controller.

### 7.3.3.13 Fault-on-Fault Halt

The default operation of the V1 ColdFire processor is the generation of an illegal address reset event if a fault-on-fault halt condition is detected. If CPUCCR[ARD] is set, the reset is disabled and the processor is halted as detailed below.

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 exit this state.

### 7.3.3.14 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 0x(00)00\_0000 is loaded into the supervisor stack pointer and the second longword at address 0x(00)00\_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 7-12](#).

BDM: Load: 0x60 (D0) Store: 0x40 (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	0	1	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	0	0	0	0	0	0	ISA				DEBUG			
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

Figure 7-12. D0 Hardware Configuration Info

Table 7-10. 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 (This is the value used for this device.) 0010 V2 ColdFire core 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. 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. (This is the value used for this device.) 1 Divide execute engine is present in core.
13–8	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 (This is the value used for this device.) 1000 ISA_A+ 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: 0x61 (D1)  
Store: 0x41 (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	0	0	0	1	0	0	0	0	FLASHSZ <sup>1</sup>				0	0	0	
W																
Reset	0	0	0	1	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	0	0	0	1	ROMSZ				SRAMSZ				0	0	0	
W																
Reset	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0

<sup>1</sup> The FLASHSZ size depends on memory size. The size shown is for 128 KB flash.

**Figure 7-13. D1 Hardware Configuration Info**

**Table 7-11. D1 Hardware Configuration Information Field Description**

Field	Description
31–24	Reserved.
23–19 FLASHSZ	Flash bank size. 00000-01110 No flash 10000 64 KB flash 10010 128 KB flash (This is the value used for this device) 10011 96 KB flash 10100 256 KB flash 10110 512 KB flash Else Reserved for future use <b>Note:</b> The FLASHSZ size depends on memory size. The size shown is for 128 KB flash.
18–16	Reserved
15–12	Reserved, resets to 0b0001
11–8 ROMSZ	Boot ROM size. Indicates the size of the boot ROM. 0000 No boot ROM (This is the value used for this device) 0001 512 bytes 0010 1 KB 0011 2 KB 0100 4 KB 0101 8 KB 0110 16 KB 0111 32 KB Else Reserved for future use

Table 7-11. D1 Hardware Configuration Information Field Description (continued)

Field	Description
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 10000 64 KB 10010 128 KB Else Reserved for future use
2–0	Reserved.

### 7.3.4 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.

#### 7.3.4.1 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 7-12](#).

**Table 7-12. 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

### 7.3.4.2 MOVE Instruction Execution Times

[Table 7-13](#) lists execution times for MOVE.{B,W} instructions; [Table 7-14](#) 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 7-13. 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)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(Ay)+	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d16,Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,Ay,Xi*SF)	3(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.w	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.l	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—

Table 7-13. MOVE Byte and Word Execution Times (continued)

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

Table 7-14. 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)	—	—
(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)	—	—	—

### 7.3.4.3 Standard One Operand Instruction Execution Times

Table 7-15. 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)	—	—	—	—	—	—	—

Table 7-15. One Operand Instruction Execution Times (continued)

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

### 7.3.4.4 Standard Two Operand Instruction Execution Times

Table 7-16. 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)	—	—	—

Table 7-16. 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
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.B	<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)
CMP.W	<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)
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.B	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.W	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
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)	—	—	—	—	—	—	—
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)	—	—	—	—	—	—	—

### 7.3.4.5 Miscellaneous Instruction Execution Times

Table 7-17. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
LINK.W	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
MOV3Q.L	#imm,<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(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)	—	—	—	—	—	—	—

Table 7-17. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
MOVE.W	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) <sup>2</sup>
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)	—	—	—
MVS	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
MVZ	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
NOP		3(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	2(0/1)	—	—	2(0/1) <sup>4</sup>	3(0/1) <sup>5</sup>	2(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
STOP	#imm	—	—	—	—	—	—	—	3(0/0) <sup>3</sup>
TRAP	#imm	—	—	—	—	—	—	—	12(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPF.L		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2(1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
WDEBUD	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

<sup>1</sup>The n is the number of registers moved by the MOVEM opcode.

<sup>2</sup>If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).

<sup>3</sup>The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

<sup>4</sup>PEA execution times are the same for (d16,PC).

<sup>5</sup>PEA execution times are the same for (d8,PC,Xn\*SF).

### 7.3.4.6 Branch Instruction Execution Times

Table 7-18. 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)	—

Table 7-18. General Branch Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
RTE		—	—	7(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

Table 7-19. 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 8

## Interrupt Controller (CF1\_INTC)

### 8.1 Introduction

The CF1\_INTC interrupt controller (CF1\_INTC) is intended for use in low-cost microcontroller designs using the Version 1 (V1) ColdFire processor core. In keeping with the general philosophy for devices based on this low-end 32-bit processor, the interrupt controller generally supports less programmability compared to similar modules in other ColdFire microcontrollers and embedded microprocessors. However, CF1\_INTC provides the required functionality with a minimal silicon cost.

These requirements guide the CF1\_INTC module definition to support Freescale's Controller Continuum:

- The priorities of the interrupt requests between comparable HCS08 and V1 ColdFire devices are identical.
- Supports a mode of operation (through software convention with hardware assists) equivalent to the S08's interrupt processing with only one level of nesting.
- Leverages the current ColdFire interrupt controller programming model and functionality, but with a minimal hardware implementation and cost.

Table 8-1 provides a high-level architectural comparison between HCS08 and ColdFire exception processing as these differences are important in the definition of the CF1\_INTC module. Throughout this document, the term IRQ refers to an interrupt request and ISR refers to an interrupt service routine to process an interrupt exception.

**Table 8-1. Exception Processing Comparison**

Attribute	HCS08	V1 ColdFire
Exception Vector Table	32 two-byte entries, fixed location at upper end of memory	115 four-byte entries, located at lower end of memory at reset, relocatable with the VBR
More on Vectors	2 for CPU + 30 for IRQs, reset at upper address	64 for CPU + 51 for IRQs, reset at lowest address
Exception Stack Frame	5-byte frame: CCR, A, X, PC	8-byte frame: F/V, SR, PC; General-purpose registers (An, Dn) must be saved/restored by the ISR
Interrupt Levels	1 = f(CCR[I])	7 = f(SR[I]) with automatic hardware support for nesting
Non-Maskable IRQ Support	No	Yes, with level 7 interrupts
Core-enforced IRQ Sensitivity	No	Level 7 is edge sensitive, else level sensitive
INTC Vectoring	Fixed priorities and vector assignments	Fixed priorities and vector assignments, plus any 2 IRQs can be remapped as the highest priority level 6 requests

Table 8-1. Exception Processing Comparison (continued)

Attribute	HCS08	V1 ColdFire
Software IACK	No	Yes
Exit Instruction from ISR	RTI	RTE

### 8.1.1 Overview

Interrupt exception processing includes interrupt recognition, aborting the current instruction execution stream, storing an 8-byte exception stack frame in the memory, calculation of the appropriate vector, and passing control to the specified interrupt service routine.

Unless specifically noted otherwise, all ColdFire processors sample for interrupts once during each instruction's execution during the first cycle of execution in the OEP. Additionally, all ColdFire processors use an instruction restart exception model.

The ColdFire processor architecture defines a 3-bit interrupt priority mask field in the processor's status register (SR[I]). This field, and the associated hardware, support seven levels of interrupt requests with the processor providing automatic nesting capabilities. The levels are defined in descending numeric order with  $7 > 6 \dots > 1$ . Level 7 interrupts are treated as non-maskable, edge-sensitive requests while levels 6–1 are maskable, level-sensitive requests. The SR[I] field defines the processor's current interrupt level. The processor continuously compares the encoded IRQ level from CF1\_INTC against SR[I]. Recall that interrupt requests are inhibited for all levels less than or equal to the current level, except the edge-sensitive level 7 request that cannot be masked.

Exception processing for ColdFire processors is streamlined for performance and includes all actions from detecting the fault condition to the initiation of fetch for the first handler instruction. Exception processing is comprised of four major steps.

1. The processor makes an internal copy of the status register (SR) and enters supervisor mode by setting SR[S] and disabling trace mode by clearing SR[T]. The occurrence of an interrupt exception also forces the master mode (M) bit to clear and the interrupt priority mask (I) to set to the level of the current interrupt request.
2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on the exception type. For interrupts, the processor performs an IACK bus cycle to obtain the vector number from the interrupt controller if CPUCR[IAE] equals 1. The IACK cycle is mapped to special locations within the interrupt controller's IPS address space with the interrupt level encoded in the address. If CPUCR[IAE] equals 0, the processor uses the vector number supplied by the interrupt controller at the time the request was signaled (for improved performance).
3. The processor saves the current context by creating an exception stack frame on the system stack. As a result, exception stack frame is created at a 0-modulo-4 address on top of the system stack defined by the supervisor stack pointer (SSP). The processor uses an 8-byte stack frame for all exceptions. It contains the vector number of the exception, the contents of the status register at the time of the exception, and the program counter (PC) at the time of the exception. 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). For interrupts, the stacked PC is always the address of the next instruction to be executed.

4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1MB boundary. This instruction address is generated by fetching a 32-bit exception vector from the table located at the address defined in the vector base register (VBR). The index into the exception table is calculated as  $(4 \times \text{vector number})$ . After the exception vector has been fetched, the contents of the vector serves as a 32-bit pointer to the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has been 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-MB address boundary. For the V1 ColdFire core, the only practical locations for the vector table are based at 0x(00)00\_0000 in the flash or 0x(00)80\_0000 in the RAM. The table contains 256 exception vectors; the first 64 are reserved for internal processor exceptions, and the remaining 192 are device-specific interrupt vectors. The IRQ assignment table is partially populated depending on the exact set of peripherals for the given device.

The exception vector table for MCF51AG128 series devices is shown in [Table 8-2](#).

**Table 8-2. MCF51AG128 Series Exception and Interrupt Vector Table**

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
2–63	0x008–0x0FC	—	Reserved for internal CPU exceptions
64	0x100	Next	IRQ_pin
65	0x104	Next	low_voltage
66	0x108	Next	Reserved
67	0x10C	Next	Reserved
68	0x110	Next	DMA_ch0
69	0x114	Next	DMA_ch1
70	0x118	Next	DMA_ch2
71	0x11C	Next	DMA_ch3
72	0x120	Next	iEvent_ch0
73	0x124	Next	FTM1–(fault + ovfl)
74	0x128	Next	FTM1_ch0
75	0x12C	Next	FTM1_ch1
76	0x130	Next	FTM1_ch2
77	0x134	Next	FTM1_ch3

Table 8-2. MCF51AG128 Series Exception and Interrupt Vector Table (continued)

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
78	0x138	Next	FTM1_ch4
79	0x13C	Next	FTM1_ch5
80	0x140	Next	FTM2--(fault + ovfl)
81	0x144	Next	FTM2_ch0
82	0x148	Next	FTM2_ch1
83	0x14C	Next	FTM2_ch2
84	0x150	Next	FTM2_ch3
85	0x154	Next	FTM2_ch4
86	0x158	Next	FTM2_ch5
87	0x15C	Next	TPM3_ovfl
88	0x160	Next	TPM3_ch0
89	0x164	Next	TPM3_ch1
90	0x168	Next	ADC
91	0x16C	Next	HSCMP1
92	0x170	Next	HSCMP2
93	0x174	Next	iEvent_ch1
94	0x178	Next	SPI1
95	0x17c	Next	SPI2
96	0x180	Next	SCI1_err
97	0x184	Next	SCI1_rx
98	0x188	Next	SCI1_tx
99	0x18C	Next	IIC
100	0x190	Next	iEvent_ch2
101	0x194	Next	SCI2_err
102	0x198	Next	SCI2_rx
103	0x19c	Next	Level 7 Software Interrupt
104	0x1A0	Next	Level 6 Software Interrupt
105	0x1A4	Next	Level 5 Software Interrupt
106	0x1A8	Next	Level 4 Software Interrupt
107	0x1AC	Next	Level 3 Software Interrupt
108	0x1B0	Next	Level 2 Software Interrupt
109	0x1B4	Next	Level 1 Software Interrupt
110	0x1B8	Next	SCI2_tx

**Table 8-2. MCF51AG128 Series Exception and Interrupt Vector Table (continued)**

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
111	0x1BC	Next	KBI[A:E]
112	0x1C0	Next	KBI[F:J]
113	0x1C4	Next	RTC + WDOG
114	0x1C8	Next	iEvent_ch3
115–255	0x1CC–0x3FC	—	Reserved

The basic ColdFire interrupt controller supports up to 63 request sources mapped as nine priorities for each of the seven supported levels (7 levels × 9 priorities per level). Within the nine priorities within a level, the mid-point is typically reserved for package-level IRQ inputs. The levels and priorities within the level follow a descending order: 7 > 6 > ... > 1 > 0.

The HCS08 architecture supports a 32-entry exception vector table: the first two vectors are reserved for internal CPU/system exceptions and the remaining are available for I/O interrupt requests. The requirement for an exact match between the interrupt requests and priorities across two architectures means the sources are mapped to a sparsely-populated two-dimensional ColdFire array of seven interrupt levels and nine priorities within the level. The following association between the HCS08 and ColdFire vector numbers applies:

$$\text{ColdFire Vector Number} = 62 + \text{HCS08 Vector Number}$$

The CF1\_INTC performs a cycle-by-cycle evaluation of the active requests and signals the highest-level, highest-priority request to the V1 ColdFire core in the form of an encoded interrupt level and the exception vector associated with the request. The module also includes a byte-wide interface to access its programming model. These interfaces are shown in the simplified block diagram of [Figure 8-1](#).

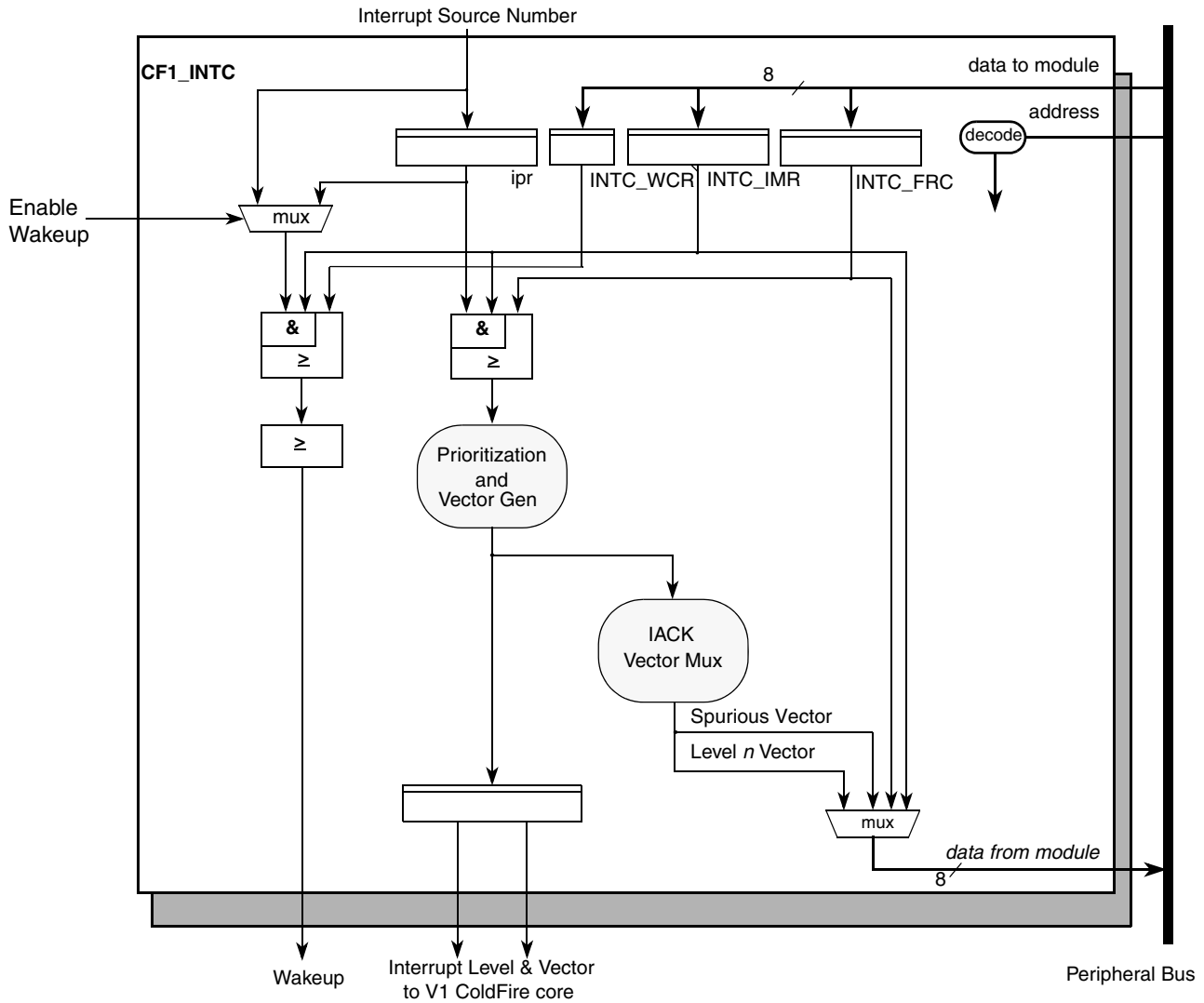


Figure 8-1. CF1\_INTC Block Diagram

### 8.1.2 Features

The Version 1 ColdFire interrupt controller includes:

- Memory-mapped off-platform slave module
  - 64-byte space located at top end of memory: 0x(FF)FF\_FFC0–0x(FF)FF\_FFFF
  - Programming model accessed via the peripheral bus
  - Encoded interrupt level and vector sent directly to processor core
- Support of 44 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests
- Fixed association between interrupt request source and level plus priority
  - 44 I/O requests assigned across seven available levels and nine priorities per level

- Exactly matches HCS08 interrupt request priorities
- Up to two requests can be remapped to the highest maskable level + priority
- Unique vector number for each interrupt source
  - ColdFire vector number = 62 + HCS08 vector number
  - Details on IRQ and vector assignments are device-specific
- Support for service routine interrupt acknowledge (software IACK) read cycles for improved system performance
- Combinatorial path provides wakeup signal from wait mode
- Ability to mask any individual interrupt source, plus global mask-all capability

### 8.1.3 Modes of Operation

The CF1\_INTC module does not support any special modes of operation. As a memory-mapped slave peripheral located on the platform's slave bus, it responds based strictly on the memory addresses of the connected bus.

One special behavior of the CF1\_INTC deserves mention. When the device enters a wait mode and certain clocks are disabled, there is an input signal that can be asserted to enable a purely-combinational logic path for monitoring the assertion of an interrupt request. After a request of unmasked level is asserted, this combinational logic path asserts an output signal that is sent to the clock generation logic to re-enable the internal device clocks to exit the low-power mode.

## 8.2 External Signal Description

The CF1\_INTC module does not include any external interfaces.

## 8.3 Memory Map/Register Definition

The CF1\_INTC module provides a 64-byte programming model mapped to the upper region of the 16 MB address space. All the register names are prefixed with INTC\_ as an abbreviation for the full module name.

The programming model is referenced using 8-bit accesses. Attempted references to undefined (reserved) addresses or with a non-supported access type (for example, a write to a read-only register) generate a bus error termination.

The programming model follows the definition from previous ColdFire interrupt controllers. This compatibility accounts for the various memory holes in this module's memory map.

The CF1\_INTC module is based at address 0x(FF)FF\_FFC0 (referred to as CF1\_INTC\_BASE throughout the chapter) and occupies the upper 64 bytes of the peripheral space. The module memory map is shown in [Table 8-3](#).

Table 8-3. CF1\_INTC Memory Map

Offset Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/ Page
0x08	INTC_IMRH	CF1_INTC Mask Register High	32	R/W	0x0000_0000	8.3.1/8-8
0x0C	INTC_IMRL	CF1_INTC Mask Register Low	32	R/W	0x0000_0000	8.3.1/8-8
0x10	INTC_FRC	CF1_INTC Force Interrupt Register	8	R/W	0x00	8.3.2/8-10
0x18	INTC_PL6P7	CF1_INTC Programmable Level 6, Priority 7	8	R/W	0x00	8.3.3/8-11
0x19	INTC_PL6P6	CF1_INTC Programmable Level 6, Priority 6	8	R/W	0x00	8.3.3/8-11
0x1B	INTC_WCR	CF1_INTC Wakeup Control Register	8	R/W	0x80	8.3.4/8-12
0x1C	INTC_SIMR	CF1_INTC Set Interrupt Mask Register	8	Write	—	8.3.5/8-13
0x1D	INTC_CIMR	CF1_INTC Clear Interrupt Mask Register	8	Write	—	8.3.6/8-13
0x1E	INTC_SFRC	CF1_INTC Set Interrupt Force Register	8	Write	—	8.3.7/8-14
0x1F	INTC_CFRC	CF1_INTC Clear Interrupt Force Register	8	Write	—	8.3.8/8-15
0x20	INTC_SWIACK	CF1_INTC Software Interrupt Acknowledge	8	Read	0x00	8.3.9/8-16
0x24	INTC_LVL1IACK	CF1_INTC Level 1 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x28	INTC_LVL2IACK	CF1_INTC Level 2 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x2C	INTC_LVL3IACK	CF1_INTC Level 3 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x30	INTC_LVL4IACK	CF1_INTC Level 4 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x34	INTC_LVL5IACK	CF1_INTC Level 5 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x38	INTC_LVL6IACK	CF1_INTC Level 6 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16
0x3C	INTC_LVL7IACK	CF1_INTC Level 7 Interrupt Acknowledge	8	Read	0x18	8.3.9/8-16

### 8.3.1 Interrupt Mask Registers (INTC\_IMR{H,L})

The INTC\_IMRH and INTC\_IMRL registers are each 32 bits in size, and provide a bit map for each interrupt to allow the request to be disabled (masked) (1 = disable the request, 0 = enable the request). The IMR is cleared by reset, enabling all interrupt requests to preserve compatibility with earlier V1 ColdFire devices. The IMR can be read and written directly, or individual mask flags can be set or cleared by accessing set/clear interrupt mask registers (INTC\_SIMR, INTC\_CIMR).

Each bit of the IMR[n] is associated with a corresponding bit of the interrupt request input vector. The equations defining this association are:

For Vectors 64–102,  $n = \text{Vector\_Number} - 64$ , else for Vectors 110–114,  $n = \text{Vector\_Number} - 71$

Therefore, vector 64 corresponds to  $n = 0$ , vector 65 to  $n = 1$ , etc., vector 113 to  $n = 42$ , and vector 114 to  $n = 43$ .

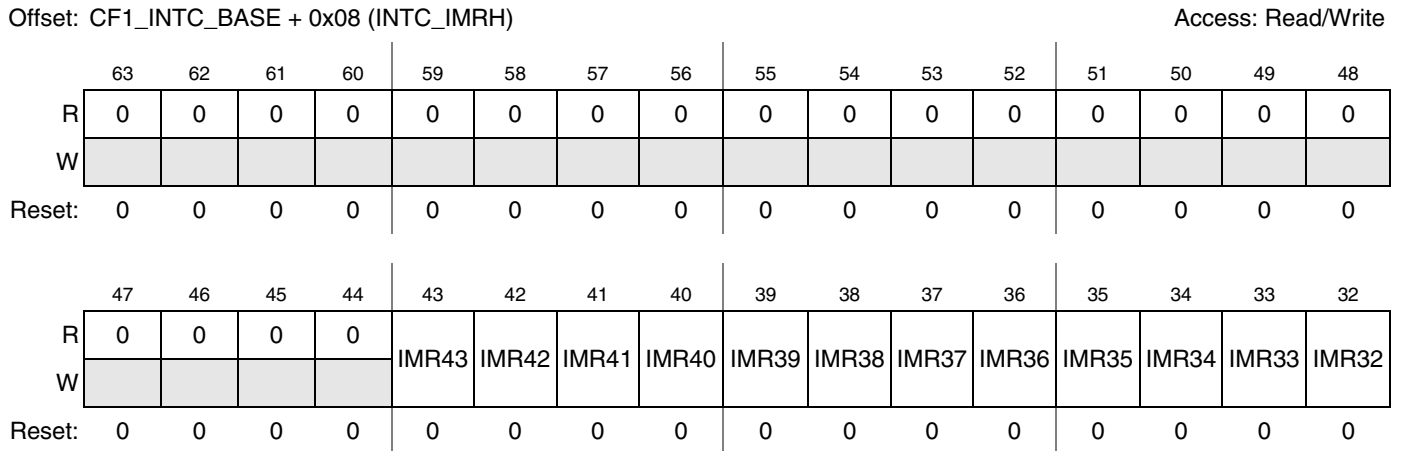
Each peripheral request input is first qualified by the contents of the IMR registers before it is used elsewhere in the interrupt controller, that is:

$$\text{qualified\_interrupt\_request}[n] = \text{interrupt\_request\_input}[n] \ \& \ \sim\text{INTC\_IMR}[n]$$

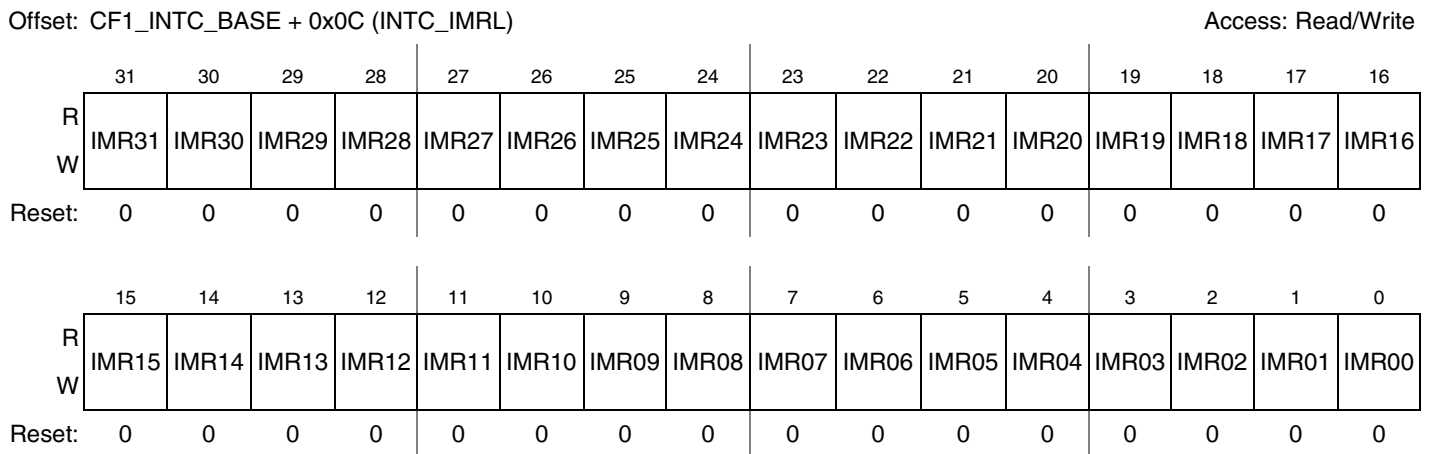
Since this interrupt controller supports 44 request inputs, the upper 20 bits of the INTC\_IMRH are reserved for future use. Writes to these bits are ignored and reads return zeroes.

The contents of the IMR do not affect the operation of the software settable force interrupt registers.

See [Figure 8-2](#), [Figure 8-3](#), and [Table 8-4](#) for the INTC\_IMR{H,L} register definitions.



**Figure 8-2. Interrupt Mask Register High (INTC\_IMRH)**



**Figure 8-3. Interrupt Mask Register Low (INTC\_IMRL)**

**Table 8-4. INTC\_IMR{H,L} Field Descriptions**

Field	Description
IMRn, n = 0,... 43	Interrupt mask register n. 0 The interrupt request n is enabled. 1 The interrupt request n is disabled.

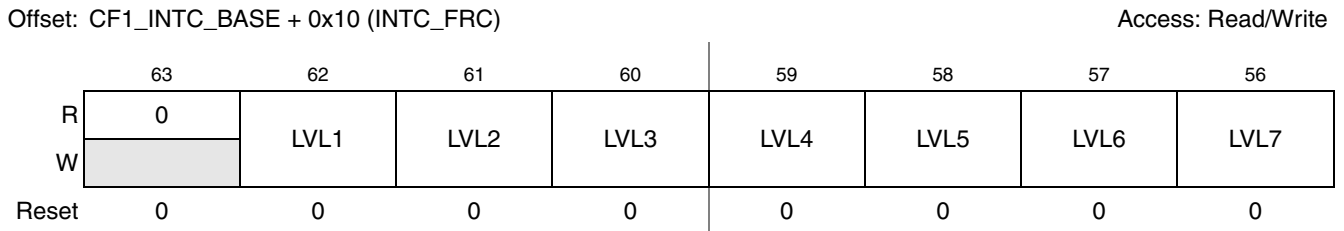
### 8.3.2 Force Interrupt Register (INTC\_FRC)

The INTC\_FRC register allows software to generate a unique interrupt for each possible level at the lowest priority within the level for functional or debug purposes. These interrupts may be self-scheduled by setting one or more of the bits in the INTC\_FRC register. In some cases, the handling of a normal interrupt request may cause critical processing by the service routine along with the scheduling (using the INTC\_FRC register) of a lower priority level interrupt request to be processed at a later time for less-critical task handling.

The INTC\_FRC register may be modified directly using a read-modify-write sequence or through a simple write operation using the set/clear force interrupt registers (INTC\_SFRC, INTC\_CFRC).

**NOTE**

Take special notice of the bit numbers within this register, 63–56. This is for compatibility with other ColdFire interrupt controllers.



**Figure 8-4. Force Interrupt Register (INTC\_FRC)**

**Table 8-5. INTC\_FRC Field Descriptions**

Field	Description
63	Reserved, must be cleared.
62 LVL1	Force Level 1 interrupt. 0 Negates the forced level 1 interrupt request. 1 Forces a level 1 interrupt request.
61 LVL2	Force Level 2 interrupt. 0 Negates the forced level 2 interrupt request. 1 Forces a level 2 interrupt request.
60 LVL3	Force Level 3 interrupt. 0 Negates the forced level 3 interrupt request. 1 Forces a level 3 interrupt request.
59 LVL4	Force Level 4 interrupt. 0 Negates the forced level 4 interrupt request. 1 Forces a level 4 interrupt request.
58 LVL5	Force Level 5 interrupt. 0 Negates the forced level 5 interrupt request. 1 Forces a level 5 interrupt request.



Table 8-5. INTC\_FRC Field Descriptions (continued)

Field	Description
57 LVL6	Force Level 6 interrupt. 0 Negates the forced level 6 interrupt request. 1 Forces a level 6 interrupt request.
55 LVL7	Force Level 7 interrupt. 0 Negates the forced level 7 interrupt request. 1 Forces a level 7 interrupt request.

### 8.3.3 INTC Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})

The level seven interrupt requests cannot have their levels reassigned. However, any of the remaining peripheral interrupt requests can be reassigned as the highest priority maskable requests using these two registers (INTC\_PL6P7 and INTC\_PL6P6). The vector number associated with the interrupt requests does not change. Rather, only the interrupt request's level and priority are altered, based on the contents of the INTC\_PL6P{7,6} registers.

#### NOTE

The requests associated with the INTC\_FRC register have a fixed level and priority that cannot be altered.

The INTC\_PL6P7 register specifies the highest-priority, maskable interrupt request that is defined as the level six, priority seven request. The INTC\_PL6P6 register specifies the second-highest-priority, maskable interrupt request defined as the level six, priority six request. Reset clears both registers, disabling any request re-mapping.

For an example of the use of these registers, see [Section 8.6.2, “Using INTC\\_PL6P{7,6} Registers.”](#)



Figure 8-5. Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})

Table 8-6. INTC\_PL6P{7,6} Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 REQN	Request number. Defines the peripheral IRQ number to be remapped as the level 6, priority 7 (for INTC_PL6P7) request and level 6, priority 6 (for INTC_PL6P6) request. <b>Note:</b> The value must be a valid interrupt number. Unused or reserved interrupt numbers are ignored.

### 8.3.4 INTC Wakeup Control Register (INTC\_WCR)

The interrupt controller provides a combinatorial logic path to generate a special wakeup signal to exit from the wait mode. The INTC\_WCR register defines wakeup condition for interrupt recognition during wait mode. This mode of operation works as follows:

1. Write to the INTC\_WCR to enable this operation (set INTC\_WCR[ENB]) and define the interrupt mask level needed to force the core to exit wait mode (INTC\_WCR[MASK]). The maximum value of INTC\_WCR[MASK] is 0x6 (0b110). The INTC\_WCR is enabled with a mask level of 0 as the default after reset.
2. Execute a stop instruction with the appropriate device configuration bit setting to place the processor into wait mode.
3. After the processor is stopped, the interrupt controller enables special logic that evaluates the incoming interrupt sources in a purely combinatorial path; no clocked storage elements are involved.
4. If an active interrupt request is asserted and the resulting interrupt level is greater than the mask value contained in INTC\_WCR[MASK], the interrupt controller asserts the wakeup output signal. This signal is routed to the clock generation logic to exit the low-power mode and resume processing.

Typically, the interrupt mask level loaded into the processor's status register field (SR[I]) during the execution of the stop instruction matches the INTC\_WCR[MASK] value.

The interrupt controller's wait mode wakeup signal is defined as:

$$\text{wait wakeup} = \text{INTC\_WCR[ENB]} \ \& \ (\text{level of any asserted\_int\_request} > \text{INTC\_WCR[MASK]})$$

Offset: CF1\_INTC\_BASE + 0x1B (INTC\_WCR)

Access: Read/Write

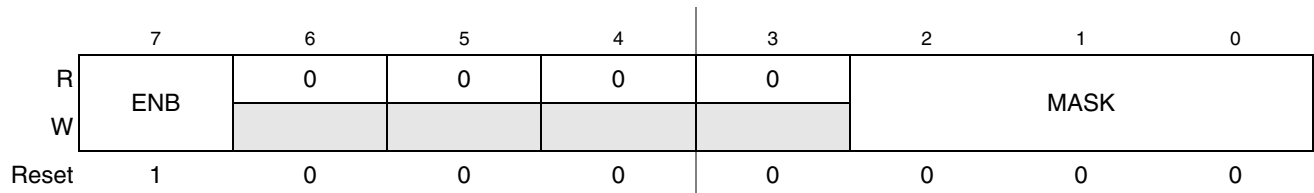


Figure 8-6. Wakeup Control Register (INTC\_WCR)

Table 8-7. INTC\_WCR Field Descriptions

Field	Description
7 ENB	Enable wakeup signal. 0 Wakeup signal disabled 1 Enables the assertion of the combinatorial wakeup signal to the clock generation logic.
6–3	Reserved, must be cleared.
2–0 MASK	Interrupt mask level. Defines the interrupt mask level during wait mode and is enforced by the hardware to be within the range 0–6. If INTC_WCR[ENB] is set, when an interrupt request of a level higher than MASK occurs, the interrupt controller asserts the wait mode wakeup signal to the clock generation logic.

### 8.3.5 Set Interrupt Mask Register (INTC\_SIMR)

The INTC\_SIMR register provides a simple memory-mapped mechanism to set a given bit in the INTC\_IMR{H,L} registers to disable (mask) a given interrupt request. The data value on a register write causes the corresponding bit in the INTC\_IMR{H,L} registers to be set. Setting INTC\_SIMR[SALL] forces the entire contents of INTC\_IMR{H,L} registers to set, masking all interrupts. Attempting to read this register generates an error termination.

IMR[63:44] are reserved for future use, so writes using these data values are ignored.

This register is provided so interrupt service routines can easily mask the given interrupt request without the need to perform a read-modify-write sequence on the INTC\_IMR{H,L} registers.

Offset: CF1\_INTIC\_BASE + 0x1C (INTC\_SIMR)

Access: Write-only



Figure 8-7. Set Interrupt Mask Register (INTC\_SIMR)

Table 8-8. INTC\_SIMR Field Descriptions

Name	Description
7	Reserved, must be cleared.
6 SALL	Set all bits in the INTC_IMR{H,L} register, masking all interrupt requests. 0 Only set those bits specified in the SIMR field. 1 Set all bits in INTC_IMR{H,L} register. The SIMR field is ignored.
5–0 SIMR	Set the corresponding bit in the INTC_IMR{H,L} register, masking the interrupt request.

### 8.3.6 Clear Interrupt Mask Register (INTC\_CIMR)

The INTC\_CIMR register provides a simple memory-mapped mechanism to clear a given bit in the INTC\_IMR{H,L} registers to enable a given interrupt request. The data value on a register write causes the corresponding bit in the INTC\_IMR{H,L} registers to be cleared. Setting INTC\_CIMR[CALL] forces the entire contents of INTC\_IMR{H,L} registers to clear, enabling all interrupts. Attempting to read this register generates an error termination.

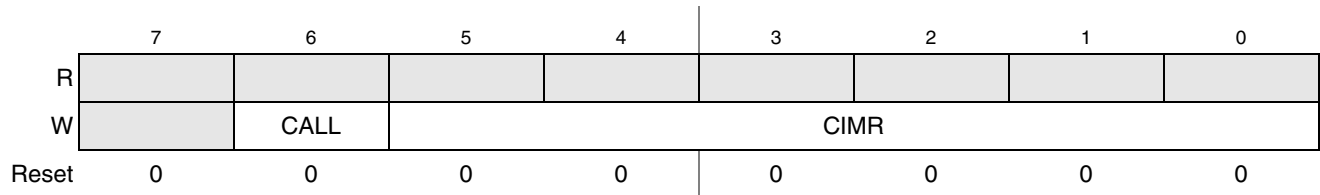
IMR[63:44] are reserved for future use, so writes using these data values are ignored.

This register is provided so interrupt service routines can easily enable the given interrupt request without the need to perform a read-modify-write sequence on the INTC\_IMR{H,L} registers.

## Interrupt Controller (CF1\_INTC)

Offset: CF1\_INTC\_BASE + 0x1D (INTC\_CIMR)

Access: Write-only



**Figure 8-8. Clear Interrupt Mask (INTC\_CIMR)**

**Table 8-9. INTC\_CIMR Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6 CALL	Clear all bits in the INTC_IMR{H,L} register, enabling all interrupt requests. 0 Only set those bits specified in the CIMR field. 1 Clear all bits in INTC_IMR{H,L} register. The CIMR field is ignored.
5-0 CIMR	Clear the corresponding bit in the INTC_IMR{H,L} registers, enabling the interrupt request.

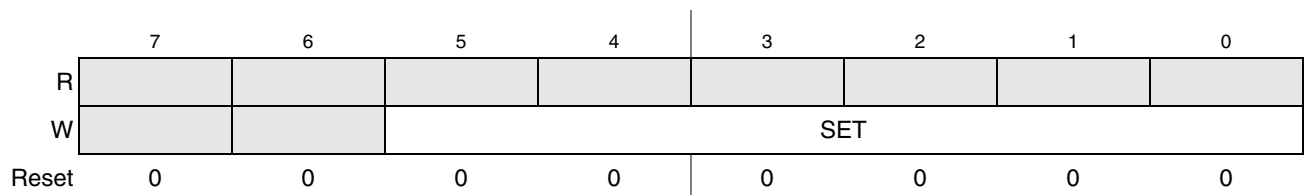
### 8.3.7 INTC Set Interrupt Force Register (INTC\_SFRC)

The INTC\_SFRC register provides a simple memory-mapped mechanism to set a given bit in the INTC\_FRC register to assert a specific level interrupt request. The data value written causes the appropriate bit in the INTC\_FRC register to be set. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can generate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC\_FRC register.

Offset: CF1\_INTC\_BASE + 0x1E (INTC\_SFRC)

Access: Write-only



**Figure 8-9. INTC\_SFRC Register**

Table 8-10. INTC\_SFRC Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SET	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is set, as defined below. 0x38 Bit 56, INTC_FRC[LVL7] is set 0x39 Bit 57, INTC_FRC[LVL6] is set 0x3A Bit 58, INTC_FRC[LVL5] is set 0x3B Bit 59, INTC_FRC[LVL4] is set 0x3C Bit 60, INTC_FRC[LVL3] is set 0x3D Bit 61, INTC_FRC[LVL2] is set 0x3E Bit 62, INTC_FRC[LVL1] is set <b>Note:</b> Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x38–0x3E (56–62) range to ensure compatibility with future devices.

### 8.3.8 INTC Clear Interrupt Force Register (INTC\_CFRC)

The INTC\_CFRC register provides a simple memory-mapped mechanism to clear a given bit in the INTC\_FRC register to negate a specific level interrupt request. The data value on the register write causes the appropriate bit in the INTC\_FRC register to be cleared. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can negate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC\_FRC register.

Offset: CF1\_INTC\_BASE + 0x1F (INTC\_CFRC)

Access: Write-only

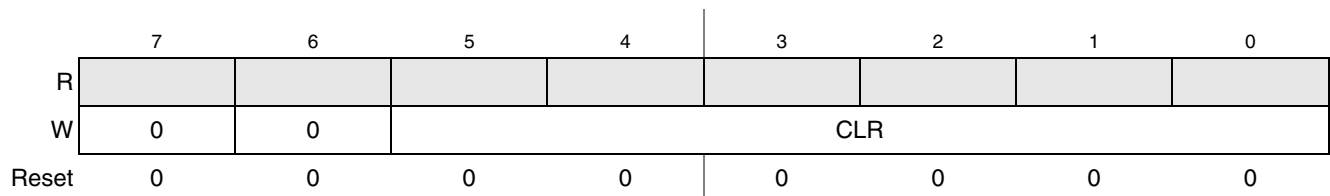


Figure 8-10. INTC\_CFRC Register

Table 8-11. INTC\_CFRC Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 CLR	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is cleared, as defined below. 0x38 Bit 56, INTC_FRC[LVL7] is cleared 0x39 Bit 57, INTC_FRC[LVL6] is cleared 0x3A Bit 58, INTC_FRC[LVL5] is cleared 0x3B Bit 59, INTC_FRC[LVL4] is cleared 0x3C Bit 60, INTC_FRC[LVL3] is cleared 0x3D Bit 61, INTC_FRC[LVL2] is cleared 0x3E Bit 62, INTC_FRC[LVL1] is cleared <b>Note:</b> Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x38–0x3E (56–62) range to ensure compatibility with future devices.

### 8.3.9 INTC Software and Level-*n* IACK Registers (*n* = 1,2,3,...,7)

The eight read-only interrupt acknowledge (IACK) registers can be explicitly addressed by the memory-mapped accesses or implicitly addressed by a processor-generated interrupt acknowledge cycle during exception processing when CPUCR[IAE] is set. In either case, the interrupt controller's actions are similar.

First, consider an IACK cycle to a specific level, a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all currently-active level-*n* 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.

If there is no active interrupt source at the time of the level-*n* IACK, a special spurious interrupt vector (vector number 24 (0x18)) 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 peripheral device by the interrupt service routine. This approach provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

Second, the interrupt controller also supports the concept of a software IACK. This is the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been negated) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the returned value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. If the returned value is zero, there is no pending interrupt request.

This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can noticeably improve overall performance. For additional details on software IACKs, see [Section 8.6.3, “More on Software IACKs.”](#)

Offset: CF1\_INTC\_BASE + 0x20 (INTC\_SWIACK) Access: Read-only  
 CF1\_INTC\_BASE + 0x20 + (4×*n*) (INTC\_LVL*n*IACK)

	7	6	5	4	3	2	1	0
R	VECN							
W								
SWIACK Reset	0	0	0	0	0	0	0	0
LVL <i>n</i> IACK Reset	0	0	0	1	1	0	0	0

**Table 8-12. Software and Level-*n* IACK Registers (INTC\_SWIACK, INTC\_LVL*n*IACK)**

**Table 8-13. INTC\_SWIACK, INTC\_LVLnIACK Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6–0 VECN	<p>Vector number. Indicates the appropriate vector number.</p> <p>For the SWIACK register, it is the highest-level, highest-priority request currently being asserted in the CF1_INTC module. If there are no pending requests, VECN is zero.</p> <p>For the LVLnIACK register, it is the highest priority request within the specified level-<i>n</i>. If there are no pending requests within the level, VECN is 0x18 (24) to signal a spurious interrupt.</p>

### 8.3.10 Interrupt Request Level and Priority Assignments

This section provides multiple views of the interrupt request assignment: a two-dimensional view of levels and priorities within the level (Table 8-15) and a tabular representation based on request priority (Table 8-16).

The CF1\_INTC module implements a sparsely-populated 7 × 9 matrix of levels (7) and priorities within each level (9). In this representation, the leftmost top cell (level 7, priority 7) is the highest interrupt request while the rightmost lowest cell (level 1, priority 0) is the lowest interrupt request. The following legend is used for this table:

**Table 8-14. Legend for Table 8-15**

Interrupt Request Source	
Interrupt Source Number	Vector Number

#### NOTE

For remapped and forced interrupts, the interrupt source number entry indicates the register or register field that enables the corresponding interrupt.

**Table 8-15. [Level][Priority with Level] Matrix Interrupt Assignments**

Level	Priority within Level													
	7	6		5		4	Midpoint	3		2	1	0		
7	—	—		—		—	IRQ_pin	low_voltage		—	—	force_lvl7		
							0   64	1   65				FRC[56]	103	
6	remapped	remapped		DMA_ch0		DMA_ch1	—	DMA_ch2		DMA_ch3		iEvent_ch0		force_lvl6
	PL6P7 *	PL6P6 *	4   68	5   69			6   70	7   71	8   72	FRC[57]	104			
5	FTM1 – (fault + ovfl) <sup>1</sup>		FTM1_ch0		FTM1_ch1		FTM1_ch2	—	FTM1_ch3		FTM1_ch4		FTM1_ch5	force_lvl5
	9   73	10   74	11   75	12   76			13   77	14   78	15   79	FRC[58]	105			

**Table 8-15. [Level][Priority with Level] Matrix Interrupt Assignments (continued)**

Level	Priority within Level																
	7		6		5		4		Midpoint	3		2		1		0	
4	FTM2 – (fault + ovfl) <sup>1</sup>		FTM2_ch0		FTM2_ch1		FTM2_ch2		—	FTM2_ch3		FTM2_ch4		FTM2_ch5		force_lvl4	
	16	80	17	81	18	82	19	83		20	84	21	85	22	86	FRC[59]	106
3	TPM3_ovfl		TPM3_ch0		TPM3_ch1		ADC		—	HSCMP1		HSCMP2		iEvent_ch1		force_lvl3	
	23	87	24	88	25	89	26	90		27	91	28	92	29	93	FRC[60]	107
2	SPI1		SPI2		SCI1_err		SCI1_rx		—	SCI1_tx		IIC		iEvent_ch2		force_lvl2	
	30	94	31	95	32	96	33	97		34	98	35	99	36	100	FRC[61]	108
1	SCI2_err		SCI2_rx		SCI2_tx		KBI[A:E] <sup>2</sup>		—	KBI[F:J] <sup>3</sup>		RTC + WDOG <sup>4</sup>		iEvent_ch3		force_lvl1	
	37	101	38	102	39	110	40	111		41	112	42	113	43	114	FRC[62]	109

<sup>1</sup> The fault and overflow requests from FTM{1,2} are logically summed per module and share a common interrupt vector.  
<sup>2</sup> The keyboard requests from ports [A-E] are logically summed together and share a common interrupt vector.  
<sup>3</sup> The keyboard requests from ports [F,G,I,J] are logically summed together and share a common interrupt vector.  
<sup>4</sup> The requests from the real-time clock and the watchdog are logically summed together and share a common interrupt vector.

Table 8-16 presents the same information on interrupt request assignments, but from the highest priority request to the lowest.

**Table 8-16. Interrupt Assignments**

IRQ Source	Level	Priority within Level	Interrupt Source Number	Vector
IRQ_pin	7	mid	0	64
low_voltage	7	3	1	65
reserved	7	2	2	66
reserved	7	1	3	67
force_lvl7	7	0	INTC_FRC[56]	103
remapped_l6p7	6	7	INTC_PL6P7	*
remapped_l6p6	6	6	INTC_PL6P6	*
DMA_ch0	6	5	4	68
DMA_ch1	6	4	5	69
DMA_ch2	6	3	6	70
DMA_ch3	6	2	7	71
iEvent_ch0	6	1	8	72
force_lvl6	6	0	INTC_FRC[57]	104
FTM1–(fault + ovfl)	5	7	9	73



Table 8-16. Interrupt Assignments (continued)

IRQ Source	Level	Priority within Level	Interrupt Source Number	Vector
FTM1_ch0	5	6	10	74
FTM1_ch1	5	5	11	75
FTM1_ch2	5	4	12	76
FTM1_ch3	5	3	13	77
FTM1_ch4	5	2	14	78
FTM1_ch5	5	1	15	79
force_lvl5	5	0	INTC_FRC[58]	105
FTM2-(fault + ovfl)	4	7	16	80
FTM2_ch0	4	6	17	81
FTM2_ch1	4	5	18	82
FTM2_ch2	4	4	19	83
FTM2_ch3	4	3	20	84
FTM2_ch4	4	2	21	85
FTM2_ch5	4	1	22	86
force_lvl4	4	0	INTC_FRC[59]	106
TPM3_ovfl	3	7	23	87
TPM3_ch0	3	6	24	88
TPM3_ch1	3	5	25	89
ADC	3	4	26	90
HSCMP1	3	3	27	91
HSCMP2	3	2	28	92
iEvent_ch1	3	1	29	93
force_lvl3	3	0	INTC_FRC[60]	107
SPI1	3	7	30	94
SPI2	3	6	31	95
SCI1_err	3	5	32	96
SCI1_rx	3	4	33	97
SCI1_tx	3	3	34	98
IIC	3	2	35	99
iEvent_ch2	3	1	36	100
force_lvl2	2	0	INTC_FRC[61]	108

**Table 8-16. Interrupt Assignments (continued)**

IRQ Source	Level	Priority within Level	Interrupt Source Number	Vector
SCI2_err	1	7	37	101
SCI2_rx	1	6	38	102
SCI2_tx	1	5	39	110
KBI[A:E]	1	4	40	111
KBI[F:J]	1	3	41	112
RTC + WDOG	1	2	42	113
iEvent_ch3	1	1	43	114
force_lvl1	1	0	INTC_FRC[62]	109

## 8.4 Functional Description

The basic operation of the CF1\_INTC is detailed in the preceding sections. This section describes special rules applicable to non-maskable level seven interrupt requests and the module's interfaces.

### 8.4.1 Handling of Non-Maskable Level 7 Interrupt Requests

In this context of this discussion, the non-maskable level 7 interrupt requests refer only to the masking capability provided by the processor's SR[I] field. The ability to mask individual interrupt requests using the interrupt controller's IMR is always available, regardless of the level of a particular interrupt request.

The CPU treats level seven interrupts as non-maskable, edge-sensitive requests, while levels one through six are maskable, level-sensitive requests. As a result of this definition, level seven interrupt requests are a special case. The edge-sensitive nature of these requests means the encoded 3-bit level input from the CF1\_INTC to the V1 ColdFire core must change state before the CPU detects an interrupt. A non-maskable interrupt (NMI) is generated each time the encoded interrupt level changes to level seven (regardless of the SR[I] field) and each time the SR[I] mask changes from seven to a lower value while the encoded request level remains at seven.

## 8.5 Initialization Information

The reset state of the CF1\_INTC module enables the default IRQ mappings and clears any software-forced interrupt requests (INTC\_FRC is cleared). Immediately after reset, the CF1\_INTC begins its cycle-by-cycle evaluation of any asserted interrupt requests and forms the appropriate encoded interrupt level and vector information for the V1 Coldfire processor core. The ability to mask individual interrupt requests using the interrupt controller's IMR is always available, regardless of the level of a particular interrupt request.

## 8.6 Application Information

This section discusses three application topics: emulation of the HCS08's one level interrupt nesting structure, elevating the priority of two IRQs, and more details on the operation of the software interrupt acknowledge (SWIACK) mechanism.

### 8.6.1 Emulation of the HCS08's 1-Level IRQ Handling

As noted in [Table 8-1](#), the HCS08 architecture specifies a 1-level IRQ nesting capability. Interrupt masking is controlled by CCR[I], the interrupt mask flag: clearing CCR[I] enables interrupts, while setting CCR[I] disables interrupts. The ColdFire architecture defines seven interrupt levels, controlled by the 3-bit interrupt priority mask field in the status register, SR[I], and the hardware automatically supports nesting of interrupts.

To emulate the HCS08's 1-level IRQ capabilities on V1 ColdFire, only two SR[I] settings are used:

- Writing 0 to SR[I] enables interrupts.
- Writing 7 to SR[I] disables interrupts.

The ColdFire core treats the level seven requests as non-maskable, edge-sensitive interrupts.

ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register as the first instruction in the ISR. In addition, the V1 instruction set architecture (ISA\_C) 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 the *ColdFire Family Programmer's Reference Manual*. A MOVE-to-SR instruction also performs a similar function.

To emulate the HCS08's 1-level IRQ nesting mechanisms, the ColdFire implementation enables interrupts by clearing SR[I] (typically when using RTE to return to a process) and disables interrupts upon entering every interrupt service routine by one of three methods:

1. Execution of STLDSR #0x2700 as the first instruction of an ISR.
2. Execution of MOVE.w #0x2700,SR as the first instruction of an ISR.
3. Static assertion of CPUCR[IME] that forces the processor to load SR[I] with seven automatically upon the occurrence of an interrupt exception. Because this method removes the need to execute multi-cycle instructions of #1 or #2, this approach improves system performance.

### 8.6.2 Using INTC\_PL6P{7,6} Registers

[Section 8.3.3, "INTC Programmable Level 6, Priority {7,6} Registers \(INTC\\_PL6P{7,6}\),"](#) describes control registers that provide the ability to dynamically alter the request level and priority of two IRQs. Specifically, these registers provide the ability to reassign two IRQs to be the highest level 6 (maskable) requests. Consider the following example.

Suppose the system operation desires to remap the receive and transmit interrupt requests of a serial communication device (SC11) as the highest two maskable interrupts. The default assignments for the SC11 transmit and receive interrupts are:

- sci1\_rx = interrupt source 33 = vector 97 = level 2, priority 4
- sci1\_tx = interrupt source 34 = vector 98 = level 2, priority 3

To remap these two requests, the INTC\_PL6P{7,6} registers are programmed with the desired interrupt source number:

- Setting INTC\_PL6P7 to 33 (0x21), remaps sci1\_rx as level 6, priority 7.
- Setting INTC\_PL6P6 to 34 (0x22), remaps sci1\_tx as level 6, priority 6.

The reset state of the INTC\_PL6P{7,6} registers disables any request remapping.

### 8.6.3 More on Software IACKs

As previously mentioned, the notion of a software IACK refers to the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been cleared) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can improve overall system performance noticeably.

To illustrate this concept, consider the following ISR code snippet shown in [Figure 8-11](#).

```

                align    4
                irqxx_entry:
00588: 4fef fff0 lea    -16(sp),sp           # allocate stack space
0058c: 48d7 0303 movem.l #0x0303,(sp)      # save d0/d1/a0/a1 on stack

                irqxx_alterate_entry:
00590:
        ....
                irqxx_swiack:
005c0: 71b8 ffe0 mvz.b  INTC_SWIACK.w,d0    # perform software IACK
005c4: 0c00 0041 cmpi.b #0x41,d0           # pending IRQ or level 7?
005c8: 6f0a      ble.b  irqxx_exit         # no pending IRQ, then exit
005ca: 91c8      sub.l  a0,a0             # clear a0
005cc: 2270 0c00 move.l 0(a0,d0.l*4),a1    # fetch pointer from xcpt table
005d0: 4ee9 0008 jmp    8(a1)             # goto alternate isr entry point

                align    4
                irqxx_exit:
005d4: 4cd7 0303 movem.l (sp),#0x0303    # restore d0/d1/a0/a1
005d8: 4fef 0010 lea    16(sp),sp         # deallocate stack space
005dc: 4e73      rte                    # return from handler

```

**Figure 8-11. ISR Code Snippet with SWIACK**

This snippet includes the prologue and epilogue for an interrupt service routine as well as code needed to perform software IACK.

At the entry point (`irqxx_entry`), there is a two-instruction prologue to allocate space on the supervisor stack to save the four volatile registers (`d0`, `d1`, `a0`, `a1`) defined in the ColdFire application binary interface. After saving these registers, the ISR continues at the alternate entry point.

The software IACK is performed near the end of the ISR, after the source of the current interrupt request is negated. First, the appropriate memory-mapped byte location in the interrupt controller is read (`PC = 0x5C0`). The `CF1_INTC` module returns the vector number of the highest priority pending request. If no request is pending, zero is returned. The compare instruction is needed to manage a special case involving pending level seven requests. Because the level seven requests are non-maskable, the ISR is interrupted to service one of these requests. To avoid any race conditions, this check ignores the level seven vector numbers. The result is the conditional branch (`PC = 0x5C8`) is taken if there are no pending requests or if the pending request is a level seven.

If there is a pending non-level seven request, execution continues with a three instruction sequence to calculate and then branch to the appropriate alternate ISR entry point. This sequence assumes the exception vector table is based at address `0x(00)00_0000` and that each ISR uses the same two-instruction prologue shown here. The resulting alternate entry point is a fixed offset (8 bytes) from the normal entry point defined in the exception vector table.

The ISR epilogue includes a three instruction sequence to restore the volatile registers from the stack and return from the interrupt exception.

This example is intentionally simple, but does show how performing the software IACK and passing control to an alternate entry point when there is a pending but masked interrupt request can avoid the execution of the ISR epilogue, another interrupt exception, and the ISR prologue.

# Chapter 9

## High-Speed Comparator (S08HSCMPV2)

### 9.1 Introduction

The high speed comparator module (HSCMP) provides a circuit for comparing two analog input voltages or one analog input voltage to an internal reference voltage. The comparator circuit is designed to operate across the full range of the supply voltage (rail-to-rail operation).

### 9.2 Low Power Mode Operation

The HSCMP module is capable of functioning in wait, stop3, and stop4 modes of operation. The output of HSCMP is available to wake the MCU from stop3 and stop4 modes.

For details on low-power mode operation, refer to [Table 3-1](#) in [Chapter 3](#), “Modes of Operation.”

### 9.3 HSCMP Configuration Information

#### 9.3.1 HSCMP Analog Input Configuration

Each of the two HSCMP blocks has eight analog inputs. Out of which four are connected to a plus side multiplexor (inputs P0 – P3) and other four are connected to a minus side multiplexor (inputs M0 – M3). [Table 9-1](#) summarizes the analog comparator input connections.

**Table 9-1. HSCMP Analog Input Connections**

Type	Analog source	HSCMP input signal
External	PTA2/CIN1	CIN1 is connected to P0 and M0 of HSCMP1 and HSCMP2
External	PTD0/C1IN2	C1IN2 is connected to P1 and M1 of HSCMP1
External	PTD1/C1IN3	C1IN3 is connected to P2 and M2 of HSCMP1
External	PTA4/C2IN2	C2IN2 is connected to P1 and M1 of HSCMP2
External	PTA5/C2IN3	C2IN3 is connected to P2 and M2 of HSCMP2
Internal	DAC1	DAC1 out is connected to P3 and M3 of HSCMP1.
Internal	DAC2	DAC2 out is connected to P3 and M3 of HSCMP2.

### 9.3.2 TPM/HSCMP Configuration Information

The HSCMP2 module can be configured to connect its output to TPM3 input capture channel 0 by setting the SOPT2[ACIC] bit (see [Section 5.8.8, “System Options 2 \(SOPT2\) Register”](#)). With ACIC set, the TPM3CH0 pin is not available externally regardless of the configuration of the TPM3 module.

### 9.3.3 HSCMP Clock Gating

The bus clock to the HSCMP can be gated on and off using the SCGC3[HSCMPx] bit (see [Section 5.8.11, “System Clock Gating Control 3 Register \(SCGC3\)”](#)). This bit is cleared after any reset that disables the bus clock to this module. The SCGC3[HSCMPx] bit must be set before operation. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

## 9.4 Features

The HSCMP has the following features:

- Operates with over the entire supply range
- Inputs may range from rail to rail
- Less than 40 mV of input offset
- Less than 15 mV of hysteresis
- Selectable interrupt on rising edge, falling edge, or either rising or falling edges of comparator output
- Selectable inversion on comparator output
- Comparator output may be:
  - Sampled
  - Windowed (ideal for certain PWM zero-crossing-detection applications)
  - Digitally Filtered
    - Filter can be bypassed
    - May be clocked via external SAMPLE signal or scaled peripheral clock
- External hysteresis can be used at the same time that the output filter is used for internal functions
- The positive and negative inputs of the comparator are both driven from 4-to-1 muxes that allow additional flexibility in assigning IO as comparator inputs during PCB design
- Two software selectable performance levels:
  - Shorter propagation delay at the expense of higher power. This mode can be used only when the  $V_{DDA}$  rail is above the low voltage interrupt trip point.
  - Low power, with longer propagation delay
- Supports DMA transfer:
  - Changes on COUT can be selected to trigger a DMA event

## 9.5 Block Diagram

The block diagram for the High Speed Comparator module is shown in [Figure 9-1](#).



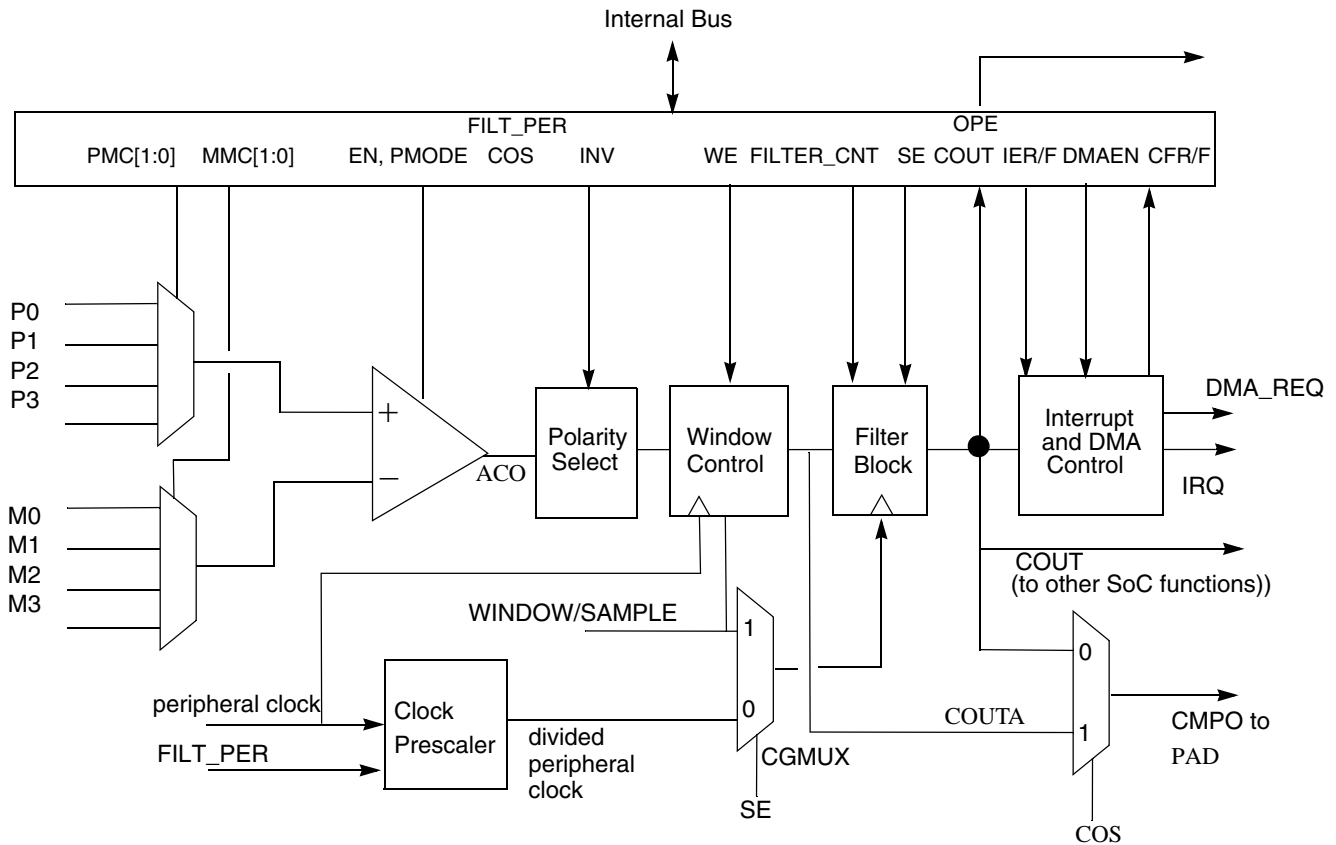


Figure 9-1. High Speed Comparator Module Block Diagram

In Figure 9-1:

- The window control block is completely bypassed when WE = 0.
- If WE = 1, the comparator output is sampled on every peripheral clock when WINDOW = 1 to generate COUTA. Sampling does not occur when WINDOW = 0.
- The filter block is bypassed when not in use.

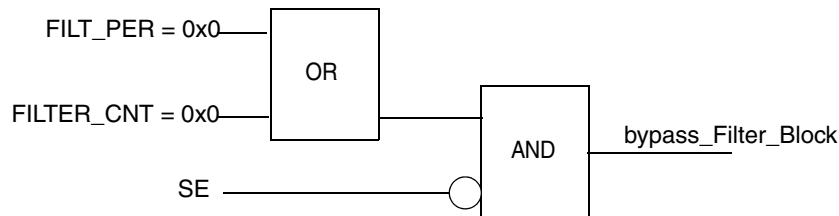


Figure 9-2. Filter Block Bypass Logic

- The filter block acts as a simple sampler if  $\overline{\text{bypass\_Filter\_Block}} \ \&\& \ \text{FILTER\_CNT} = 0x1$ .
- The filter block filters based on multiple samples when  $\overline{\text{bypass\_Filter\_Block}} \ \&\& \ \text{FILTER\_CNT} > 0x1$ 
  - If SE = 1, the external SAMPLE input is used as sampling clock.
  - If SE = 0, the divided peripheral clock is used as sampling clock.

- If enabled, the filter block incurs up to 1 IP Bus additional latency penalty on COUT due to the fact that COUT (which is crossing clock domain boundaries) must be resynchronized to the peripheral clock.
- WE and SE are mutually exclusive.

## 9.6 Pin Descriptions

### 9.6.1 External Pins

The HSCMP has up to eight external analog input pins and one external output pin. Each input pin can accept an input voltage that varies across the full operating voltage range of the MCU. If the module is not enabled, these pins can be used as digital inputs or outputs. Unused inputs can also be used as digital inputs or outputs. Comparator input pins that are not associated with specific SoC IO pads are normally tied to  $V_{SSA}$ . You are responsible for ensuring that both comparator inputs are not simultaneously tied to  $V_{SSA}$ .

Consult the specific MCU documentation to determine what functions are shared with analog inputs. As shown in the block diagram, the  $P_n$  pins are connected to the comparator non-inverting input.  $M_n$  pins are connected to the inverting input of the comparator.

You can select filtered or unfiltered comparator outputs for use on an external IO pad.

Operation of HSCMPxCR1[OPE] varies from chip to chip. Simplified examples are shown in [Figure 9-3](#).

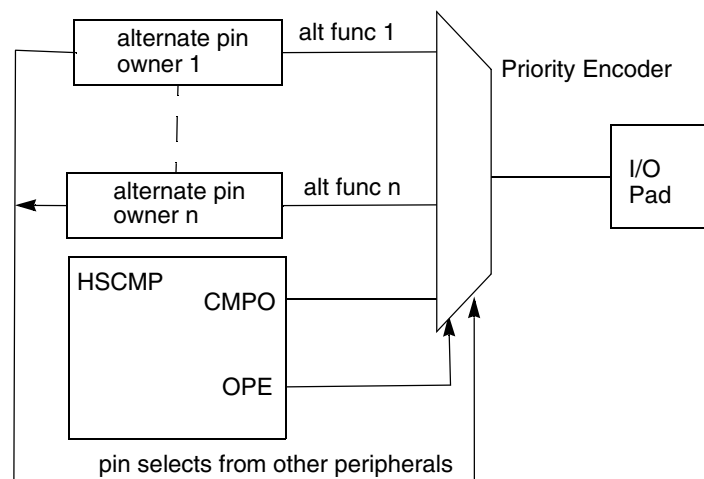


Figure 9-3. OPE Operation on S08 and Flexis Devices

## 9.7 Functional Description

The HSCMP can be used to compare two analog input voltages applied to  $P_n$  and  $M_n$ . The analog comparator output (ACO) is high when the non-inverting input is greater than the inverting input, and is low when the non-inverting input is less than the inverting input. This signal can be selectively inverted by setting HSCMPxCR1[INV] = 1.

The HSCMPxSCR[IER] and HSCMPxSCR[IEF] bits select the condition that causes the comparator module to assert an interrupt to the processor. HSCMPxSCR[CFR] bit is set on a rising edge of the comparator output. HSCMPxSCR[CFF] is set on a falling edge of the comparator output. The (optionally filtered) comparator output can be read directly through the HSCMPxSCR[COU] bit.

## 9.7.1 HSCMP Functional Modes

There are three main sub-blocks to the comparator module: the comparator itself, the window function, and the filter function. The filter can be clocked from an internally generated clock, or via an external sample input. Additionally, the filter is programmable with respect to how many samples must agree before a change on the output is registered. In the simplest case, only 1 sample must agree. In this case, the filter acts as a simple sampler.

The external sample input is enabled using HSCMPxCR1[SE]. When set, the output of the comparator is sampled only on rising edges of the sample input.

The windowing mode is enabled by setting HSCMPxCR1[WE]. When set, the comparator output is sampled only when the WINDOW input signal is equal to one. This feature can be used to ignore the comparator output during time periods in which the input voltages are not valid. This is especially useful when implementing zero-crossing-detection for certain PWM applications.

The comparator filter and sampling features can be combined as shown in [Table 9-2](#). Individual modes are discussed below.

**Table 9-2. Comparator Sample/Filter Controls**

Mode #	EN	WE	SE	FILT_CNT	FILT_PER	Operation
1	0	X	X	X	X	<b>Disabled</b> Section 9.7.1.1, "Disabled Mode (# 1)"
2A	1	0	0	0x0	X	<b>Continuous Mode</b> Section 9.7.1.2, "Continuous Mode (#s 2A & 2B)"
2B	1	0	0	X	0x00	
3A	1	0	1	0x1	X	<b>Sampled, Non-Filtered mode</b> Section 9.7.1.3, "Sampled, Non-Filtered Mode (#s 3A & 3B)"
3B	1	0	0	0x1	> 0x0	
4A	1	0	1	> 0x1	X	<b>Sampled, Filtered mode</b> Section 9.7.1.4, "Sampled, Filtered Mode (#s 4A & 4B)"
4B	1	0	0	> 0x1	> 0x0	
5A	1	1	0	0x0	X	<b>Windowed mode</b> Comparator output is sampled on every rising peripheral clock edge when SAMPLE = 1 to generate COUTA Section 9.7.1.5, "Windowed Mode (#s 5A & 5B)"
5B	1	1	0	X	0x00	
6	1	1	0	0x1	0x01 – 0xFF	<b>Windowed/Resampled mode</b> Comparator output is sampled on every rising peripheral clock edge when SAMPLE = 1 to generate COUTA that is then resampled on an interval determined by FILT_PER to generate COUT. Section 9.7.1.6, "Windowed/Resampled Mode (# 6)"

Table 9-2. Comparator Sample/Filter Controls (continued)

Mode #	EN	WE	SE	FILT_CNT	FILT_PER	Operation
7	1	1	0	> 0x1	0x01– 0xFF	<p><b>Windowed/Filtered mode</b></p> <p>Comparator output is sampled on every rising peripheral clock edge when SAMPLE = 1 to generate COUTA that is then resampled and filtered to generate COUT.</p> <p><a href="#">Section 9.7.1.7, “Windowed/Filtered Mode (#7)”</a></p>
All other combinations of EN, WE, SE, FILT_CNT and FILT_PER are illegal.						

For cases where a comparator drives a fault input of the PWM, it must generally be configured to operate in continuous mode, so that an external fault can immediately pass through the comparator to the PWM fault circuitry.

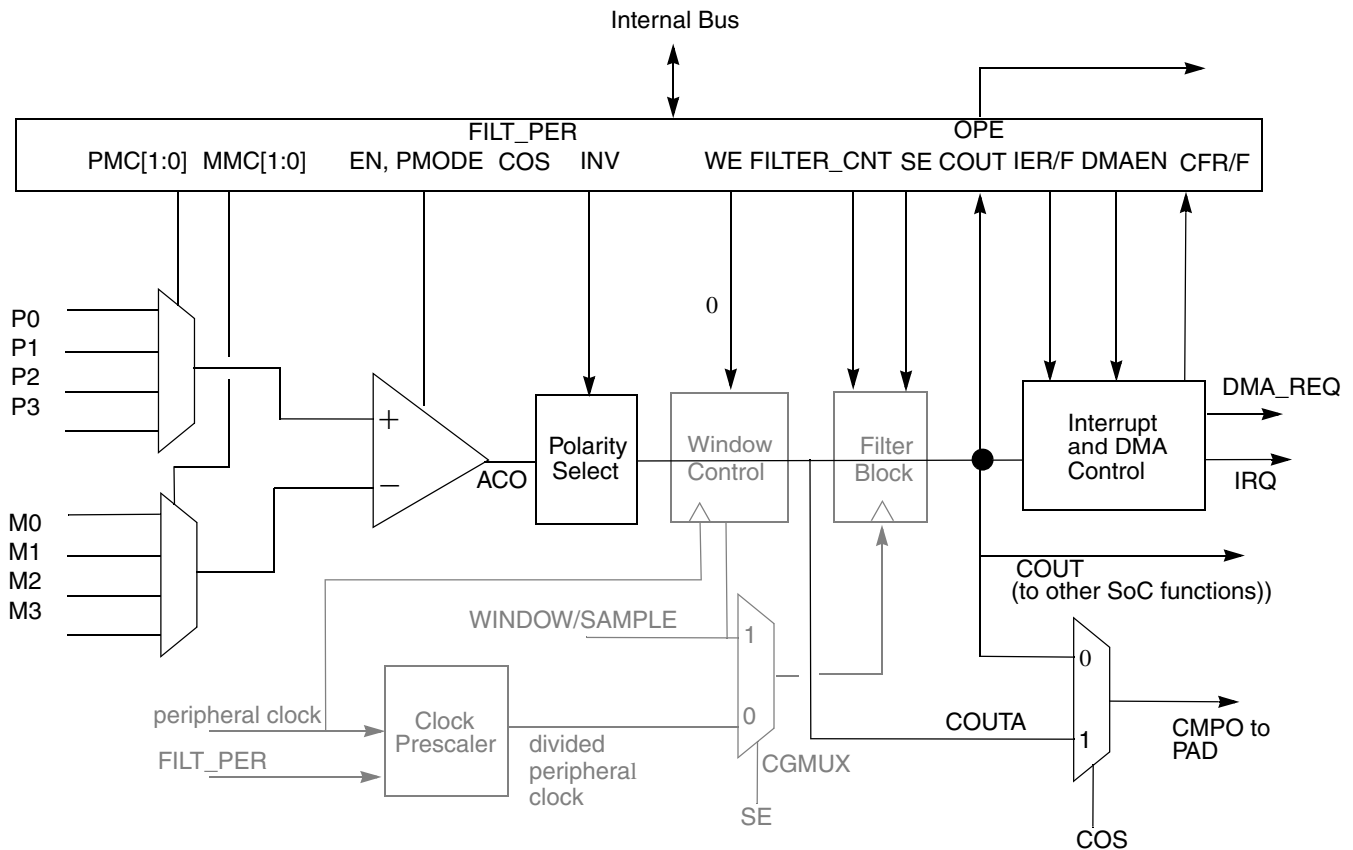
### CAUTION

Filtering and sampling settings must be changed only after setting SE = 0 and FILTER\_CNT = 0x0. This has the effect of resetting the filter to a known state.

#### 9.7.1.1 Disabled Mode (# 1)

In disabled mode, the analog comparator is non-functional and consumes no power. The output of the analog comparator block (ACO) is zero in this mode. ColdFire V1 and DSC devices can further reduce power consumed in the digital block by disabling the peripheral clock to the comparator logic. This is usually a function of the System Integration Module (SIM).

### 9.7.1.2 Continuous Mode (#s 2A & 2B)

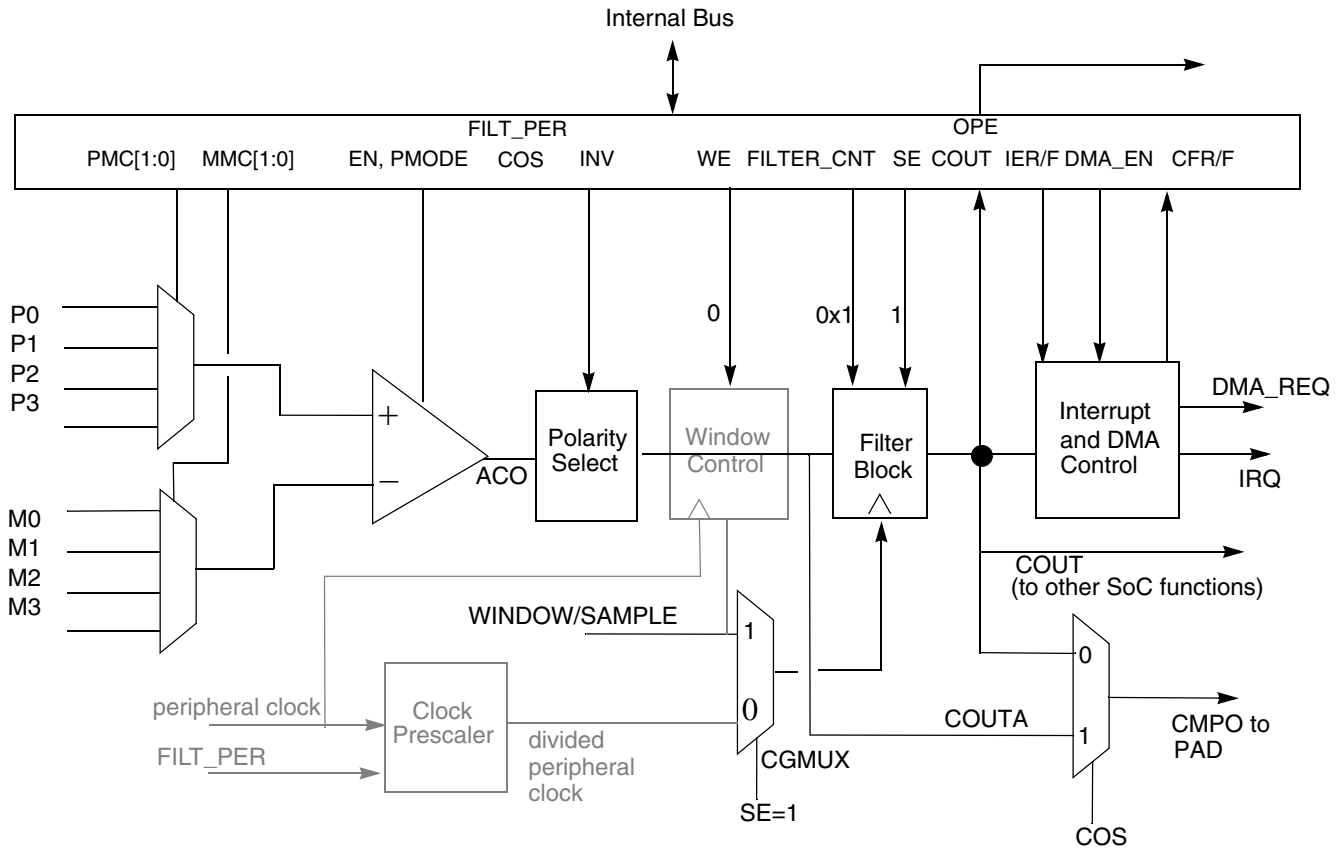


**Figure 9-4. Comparator Operation in Continuous Mode**

The analog comparator block is powered and active. `ACO` may be optionally inverted, but is not subject to external sampling or filtering. Window Control and Filter Blocks are completely bypassed. `HSCMPxSCR[COUT]` is updated continuously. The path from comparator inputs pins to output pin is operating in combinational (unlocked) mode. `COUT` and `COUTA` are identical.

See [Figure 9-2](#) for control configurations that result in disabling the Filter Block.

### 9.7.1.3 Sampled, Non-Filtered Mode (#s 3A & 3B)



**Figure 9-5. Sampled, Non-Filtered (# 3A): Sampling point externally driven**

In Sampled, Non-Filtered mode, the analog comparator block is powered and active. The path from analog inputs to COUTA is combinational (unlocked). Windowing Control is completely bypassed. COUTA is sampled whenever a rising edge is detected on the Filter Block clock input.

The only difference in operation between [Figure 9-5](#) and [Figure 9-6](#) is in how the clock to the Filter Block is derived.

The comparator filter has no other function than sample/hold of the comparator output in this mode.

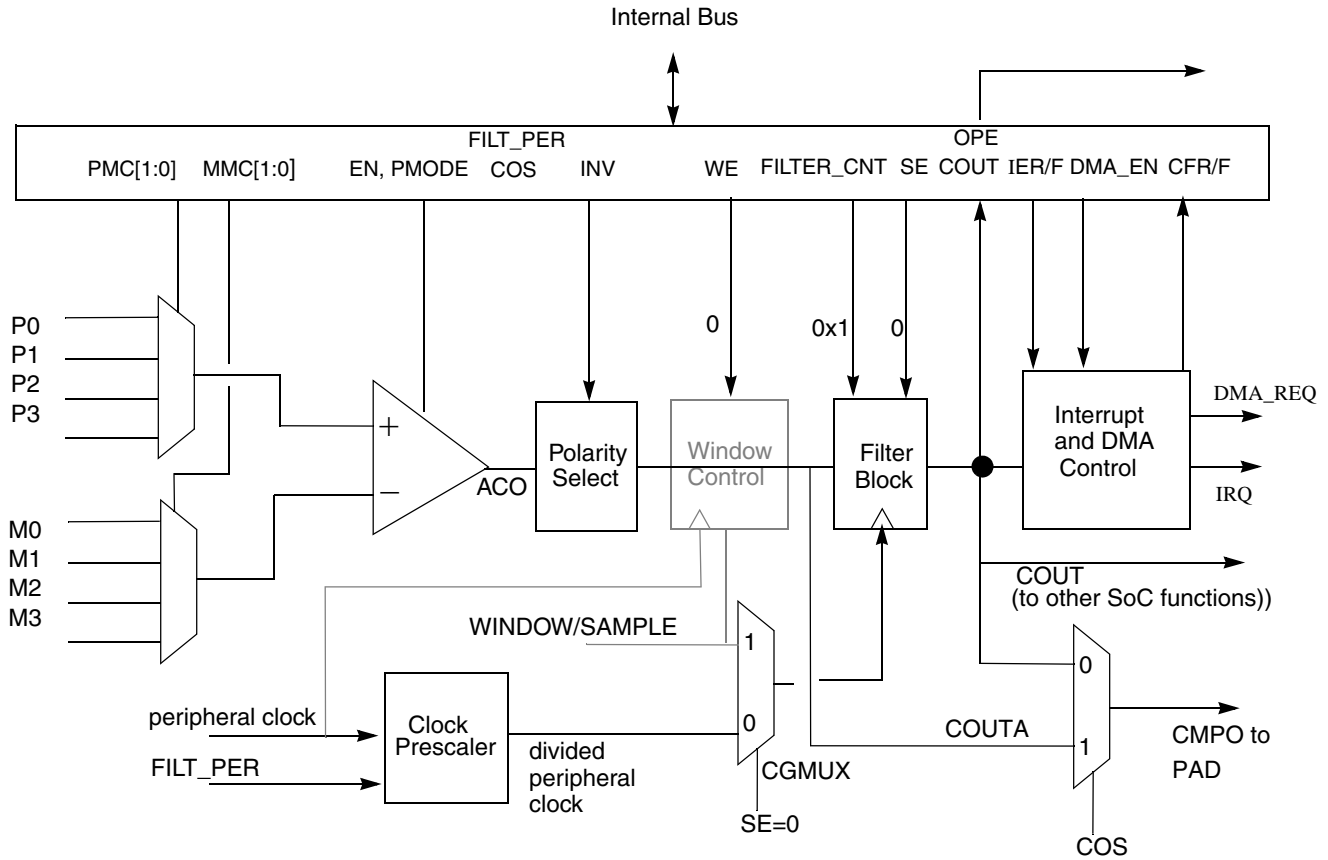


Figure 9-6. Sampled, Non-Filtered (# 3B): Sampling interval internally derived

#### 9.7.1.4 Sampled, Filtered Mode (#s 4A & 4B)

In Sampled, Filtered mode, the analog comparator block is powered and active. The path from analog inputs to COUTA is combinational (unlocked). Windowing Control is completely bypassed. COUTA is sampled whenever a rising edge is detected on the Filter Block clock input.

The only difference in operation between [Figure 9-5](#) (Sampled, Non-Filtered # 3A) and [Figure 9-8](#) (Sampled, Filtered # 4A) is that FILTER\_CNT is now greater than 1 that activates filter operation.

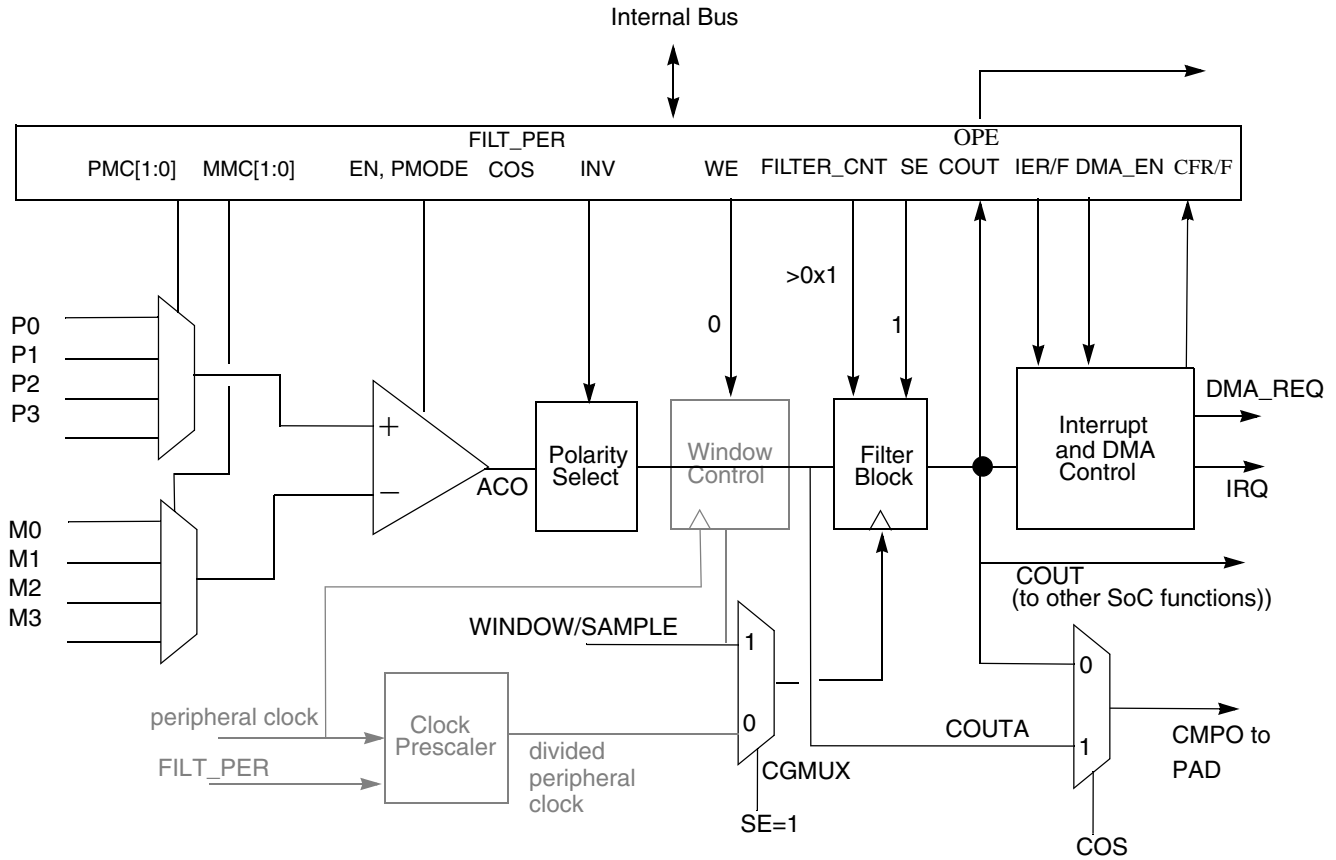
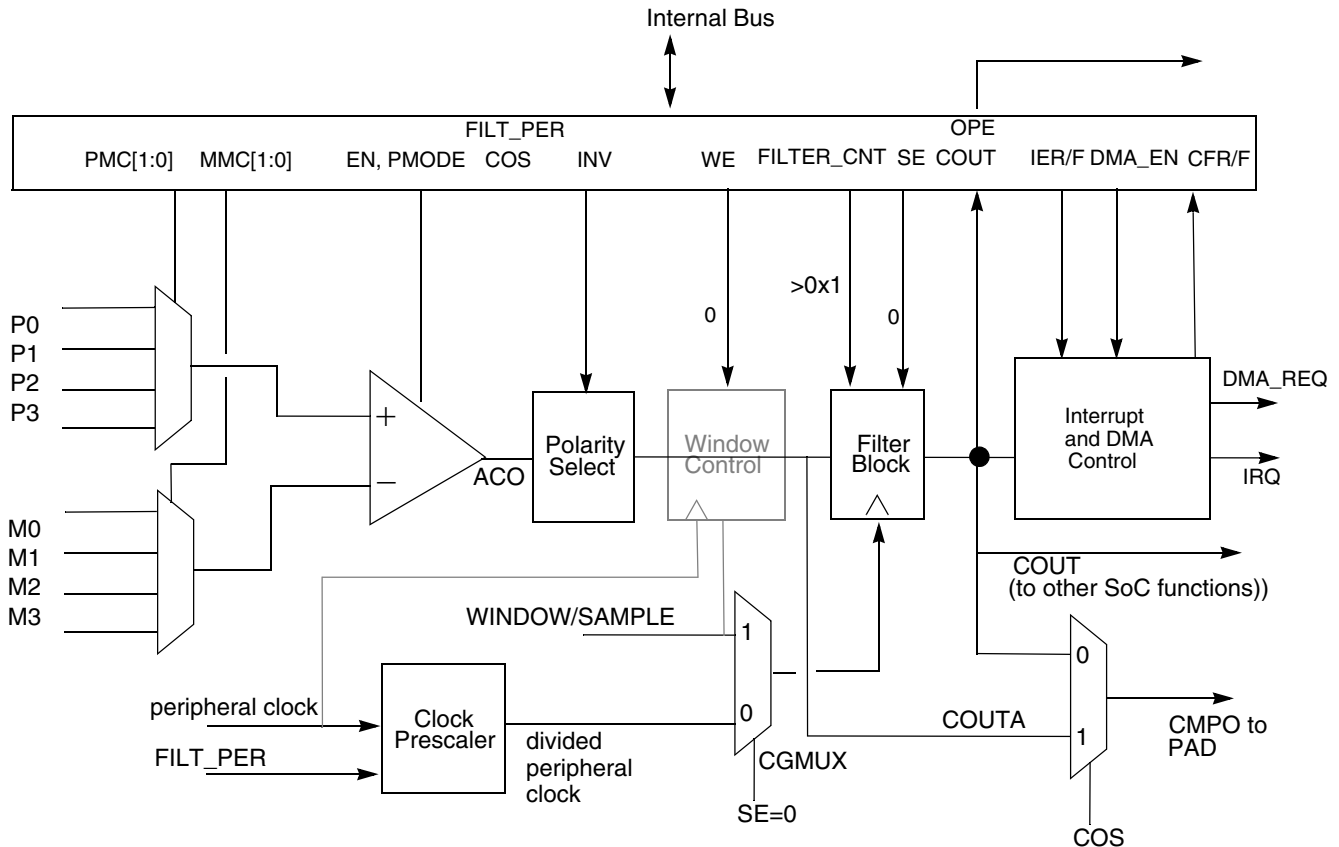


Figure 9-7. Sampled, Filtered (# 4A): Sampling point externally driven





**Figure 9-8. Sampled, Filtered (# 4B): Sampling point internally derived**

The only difference in operation between [Figure 9-6](#) (Sampled, Non-Filtered # 3B) and [Figure 9-8](#) (Sampled, Filtered # 4B) is that `FILTER_CNT` is now greater than 1 that activates filter operation.

### 9.7.1.5 Windowed Mode (#s 5A & 5B)

[Figure 9-9](#)<sup>1</sup> illustrates comparator operation in the windowed mode, ignoring latency of the analog comparator, Polarity Select, and Window Control block. It also assumes that the Polarity Select is set to non-inverting.

#### NOTE

The analog comparator output is passed to `COUTA` only when the `WINDOW` signal is high.

In actual operation, `COUTA` may lag the analog inputs by up to one peripheral clock cycle plus the combinational path delay through the comparator and polarity select logic.

1.  $P_x$  and  $M_y$  represent different external voltages on the positive and minus inputs respectively.

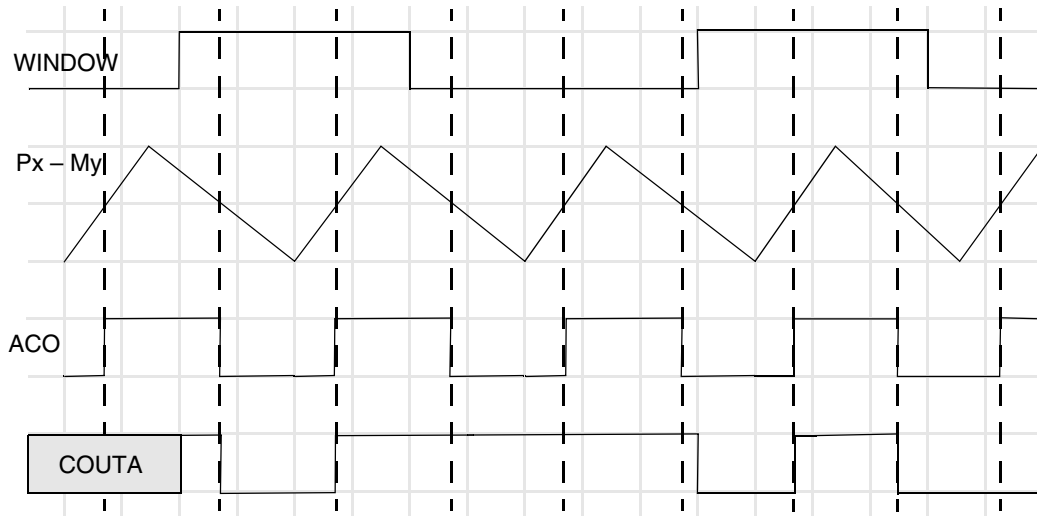


Figure 9-9. Windowed Mode Operation

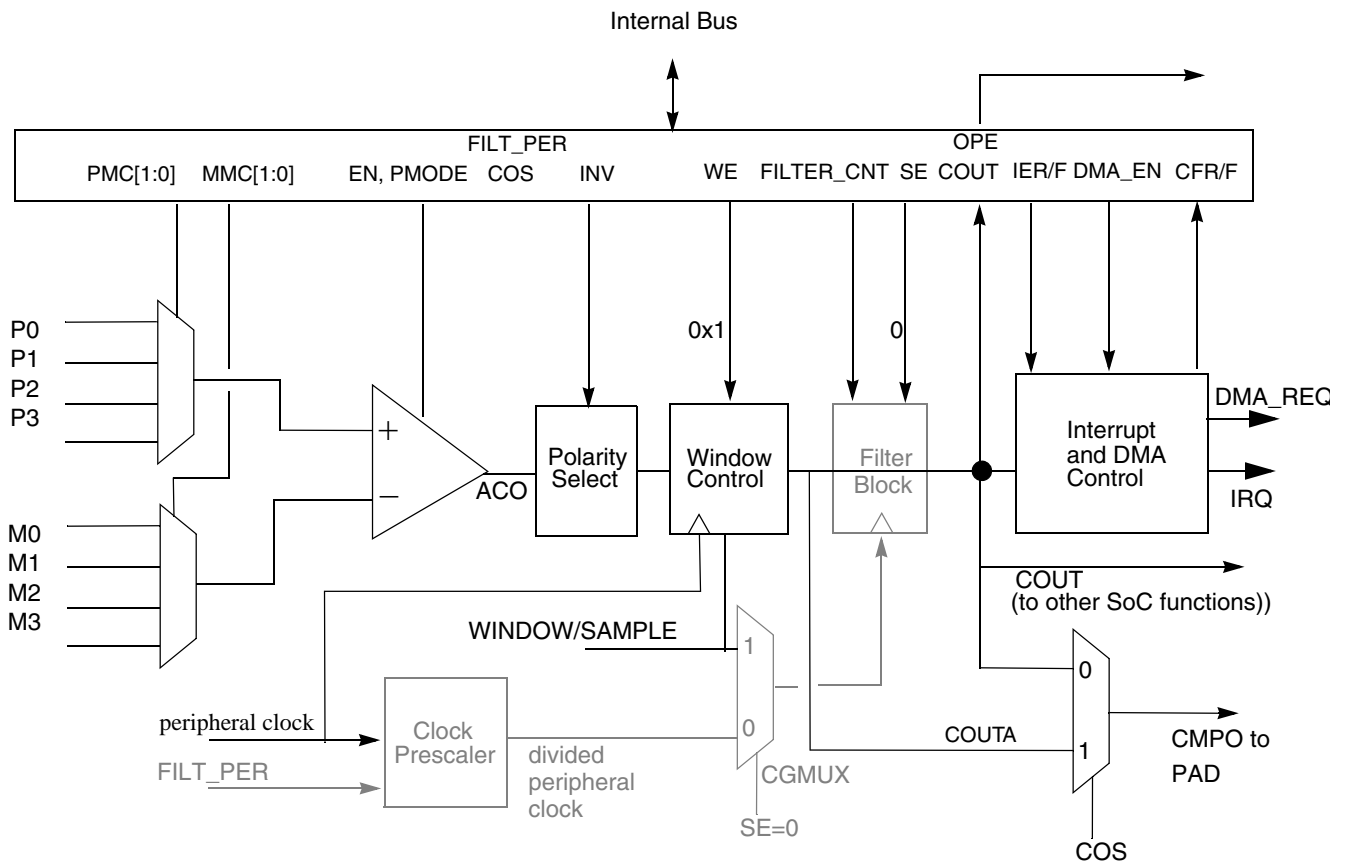


Figure 9-10. Windowed Mode

See Figure 9-2 for control configurations that result in disabling the Filter Block.

When any windowed mode is active, COUTA is clocked by the peripheral clock whenever WINDOW = 1. The last latched value is held when WINDOW = 0.

### 9.7.1.6 Windowed/Resampled Mode (# 6)

Figure 9-11<sup>1</sup> uses the same input stimulus shown in Figure 9-9, and adds resampling of COUTA to generate COUT. Samples are taken at the time points indicated by the arrows. Again, prop delays and latency is ignored for clarity's sake. This example was generated solely to demonstrate operation of the comparator in windowing/resampled mode, and does not reflect any specific application. Depending upon the sampling rate and window placement, COUT may not see zero-crossing events detected by the analog comparator. Sampling period and/or window placement must be carefully considered for a given application.

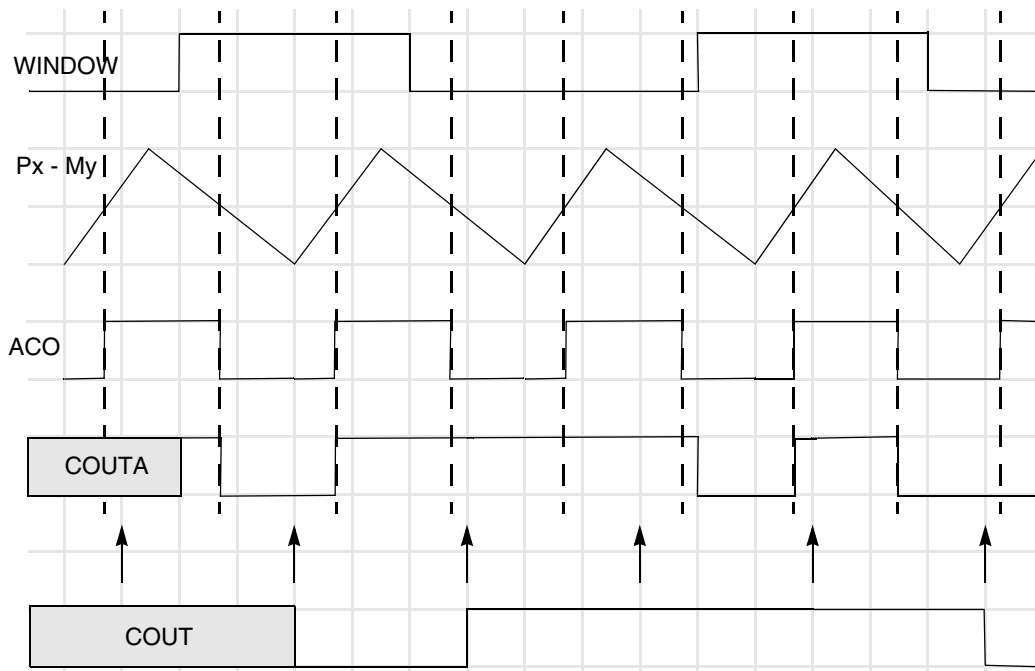


Figure 9-11. Windowed/Resampled Mode Operation

This mode of operation results in an unfiltered string of comparator samples where the interval between the samples is determined by `FILT_PER` and the peripheral clock rate. Configuration for this mode is virtually identical to that for the Windowed/Filtered mode shown in the next section. The only difference is that the value of `FILTER_CNT` must be exactly one.

### 9.7.1.7 Windowed/Filtered Mode (#7)

This is the most complex mode of operation for the comparator block as it utilizes windowing and filtering features. It also has the highest latency of any of the modes. This can be approximated: up to 1 peripheral clock synchronization in the window function +  $((\text{FILT\_CNT} \times \text{FILT\_PER}) + 1) \times$  peripheral clock for the filter function.

When any windowed mode is active, `COUTA` is clocked by the peripheral clock whenever `WINDOW = 1`. The last latched value is held when `WINDOW = 0`.

1. `Px` and `My` represent different external voltages on the positive and minus inputs respectively.

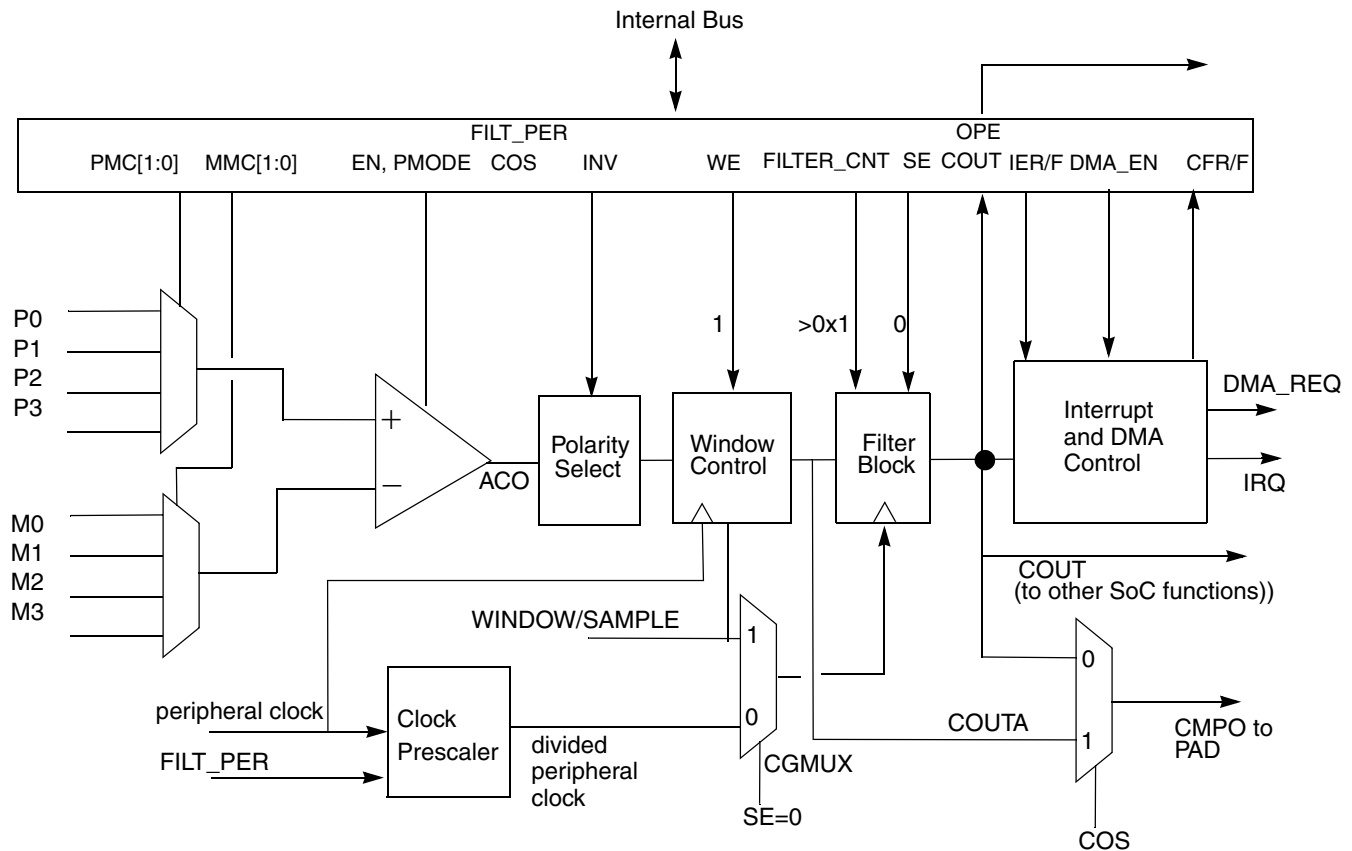


Figure 9-12. Windowed/Filtered Mode

## 9.7.2 Power Modes

This module is designed for compatibility with several Freescale processor families. Power modes are similar, although the nomenclature is somewhat different.

### 9.7.2.1 Wait Mode Operation

During wait and *lpwait* modes, the HSCMP if enabled, continues to operate normally. Also, if enabled, the interrupt can wake the MCU.

### 9.7.2.2 Stop Mode Operation

#### 9.7.2.2.1 Stop3 & Stop4 Mode Operation

Subject to platform-specific clock restrictions outlined below, the MCU is brought out of stop when a compare event occurs and the corresponding interrupt is enabled. Similarly, if HSCMPxCR1[OPE] is enabled, the comparator output operates as in the normal operating mode and comparator output is placed onto the external pin.

If stop is exited with a reset, all comparator registers are put into their reset state.

### For S08 and Flexis Devices:

Because there is no peripheral clock available during STOP modes, windowed, sampled, and filtered modes of operation are not available, and the HSCMP must not be left in these configurations during STOP.

However, the analog comparator within HSCMP can be left enabled during STOP. If HSCMPxSCR[SMLEB] is set to zero, compare events are converted to level sensitive operation. The MCU can then be brought out of stop when a compare event occurs and corresponding interrupt is enabled.

#### 9.7.2.2.2 Stop2 and Stop1 Mode Operation

During stop2 and stop1 mode, the HSCMP module is fully powered down. Upon wake from stop2 or stop1 mode, the HSCMP module is in the reset state.

#### 9.7.2.3 Background Mode Operation

When the microcontroller is in active background mode, the HSCMP continues to operate normally.

### 9.7.3 Hysteresis

Figure 9-13 illustrates implementation of an external hysteresis resistor bridge between the asynchronous comparator output and the positive (+) input of the comparator. Since positive feedback is required, INV must be set to 0 when the hysteresis resistor bridge is added to the positive (+) input of the comparator. INV must be set to 1 when the hysteresis resistor bridge is added to the negative (-) input of the comparator.

The option of adding an external resistor bridge for the purpose of adding hysteresis to the comparator and the amount of hysteresis depends on the user's individual requirements. Hysteresis can be important in some system designs. In the absence of hysteresis, the continuous comparison of nearly identical analog inputs may add noise and waste power by generating high-frequency oscillations at CMPO.

If external hysteresis is added to the comparator, the bridge must be designed to consider other issues than simply how much hysteresis is needed. The resistor values must be sufficiently high so that they do not cause the drive strength of the digital output driver in the CMPO IO cell to be exceeded. Also, if any digital function other than CMPO must operate on the CMPO pad in the presence of such a bridge, the resistor values must be sufficiently high so that the IO cell can function appropriately in its digital role.

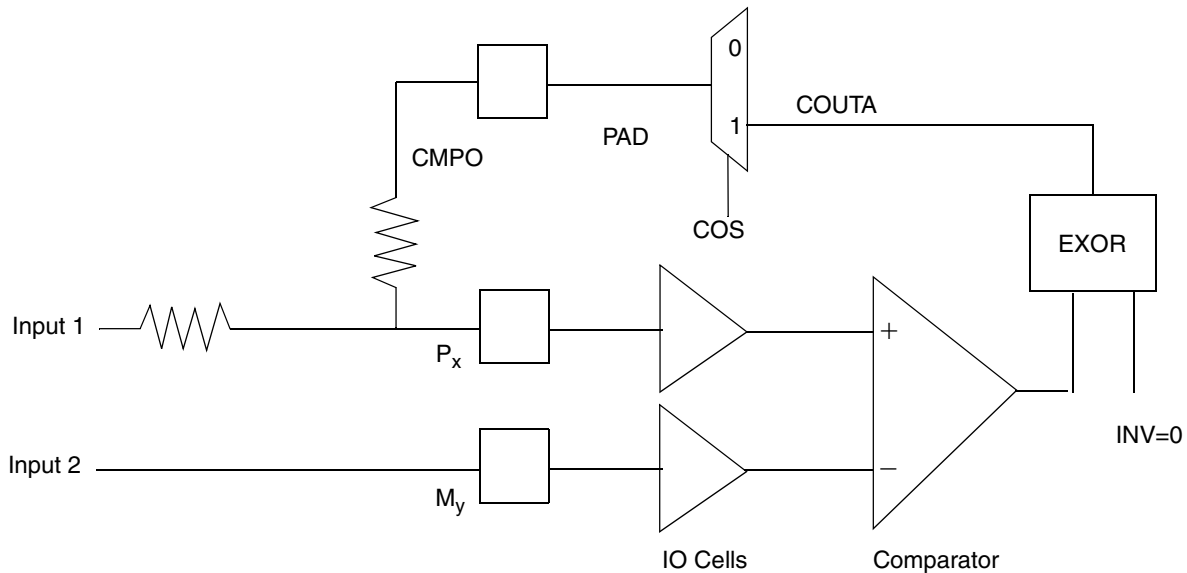


Figure 9-13. External Hysteresis Circuit

## 9.7.4 Startup and Operation

A typical startup sequence is as follows.

The time required to stabilize COUT is power-on delay of the comparators plus the largest propagation delay from a selected analog source through the analog comparator, windowing function, and filter. Power on delay of the comparators are available from the data sheets. The windowing function has a maximum of 1 peripheral bus clock period delay. Filter delay is specified in [Section 9.7.5, “Low Pass Filter.”](#)

During operation, the propagation delay of the selected data paths must always be considered. It can take many peripheral bus clock cycles for COUTA and the CFR/CFR status bits to reflect an input change or a configuration change to one of the components involved in the data path.

When programmed for filtering modes, COUTA is initially equal to zero until sufficient clock cycles have elapsed to fill all stages of the filter. This occurs even if COUTA is at a logic one.

## 9.7.5 Low Pass Filter

### 9.7.5.1 Introduction

The low-pass filter operates on the unfiltered and unsynchronized and optionally inverted comparator output COUTA and generates the filtered and synchronized output COUT. COUTA and COUT can be configured as module outputs and are used for different purposes within the system.

Synchronization and edge detection are always used to determine status register bit values. They also apply to COUT for all sampling and windowed modes. Filtering can be performed using an internal timebase defined by HSCMPxFPR[FILT\_PER], or using an external SAMPLE input to determine sample time.

The need for digital filtering and the amount of filtering is dependent on the user's individual requirements. Filtering can become more useful in the absence of an external hysteresis circuit. Without external hysteresis, high frequency oscillations can be generated at COUTA when the selected N and P input voltages differ by less than the offset voltage of the differential comparator.

### 9.7.5.2 Enabling Filter Modes

Filter modes are enabled by setting HSCMPxCR0[FILTER\_CNT] greater than 0x1 and (setting HSCMPxFPR[FILT\_PER] to a non-zero value or setting SE = 1). If using the divided peripheral clock to drive the filter, it takes samples of COUTA every FILT\_PER peripheral bus cycles.

The filter output is at logic zero when first initialized, and is subsequently changed when FILT\_CNT consecutive samples all agree that the output value has changed. Said another way, COUT is zero for some initial period, even when COUTA is at logic one.

Setting SE and FILT\_PER to 0 disables the filter and eliminates switching current associated with the filtering process.

#### NOTE

Always switch to this setting prior to making any changes in filter parameters. This resets the filter to a known state. Switching FILTER\_CNT on the fly without this intermediate step can result in unexpected behavior.

If SE = 1, the filter takes samples of COUTA on each positive transition of the SAMPLE input. The output state of the filter changes when FILT\_CNT consecutive samples all agree that the output value has changed.

### 9.7.5.3 Latency Issues

The FILT\_PER value (or SAMPLE period) must be set such that the sampling period is larger than the period of the expected noise. This way a noise spike only corrupts one sample. The FILT\_CNT value must be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT\_CNT power.

Table 9-3 summarizes maximum latency values for the various modes of operation in the absence of noise. Filtering latency is restarted each time an actual output transition is masked by noise.

The values of FILT\_PER (or SAMPLE period) and FILT\_CNT must also be traded off against the desire for minimal latency in recognizing actual comparator output transitions. The probability of detecting an actual output change within the nominal latency is the probability of a correct sample raised to the FILT\_CNT power.

**Table 9-3. Comparator Sample/Filter Maximum Latencies**

Mode #	EN	WE	SE	FILT_CNT	FILT_PER	Operation	Maximum Latency <sup>1</sup>
1	0	X	X	X	X	Disabled	N/A

Table 9-3. Comparator Sample/Filter Maximum Latencies (continued)

Mode #	EN	WE	SE	FILT_CNT	FILT_PER	Operation	Maximum Latency <sup>1</sup>
2A	1	0	0	0x0	X	Continuous Mode	$T_{PD}$
2B	1	0	0	X	0x00		
3A	1	0	1	0x1	X	Sampled, Non-Filtered Mode	$T_{PD} + T_{SAMPLE} + T_{per}$
3B	1	0	0	0x1	> 0x0		$T_{PD} + (FILT\_PER \times T_{per}) + T_{per}$
4A	1	0	1	> 0x1	X	Sampled, Filtered Mode	$T_{PD} + (FILT\_CNT \times T_{SAMPLE}) + T_{per}$
4B	1	0	0	> 0x1	> 0x0		$T_{PD} + (FILT\_CNT \times FILT\_PER \times T_{per}) + T_{per}$
5A	1	1	0	0x0	X	Windowed Mode	$T_{PD} + T_{per}$
5B	1	1	0	X	0x00		$T_{PD} + T_{per}$
6	1	1	0	0x1	0x01 – 0xFF	Windowed / Resampled Mode	$T_{PD} + (FILT\_PER \times T_{per}) + 2T_{per}$
7	1	1	0	> 0x1	0x01 – 0xFF	Windowed / Filtered Mode	$T_{PD} + (FILT\_CNT \times FILT\_PER \times T_{per}) + 2T_{per}$

<sup>1</sup>  $T_{PD}$  represents the intrinsic delay of the analog component plus the polarity select logic.  $T_{SAMPLE}$  is the clock period of the external sample clock.  $T_{per}$  is the period of the peripheral bus clock.

## 9.8 Interrupts

The HSCMP module is capable of generating an interrupt on the rising or falling edge of the comparator output (or both). The interrupt request is asserted when HSCMPxSCR[IER] and HSCMPxSCR[CFR] bits are set. It is also asserted when HSCMPxSCR[IEF] and HSCMPxSCR[CFF] bits are set. The interrupt is de-asserted by clearing HSCMPxSCR[IER] or HSCMPxSCR[CFR] for a rising edge interrupt, or HSCMPxSCR[IEF] or HSCMPxSCR[CFF] for a falling edge interrupt.

## 9.9 DMA Support<sup>1</sup>

Normally, the HSCMP generates a CPU interrupt if there is a change on the COUT. When DMA support (set HSCMPxSCR[DMAEN]) enables and the interrupt enables (set HSCMPxSCR[IER] or HSCMPxSCR[IEF] or both), the corresponding change on COUT forces a DMA transfer request rather than a CPU interrupt instead. When the DMA has completed the transfer, it sends a dma\_done signal that de-asserts the dma\_request and clears the flag to allow a subsequent change on comparator output to occur and force another DMA request.

1. For the DMA transfer request to be generated, the DMA enable and the corresponding interrupt enable must be set.



## 9.10 Memory Map & Register Definitions

Table 9-4. Module Memory Map

Address	Use	Access
Base + 0x00	HSCMP Control Register 0 (HSCMPxCR0)	Read/Write
Base + 0x01	HSCMP Control Register 1 (HSCMPxCR1)	Read/Write
Base + 0x02	HSCMP Filter Period Register (HSCMPxFPR)	Read/Write
Base + 0x03	HSCMP Status & Control Register (HSCMPxSCR)	Read/Write
Base + 0x04	HSCMP Pin Control Register (HSCMPxPCR)	Read/Write

### 9.10.1 Control Register 0 (HSCMPxCR0)

Address: HSCMP\_BASE + 0x00

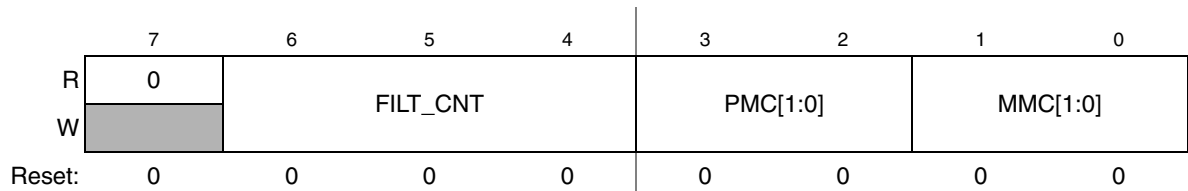


Figure 9-14. HSCMP Control Register 0 (HSCMPxCR0)

Table 9-5. HSCMPxCR0 Field Descriptions

Field	Description
7	Reserved, must be cleared.
6–4 FILT_CNT	Filter sample count. These bits represent the number of consecutive samples that must agree prior to the comparator output filter accepting a new output state. Filter programming and latency details are described in <a href="#">Section 9.7.5, “Low Pass Filter”</a> . 000 Filter is disabled. If SE = 1, COUT is cleared (this is not a legal state in <a href="#">Table 9-2</a> , and is not recommended). If SE = 0, COUT = COUTA. 001 1 consecutive samples must agree (comparator output is simply sampled) 010 2 consecutive samples must agree 011 3 consecutive samples must agree 100 4 consecutive samples must agree 101 5 consecutive samples must agree 110 6 consecutive samples must agree 111 7 consecutive samples must agree
3–2 PMC[1:0]	Positive input mux control. Determines which input is selected for the positive input of the comparator. 00 P0 01 P1 10 P2 11 P3
1–0 MMC[1:0]	Minus input mux control. Determines which input is selected for the minus input of the comparator. 00 M0 01 M1 10 M2 11 M3

### 9.10.2 Control Register 1 (HSCMPxCR1)

Address: HSCMP\_BASE + 0x01

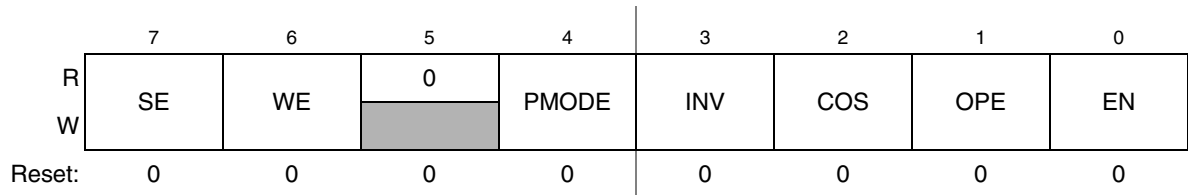


Figure 9-15. HSCMP Control Register 1 (HSCMPxCR1)

Table 9-6. HSCMPxCR1 Field Descriptions

Field	Description
7 SE	Sample enable 0 Sampling mode not selected 1 Sampling mode selected At most, one of SE and WE can be set at a time. If a write to this register attempts to set both, then SE is set and WE is cleared. However, writing ones to both bit locations must be avoided, as the “11” case is reserved and may change in future implementations.
6 WE	Windowing enable 0 Windowing mode not selected 1 Windowing mode selected At most, one of SE and WE can be set at a time. If a write to this register attempts to set both, then SE is set and WE is cleared. However, writing ones to both bit locations must be avoided, as the “11” case is reserved and may change in future implementations.
5	Reserved, must be cleared.
4 PMODE	Power mode select 0 Power savings mode selected. 1 High speed comparison mode selected.
3 INV	Comparator invert. This bit allows you to select the polarity of the comparator function. It is also driven to the COUT output (on both the device pin and as HSCMPxSCR[COUT]) when OPE = 0. 0 Do not invert the comparator output. 1 Invert the output of the analog comparator.
2 COS	Comparator output select 0 Set CMPO equal to COUT (filtered comparator output). 1 Set CMPO equal to COUTA (unfiltered comparator output).
1 OPE	Comparator output pin enable. OPE is used to enable the comparator output to be placed onto the external pin, CMPO. 0 The comparator output (CMPO) is not available on the associated CMPO output pin. The pin is available for use by other on-chip functions. 1 The comparator output (CMPO) is driven out on associated CMPO output pin.
0 EN	Comparator module enable. The EN bit enables the Analog Comparator module. When the module is not enabled, it remains in the off state, and consumes no power. 0 Analog comparator disabled. 1 Analog comparator enabled.

### 9.10.3 Filter Period Register (HSCMPxFPR)

Address: HSCMP\_BASE + 0x02

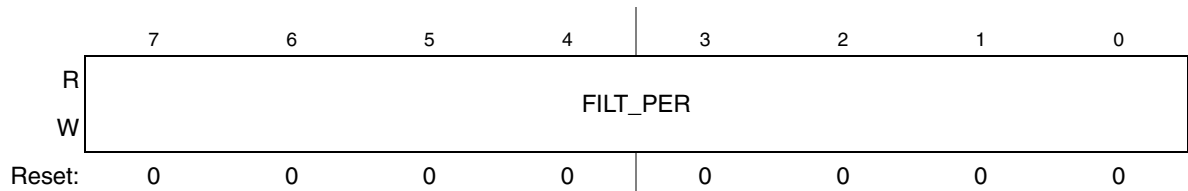


Figure 9-16. HSCMP Filter Period Register (HSCMPxFPR)

Table 9-7. HSCMPxFPR Field Descriptions

Field	Description
7–0 FILT_PER	Filter sample period. When HSCMPxCR1[SE] is cleared, this field specifies the sampling period, in peripheral clock cycles, of the comparator output filter. Setting FILT_PER to 0x0 disables the filter. Filter programming and latency details are described in <a href="#">Section 9.7.5, “Low Pass Filter”</a> . This field has no affect when HSCMPxCR1[SE] is equal to one. In that case, the external SAMPLE signal is used to determine the sampling period.

### 9.10.4 Status & Control Register (HSCMPxSCR)

Address: HSCMP\_BASE + 0x03

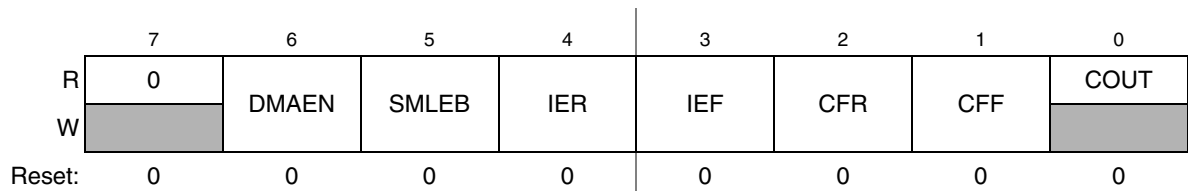


Figure 9-17. HSCMP Status and Control Register (HSCMPxSCR)

Table 9-8. HSCMPxSCR Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 DMAEN	DMA enable control. The DMAEN bit enables the DMA transfer triggered from the ACM. When this bit is set, a DMA requirement is asserted when CFR or CFF bit is set. 0 DMA disabled. 1 DMA enabled.
5 SMELB	Stop mode edge/level interrupt control. This bit controls whether the CFR and CFF bits are edge sensitive or level sensitive in stop mode. 0 CFR/CFF are level sensitive in stop mode. CFR is asserted when COUT is high. CFF is asserted when COUT is low. 1 CFR/CFF are edge sensitive in stop mode. An active low-to-high transition must be seen on COUT to assert CFR, and an active high-to-low transition must be seen on COUT to assert CFF.

**Table 9-8. HSCMPxSCR Field Descriptions (continued)**

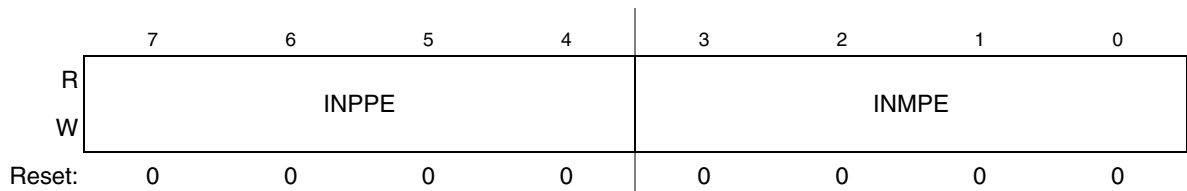
Field	Description
4 IER	Comparator interrupt enable rising. When DMA is not supported (DMAEN = 0), the IER bit enables the CFR interrupt from the ACM. When this bit is set, an interrupt will be asserted when the CFR bit is set. When DMAEN = 1, the IER bit enables the CFR causing a DMA event. 0 Interrupt disabled (DMAEN = 0) or DMA on CFR disabled (DMAEN = 0). 1 Interrupt enabled (DMAEN = 0) or DMA on CFR enabled (DMAEN = 1).
3 IEF	Comparator interrupt enable falling. When DMA is not supported (DMAEN = 0), the IEF bit enables the CFF interrupt from the ACM. When this bit is set, an interrupt will be asserted when the CFF bit is set. When DMAEN = 1, the IEF bit enables the CFR causing a DMA event. 0 Interrupt disabled (DMAEN = 0) or DMA on CFF disabled (DMAEN = 0). 1 Interrupt enabled (DMAEN = 0) or DMA on CFF enabled (DMAEN = 1).
2 CFR	Analog comparator flag rising. During normal operation, the CFR bit is set when a rising edge on COUT has been detected. The CFR bit is cleared by writing a logic one to the bit in normal mode, when DMA is enabled, CFR bit is cleared by the completion of a DMA transfer. During stop mode, CFR can be programmed as either edge or level sensitive via the SMELB bit. <sup>1</sup> 0 Rising edge on COMPO has not been detected. 1 Rising edge on COUT has occurred.
1 CFF	Analog comparator flag falling. During normal operation, the CFF bit is set when a falling edge on COUT has been detected. The CFF bit is cleared by writing a logic one to the bit in normal mode, when DMA is enabled, CFF bit is cleared by the completion of a DMA transfer. During STOP mode, CFF can be programmed as either edge or level sensitive via the SMELB bit. <sup>1</sup> 0 A falling edge on COUT has not been detected. 1 Falling edge on COUT has occurred.
0 COUT	Analog comparator output. Reading the COUT bit returns the current value of the analog comparator output. The register bit is reset to zero and is read as HSCMPxCR1[INV] when the Analog Comparator module is disabled (HSCMPxCR1[EN] = 0). Writes to this bit are ignored.

<sup>1</sup> Edge detection during STOP is only supported on platforms which allow peripherals to be clocked during stop modes. If the CFF and CFR flags are to be active during STOP, then SMELB must be set to 0 for cases where the it is not receiving a clock during STOP.

### 9.10.5 Pin Control Register (HSCMPxPCR)

The HSCMPxPCR is used to request static ownership of a given package pin by the HSCMP. The HSCMPxPCR must be programmed to enable HSCMP ownership of all input pins which may be required by an application. These fields are in addition to HSCMPxCR0[PMC] and HSCMPxCR0[MMC] that control on-the-fly switching between inputs.

Address: HSCMP\_BASE + 0x04



**Figure 9-18. HSCMP Pin Control Register (HSCMPxPCR)**

Table 9-9. HSCMPxPCR Field Descriptions

Field	Description
7–4 INPPE	Positive input pin enable xxx0 Input pin P0 is not required by the HSCMP xxx1 Input pin P0 is required by the HSCMP xx0x Input pin P1 is not required by the HSCMP xx1x Input pin P1 is required by the HSCMP x0xx Input pin P2 is not required by the HSCMP x1xx Input pin P2 is required by the HSCMP 0xxx Input pin P3 is not required by the HSCMP 1xxx Input pin P3 is required by the HSCMP
3–0 INMPE	Minus input pin enable xxx0 Input pin M0 is not required by the HSCMP xxx1 Input pin M0 is required by the HSCMP xx0x Input pin M1 is not required by the HSCMP xx1x Input pin M1 is required by the HSCMP x0xx Input pin M2 is not required by the HSCMP x1xx Input pin M2 is required by the HSCMP 0xxx Input pin M3 is not required by the HSCMP 1xxx Input pin M3 is required by the HSCMP

As an example, if an application would like to cycle through P0 – P3, comparing each in turn to M0. In this case, M1, M2, and M3 would be unused. Set HSCMPxPCR to 0xF1, HSCMPxCR0[MMC] to zero, and HSCMPxCR0[PMC] cycled through the values 00, 01, 10, and 11 via software.

# Chapter 10

## Analog-to-Digital Converter (S08ADC12V3)

### 10.1 Introduction

The 12-bit analog-to-digital converter (ADC) is a successive approximation ADC designed for operation within an integrated microcontroller system-on-chip.

#### NOTE

The MCF51AG128 series of MCUs do not include stop1 low-power mode. Ignore all references to stop1.

#### 10.1.1 ADC Clock Gating

The bus clock to the ADC can be gated on and off using the SCGC1[ADC] bit (see [Section 5.8.9, “System Clock Gating Control 1 Register \(SCGC1\)”](#)). This bit is cleared after any reset, which disables the bus clock to this module. The SCGC1[ADC] bit should be set before operation. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

#### 10.1.2 Module Configurations

This section provides information for configuring the ADC on this device.

##### 10.1.2.1 Channel Assignments

The ADC channel assignments for this device are shown in [Table 10-1](#). Reserved channels convert to an unknown value. Channels which are connected to an I/O pin have an associated pin control bit on System ADC Pin Enable Registers (SAPE1 to SAPE3) as shown.

**Table 10-1. ADC Channel Assignment**

ADCH	Channel	Input	Pin Control
00000	AD0	PTD6/ADP0	ADPC0
00001	AD1	PTD5/ADP1	ADPC1
00010	AD2	PTD4/ADP2	ADPC2
00011	AD3	PTG4/ADP3	ADPC3
00100	AD4	PTG3/ADP4	ADPC4
00101	AD5	PTD3/ADP5	ADPC5
00110	AD6	PTD2/ADP6	ADPC6
00111	AD7	PTD7/ADP7	ADPC7
10000	AD16	PTB2/ADP16	ADPC16
10001	AD17	PTB1/ADP17	ADPC17
10010	AD18	PTB0/ADP18	ADPC18
10011	AD19	PTH3/ADP19	ADPC19
10100	AD20	PTH2/ADP20	ADPC20
10101	AD21	PTH1/ADP21	ADPC21
10110	AD22	PTH0/ADP22	ADPC22
10111	AD23	PTA7/ADP23	ADPC23

Table 10-1. ADC Channel Assignment (continued)

ADCH	Channel	Input	Pin Control	ADCH	Channel	Input	Pin Control
01000	AD8	PTC2/ADP8	ADPC8	11000	AD24	Reserved	N/A
01001	AD9	PTD1/ADP9	ADPC9	11001	AD25	Reserved	N/A
01010	AD10	PTD0/ADP10	ADPC10	11010	AD26	Temperature Sensor <sup>1</sup>	N/A
01011	AD11	PTB7/ADP11	ADPC11	11011	AD27	Internal Bandgap	N/A
01100	AD12	PTB6/ADP12	ADPC12	11100	—	Reserved	N/A
01101	AD13	PTB5/ADP13	ADPC13	11101	V <sub>REFH</sub>	V <sub>REFH</sub>	N/A
01110	AD14	PTB4/ADP14	ADPC14	11110	V <sub>REFL</sub>	V <sub>REFL</sub>	N/A
01111	AD15	PTB3/ADP15	ADPC15	11111	Module Disabled	None	N/A

<sup>1</sup> For information, see [Section 10.4.4, “Temperature Sensor.”](#)

#### NOTE

Selecting the internal bandgap channel requires SPMSC1[BGBE] to be set (see [Section 5.8.6, “System Power Management Status and Control 1 Register \(SPMSC1\)”](#)). See the *MCF51AG128 Data Sheet* (document MCF51AG128) for the value of the bandgap voltage.

#### 10.1.2.2 Alternate Clock

The ADC is capable of performing conversions using the MCU bus clock, the bus clock divided by two, the local asynchronous clock (ADACK) within the module, or the alternate clock (ALTCLK). The ALTCLK on this device is the IC SERCLK.

#### 10.1.2.3 Configurations for Stop Modes

The ADC is capable of running in stop3 mode but requires LVDSE and LVDE in SPMSC1 to be set. The ADC, if enabled, must be configured to use the asynchronous clock source, ADACK, to meet the ADC minimum frequency requirements.

#### 10.1.3 ADC Hardware Trigger

The ADC hardware trigger, ADHWT, is enabled by the ADCSC2[ADTRG] bit. The hardware trigger sources are TriggerA output from PDB or RTC count overflow or iEvent\_0 output that could be selected with the SOPT2[ADHWTS1:ADHWTS0] bits.

The PDB PreTriggerA and PreTriggerB are used as SSEL[0:1] signals of ADC module to support ping-pong mode.

For more details on PDB triggering see [Chapter 24, “Programmable Delay Block \(S08PDBV1\).”](#)

## 10.1.4 Features

Features of the ADC module include:

- Input voltage values may range from  $V_{SSA}$  to  $V_{DDA}$ .
- Linear successive approximation algorithm with 12 bits resolution.
- Up to 28 analog inputs.
- Output formatted in 12-, 10- or 8-bit right-justified format.
- Single, back-to-back, or continuous conversion (automatic return to idle after single conversion).
- Configurable sample time and conversion speed/power.
- Conversion complete flag and interrupt or DMA request.
- Input clock selectable from up to four sources.
- Operation in wait or stop modes for lower noise operation.
- Asynchronous clock source for lower noise operation.
- Selectable asynchronous hardware conversion trigger.
- Temperature sensor.
- Can be configured to take two samples (with no software reconfiguration required) based on hardware triggers during ping-pong mode.

## 10.1.5 Related Material

This block interfaces directly with the programmable gain amplifier and programmable delay block. See [Chapter 24, “Programmable Delay Block \(S08PDBV1\),”](#) for additional information.

## 10.1.6 Block Diagram

[Figure 10-1](#) provides a block diagram of the ADC module.



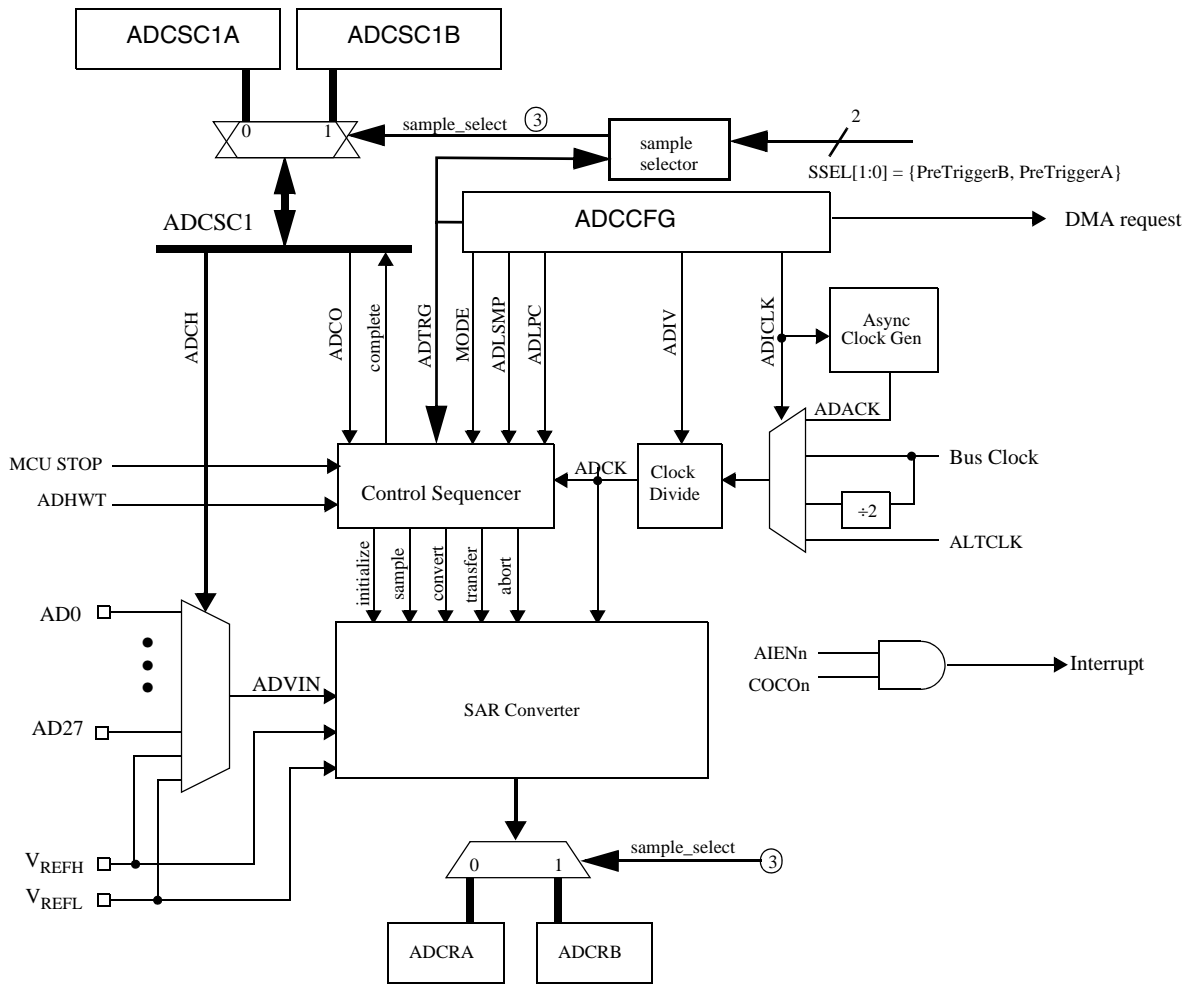


Figure 10-1. ADC Block Diagram

## 10.2 External Signal Description

The ADC module supports up to 28 separate analog inputs. It also requires four supply/reference/ground connections.

Table 10-2. Signal Properties

Name	Function
AD27–AD0	Analog Channel inputs
V <sub>REFH</sub>	High reference voltage
V <sub>REFL</sub>	Low reference voltage

**Table 10-2. Signal Properties (continued)**

Name	Function
$V_{DDAD}$	Analog power supply
$V_{SSAD}$	Analog ground

### 10.2.1 Analog Power ( $V_{DDAD}$ )

The ADC analog portion uses  $V_{DDAD}$  as its power connection. In some packages,  $V_{DDAD}$  is connected internally to  $V_{DD}$ . If externally available, connect the  $V_{DDAD}$  pin to the same voltage potential as  $V_{DD}$ . External filtering may be necessary to ensure clean  $V_{DDAD}$  for good results.

### 10.2.2 Analog Ground ( $V_{SSAD}$ )

The ADC analog portion uses  $V_{SSAD}$  as its ground connection. In some packages,  $V_{SSAD}$  is connected internally to  $V_{SS}$ . If externally available, connect the  $V_{SSAD}$  pin to the same voltage potential as  $V_{SS}$ .

### 10.2.3 Voltage Reference High ( $V_{REFH}$ )

$V_{REFH}$  is the high reference voltage for the converter. In some packages,  $V_{REFH}$  is connected internally to  $V_{DDAD}$ . If externally available,  $V_{REFH}$  may be connected to the same potential as  $V_{DDAD}$ , or may be driven by an external source that is between the minimum  $V_{DDAD}$  spec and the  $V_{DDAD}$  potential ( $V_{REFH}$  must never exceed  $V_{DDAD}$ ).

### 10.2.4 Voltage Reference Low ( $V_{REFL}$ )

$V_{REFL}$  is the low reference voltage for the converter. In some packages,  $V_{REFL}$  is connected internally to  $V_{SSAD}$ . If externally available, connect the  $V_{REFL}$  pin to the same voltage potential as  $V_{SSAD}$ .

### 10.2.5 Analog Channel Inputs (ADx)

The ADC module supports up to 28 separate analog inputs. An input is selected for conversion through the ADCH channel select bits.

## 10.3 Register Definition

These memory mapped registers control and monitor operation of the ADC:

**Table 10-3. ADC Registers**

Register Name	Address Offset	Description
ADCSC1A	0x0	Status and control register 1A
ADCSC2	0x1	Status and control register
Reserved	0x2	—

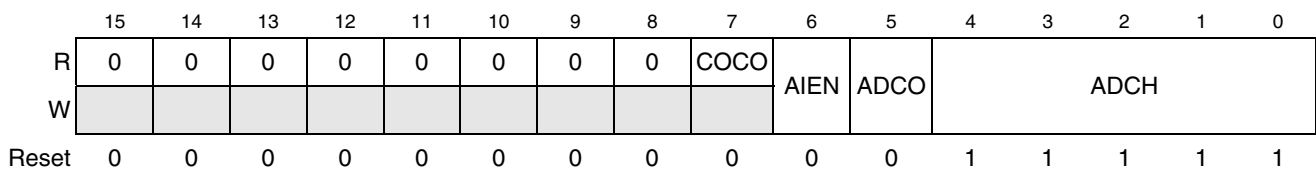
Table 10-3. ADC Registers (continued)

Register Name	Address Offset	Description
Reserved	0x3	—
Reserved	0x4	—
Reserved	0x5	—
ADCCFG	0x6	Configuration register
Reserved	0x7	—
Reserved	0x8	—
Reserved	0x9	—
ADCSC1B	0xA	Status and control register 1B
ADCRA	0xB	Data result register A
ADCRB	0xC	Data result register B
Reserved	0xD	—
Reserved	0xE	—

### 10.3.1 Status and Control Register 1A and 1B (ADCSC1A & ADCSC1B)

This section describes the function of the ADC status and control registers, ADCSC1A and ADCSC1B. These registers have identical fields, and are used in a “ping-pong” approach to control ADC operation. At any one point in time, only one of ADCSC1A and ADCSC1B is actively controlling the ADC analog core. It is possible to write to ADCSC1A while ADCSC1B is driving a conversion, and vice-versa. Writing ADCSC1A while it is actively controlling a conversion aborts the current conversion and initiates a new conversion (if the ADCH bits are equal to a value other than all 1s). The same applies to ADCSC1B.

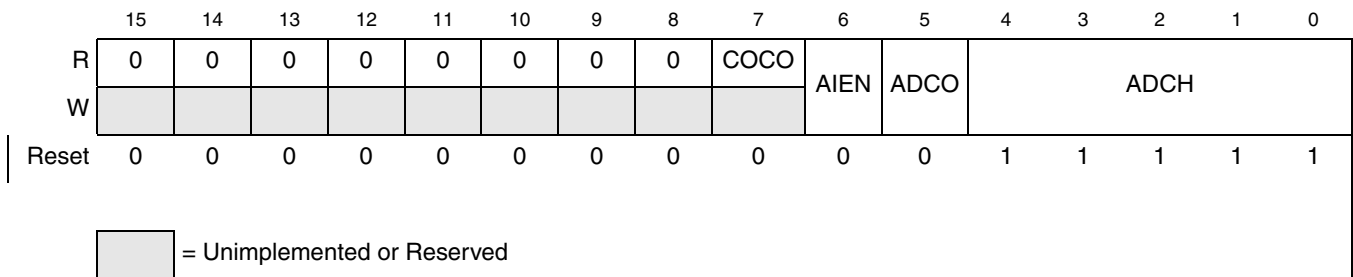
Register address: ADC\_BASE + 0x0



= Unimplemented or Reserved

Figure 10-2. Status and Control Register 1A (ADCSC1A)

Register address: ADC\_BASE + 0xA



**Figure 10-3. Status and Control Register 1B (ADCSC1B)**

**Table 10-4. ADCSC1A/B Register Field Descriptions**

Field	Description
15:8	<b>Reserved</b> — Read and write as zero.
7 COCO	<b>Conversion Complete Flag</b> — The COCO flag is a read-only bit that is set each time a conversion is completed when the compare function is disabled (ACFE = 0). When the compare function is enabled (ACFE = 1), the COCO flag is set upon completion of a conversion only if the compare result is true. This bit is cleared whenever ADCSC1 is written or whenever ADCR is read. 0 Conversion not completed. 1 Conversion completed.
6 AIEN	<b>Interrupt Enable</b> — AIEN is used to enable conversion complete interrupts. When COCO becomes set while AIEN is high, an interrupt is asserted. Interrupts are not generated when DMAEN is set. 0 Conversion complete interrupt disabled. 1 Conversion complete interrupt enabled.
5 ADCO	<b>Continuous Conversion Enable</b> — ADCO is used to enable continuous conversions. 0 One conversion following a write to the ADCSC1 when software triggered operation is selected, or one conversion following assertion of ADHWT when hardware triggered operation is selected. Two conversions will be performed if BB is set. 1 Continuous conversions initiated following a write to ADCSC1 when software triggered operation is selected. Continuous conversions are initiated by an ADHWT event when hardware triggered operation is selected.
4:0 ADCH	<b>Input Channel Select</b> — The ADCH bits form a 5-bit field that is used to select one of the input channels. The input channels are detailed in <a href="#">Figure 10-4</a> . The successive approximation converter subsystem is turned off when the channel select bits are all set to 1. This feature allows for explicit disabling of the ADC and isolation of the input channel from all sources. Terminating continuous conversions this way prevents an additional, single conversion from being performed. It is not necessary to set the channel select bits to all 1s to place the ADC in a low-power state when continuous conversions are not enabled because the module automatically enters a low-power state when a conversion completes.

**Figure 10-4. Input Channel Select**

<b>ADCH</b>	<b>Input Select</b>	<b>ADCH</b>	<b>Input Select</b>
00000	AD0	10000	AD16
00001	AD1	10001	AD17
00010	AD2	10010	AD18

Figure 10-4. Input Channel Select (continued)

ADCH	Input Select	ADCH	Input Select
00011	AD3	10011	AD19
00100	AD4	10100	AD20
00101	AD5	10101	AD21
00110	AD6	10110	AD22
00111	AD7	10111	AD23
01000	AD8	11000	AD24
01001	AD9	11001	AD25
01010	AD10	11010	AD26
01011	AD11	11011	AD27
01100	AD12	11100	Reserved
01101	AD13	11101	V <sub>REFH</sub>
01110	AD14	11110	V <sub>REFL</sub>
01111	AD15	11111	Module disabled

### 10.3.2 Status and Control Register 2 (ADCSC2)

The ADCSC2 register is used to control the conversion trigger and conversion active of the ADC module.

Register address: ADC\_BASE + 0x1

	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	ADACT	ADTRG	0	BB	DMAEN	0	REFSEL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0


 = Unimplemented or Reserved

Figure 10-5. Status and Control Register 2 (ADCSC2)

Table 10-5. ADCSC2 Register Field Descriptions

Field	Description
15:0	<b>Reserved</b> — Read and write as zero.
7 ADACT	<b>Conversion Active</b> — ADACT indicates that a conversion is in progress. ADACT is set when a conversion is initiated and cleared when a conversion is completed or aborted. 0 Conversion not in progress. 1 Conversion in progress.

**Table 10-5. ADCSC2 Register Field Descriptions (continued)**

Field	Description
6 ADTRG	<b>Conversion Trigger Select</b> — ADTRG is used to select the type of trigger to be used for initiating a conversion. Two types of trigger are selectable: software trigger and hardware trigger. When software trigger is selected, a conversion is initiated following a write to ADCSC1. When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input. 0 Software trigger selected. 1 Hardware trigger selected.
5	<b>Reserved</b> — Read and write as zero.
4 BB	<b>Back-to-back Conversion Enable</b> 0 Back-to-back conversion mode is disabled. A and B conversions triggered independently with B conversions only triggerable via hardware trigger sources. 1 Back-to-back conversion mode is enabled. B conversions occur immediately following A conversions, either by software or hardware triggered conversions.
3 DMAEN	<b>DMA Enable</b> — When DMAEN is set, setting one of the COCO flags cause a DMA read request to be sent to the chip's DMA controller. Reading the data result register causes the corresponding COCO flag to be cleared. The DMA read request de-asserts via the handshake from the DMA controller. 0 DMA read request is disabled. 1 DMA read request is enabled, the conversion complete flag asserts the DMA request signal.
2	<b>Reserved</b> — Read and write as zero.
1:0 REFSEL	<b>Voltage Reference Selection</b> — REFSEL bits are used to select the voltage reference source used for conversions. 00 Default voltage reference pin pair ( $V_{REFH}/V_{REFL}$ ). 01 Analog supply pin pair ( $V_{DDA}/V_{SSA}$ ). 10 On-chip bandgap reference. 11 Reserved — Selects default voltage reference ( $V_{REFH}/V_{REFL}$ ) pin pair.

### 10.3.3 Data Result Registers A & B (ADCRA and ADCRB)

Register address: ADC\_BASE + 0xB


	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	ADR11	ADR10	ADR9	ADR8	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 10-6. Data Result Register (ADCRA)**

Register address: ADC\_BASE + 0xC

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	ADR11	ADR10	ADR9	ADR8	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

**Figure 10-7. Data Result Register (ADCRB)**

In 12-bit operation, ADCR[14:3] contains the full 12-bit conversion result. In 10-bit mode, ADCR[12:3] contains the 10-bit conversion result and ADR[14:13] are both zero. Likewise, when configured for 8-bit mode, the result is in ADR[10:3] and ADR[14:11] are zero.

ADCR is updated each time a conversion completes.


In the case that the MODE bits are changed, any data in ADCR becomes invalid.

### 10.3.4 Configuration Register (ADCCFG)

ADCCFG is used to select the mode of operation, clock source, clock divide, and configure for low power or long sample time.

Register address: ADC\_BASE + 0x6

	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	ADLPC	ADIV	ADLSMP	MODE	ADICK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

 = Unimplemented or Reserved

**Figure 10-8. Configuration Register (ADCCFG)**

**Table 10-6. ADCCFG Register Field Descriptions**

Field	Description
7 ADLPC	<b>Low Power Configuration</b> — ADLPC controls the speed and power configuration of the successive approximation converter. This is used to optimize power consumption when higher sample rates are not required. 0 High speed configuration 1 Low power configuration: The power is reduced at the expense of maximum clock speed.
6:5 ADIV	<b>Clock Divide Select</b> — ADIV select the divide ratio used by the ADC to generate the internal clock ADCK. <a href="#">Table 10-7</a> shows the available clock configurations.

**Table 10-6. ADCCFG Register Field Descriptions (continued)**

Field	Description
4 ADLSMP	<b>Long Sample Time Configuration</b> — ADLSMP selects between long and short sample time. This adjusts the sample period to allow higher impedance inputs to be accurately sampled or to maximize conversion speed for lower impedance inputs. Longer sample times can also be used to lower overall power consumption when continuous conversions are enabled if high conversion rates are not required. 0 Short sample time 1 Long sample time
3:2 MODE	<b>Conversion Mode Selection</b> — MODE bits are used to select between 12-, 10-, or 8-bit operation. See <a href="#">Table 10-8</a> .
1:0 ADICLK	<b>Input Clock Select</b> — ADICLK bits select the input clock source to generate the internal clock ADCK. See <a href="#">Table 10-9</a> .

**Table 10-7. Clock Divide Select**

ADIV	Divide Ratio	Clock Rate
00	1	Input clock
01	2	Input clock ÷ 2
10	4	Input clock ÷ 4
11	8	Input clock ÷ 8

**Table 10-8. Conversion Modes**

MODE	Mode Description
00	8-bit conversion (N=8)
01	12-bit conversion (N=12)
10	10-bit conversion (N=10)
11	Reserved

**Table 10-9. Input Clock Select**

ADICLK	Selected Clock Source
00	Bus clock
01	Bus clock divided by 2
10	Alternate clock (ALTCLK)
11	Asynchronous clock (ADACK)



## 10.4 Functional Description

The ADC module is disabled during reset or when the ADCH bits are all high. The module is idle when a conversion has completed and another conversion has not been initiated. When idle, the module is in its lowest power state.

The ADC can perform an analog-to-digital conversion on any of the software selectable channels. In 12-bit and 10-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 12-bit digital result. In 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 9-bit digital result.

When the conversion is completed, the result is placed in the data register. In 10-bit mode, the result is rounded to 10 bits and placed in the data register. In 8-bit mode, the result is rounded to 8 bits and placed in ADCR. The conversion complete flag (COCO) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (AIEN = 1).

### 10.4.1 Clock Select and Divide Control

One of four clock sources can be selected as the clock source for the ADC module. This clock source is then divided by a configurable value to generate the input clock to the converter (ADCK). The clock is selected from one of the following sources by means of the ADICLK bits.

- The bus clock, which is equal to the frequency at which software is executed. This is the default selection following reset.
- The bus clock divided by 2. For higher bus clock rates, this allows a maximum divide by 16 of the bus clock.
- ALTCLK, as defined for this MCU (See module section introduction).
- The asynchronous clock (ADACK) – This clock is generated from a clock source within the ADC module. When selected as the clock source this clock remains active while the MCU is in wait or stop3 mode and allows conversions in these modes for lower noise operation.

Whichever clock is selected, its frequency must fall within the specified frequency range for ADCK. If the available clocks are too slow, the ADC will not perform according to specifications. If the available clocks are too fast, then the clock must be divided to the appropriate frequency. This divider is specified by the ADIV bits and can be divide-by 1, 2, 4, or 8.

### 10.4.2 Hardware Trigger

The ADC module has a selectable asynchronous hardware conversion trigger, ADHWT, that is enabled when the ADTRG bit is set. This source is not available on all MCUs. Consult the module introduction for information on the ADHWT source specific to this MCU.

When ADHWT source is available and hardware trigger is enabled (ADTRG=1), a conversion is initiated on the rising edge of ADHWT. If a conversion is in progress when a rising edge occurs, the rising edge is ignored. In continuous or back-to-back convert configuration, only the initial rising edge to launch a conversion is observed. The hardware trigger function operates in conjunction with any of the conversion modes and configurations.

Figure 10-1 includes the following code segment to describe the operation of the sample select function:

```

if (ADTRG==0) {                                     // software trigger selected
    sample_select=0;
} else {                                           // hardware trigger selected
    if (posedge SSEL[0] & NOT posedge SSEL[1]) sample_select = 0;
    else if (posedge SSEL[1] & NOT posedge SSEL[0]) sample_select = 1;
}
// else NO CHANGE

```

This implies that ADSCS1A and ADCRA are used whenever hardware triggering is not in use. When hardware triggering IS used, then edges on SSEL[1:0] determine which set of control/result registers will be used. A positive edge on SSEL[0] but not on SSEL[1] selects ADSCS1A/ADCRA. A positive edge on SSEL[1] but not on SSEL[0] selects ADSCS1B/ADCRB. Simultaneous changes on both SSEL[1] and SSEL[0] result in no change to the currently selected control/register set.

In essence, SSEL[1:0] act as one-hot control bits to pre-specify the control/result registers to use for the next conversion.

### NOTE

During back-to-back conversions, an A conversion (using ADSCS1A/ADCRA) is always performed first followed by a B conversion (using ADSCS1B/ADCRB) regardless of which trigger SSEL[0] or SSEL[1] asserts.

## 10.4.3 Conversion Control

Conversions can be performed in 12-bit mode, 10-bit mode or 8-bit mode as determined by the MODE bits. Conversions can be initiated by either a software or hardware trigger. In addition, the ADC module can be configured for low power operation, long sample time, and continuous conversion.

### 10.4.3.1 Initiating Conversions

A conversion is initiated:

- Following a write to ADCSC1 (with ADCH bits not all 1s) if software triggered operation is selected.
- Following a hardware trigger (ADHWT) event if hardware triggered operation is selected.
- Following the transfer of the result to the data registers when continuous or back-to-back conversions are enabled.

If continuous or back-to-back conversions are enabled a new conversion is automatically initiated after the completion of the current conversion. In software triggered operation, continuous or back-to-back conversions begin after ADCSC1 is written and continue until aborted (in the case of continuous conversions) or until the second conversion is completed (in the case of back-to-back conversions). In hardware triggered operation, continuous conversions begin after a hardware trigger event and continue until aborted.

### 10.4.3.2 Completing Conversions

A conversion is completed when the result of the conversion is transferred into the data result register, ADCRA or ADCRB. This is indicated by the setting of COCO. An interrupt is generated if AIEN is high at the time that COCO is set.

### 10.4.3.3 Aborting Conversions

Any conversion in progress will be aborted when:

- A write to ADCSC1 occurs (the current conversion will be aborted and a new conversion will be initiated, if ADCH are not all 1s).
- A write to ADCSC2 or ADCCFG occurs. This indicates a mode of operation change has occurred and the current conversion is therefore invalid. No new conversion is initiated.
- The MCU is reset.
- The MCU enters stop mode with ADACK not enabled.

When a conversion is aborted, the contents of the data registers, ADCR, are not altered but continue to be the values transferred after the completion of the last successful conversion. In the case that the conversion was aborted by a reset, ADCR return to its reset state.

### 10.4.3.4 Power Control

The ADC module remains in its idle state until a conversion is initiated. If ADACK is selected as the conversion clock source, the ADACK clock generator is also enabled.

Power consumption when active can be reduced by setting ADLPC. This results in a lower maximum value for  $f_{ADCK}$ .

### 10.4.3.5 Sample Time and Total Conversion Time

The total conversion time depends on the sample time (as determined by ADLSMP), the MCU bus frequency, the conversion mode (8-bit, 10-bit or 12-bit), and the frequency of the conversion clock ( $f_{ADCK}$ ). After the module becomes active, sampling of the input begins. ADLSMP is used to select between short (3.5 ADCK cycles) and long (23.5 ADCK cycles) sample times. When sampling is complete, the converter is isolated from the input channel and a successive approximation algorithm is performed to determine the digital value of the analog signal. The result of the conversion is transferred to ADCR upon completion of the conversion algorithm.

If the bus frequency is less than the  $f_{ADCK}$  frequency, precise sample time for continuous conversions cannot be guaranteed when short sample is enabled (ADLSMP=0). If the bus frequency is less than 1/11th of the  $f_{ADCK}$  frequency, precise sample time for continuous conversions cannot be guaranteed when long sample is enabled (ADLSMP=1).

The maximum total conversion time for different conditions is summarized in [Table 10-10](#).

**Table 10-10. Total Conversion Time vs. Control Conditions**

Conversion Type	ADICLK	ADLSMP	Max Total Conversion Time
Single or first continuous 8-bit	0x, 10	0	20 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	0x, 10	0	23 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	0x, 10	1	40 ADCK cycles + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	0x, 10	1	43 ADCK cycles + 5 bus clock cycles
Single or first continuous 8-bit	11	0	5 $\mu$ s + 20 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	0	5 $\mu$ s + 23 ADCK + 5 bus clock cycles
Single or first continuous 8-bit	11	1	5 $\mu$ s + 40 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	1	5 $\mu$ s + 43 ADCK + 5 bus clock cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	17 ADCK cycles
Subsequent continuous 10-bit or 12-bit; $f_{BUS} \geq f_{ADCK}$	xx	0	20 ADCK cycles
Subsequent continuous 8-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	37 ADCK cycles
Subsequent continuous 10-bit or 12-bit; $f_{BUS} \geq f_{ADCK}/11$	xx	1	40 ADCK cycles

The maximum total conversion time is determined by the clock source chosen and the divide ratio selected. The clock source is selectable by the ADICLK bits, and the divide ratio is specified by the ADIV bits. For example, in 10-bit mode, with the bus clock selected as the input clock source, the input clock divide-by-1 ratio selected, and a bus frequency of 8 MHz, then the conversion time for a single conversion is:

$$\text{Conversion time} = \frac{23 \text{ ADCK cyc}}{8 \text{ MHz}/1} + \frac{5 \text{ bus cyc}}{8 \text{ MHz}} = 3.5 \mu\text{s}$$

$$\text{Number of bus cycles} = 3.5 \mu\text{s} \times 8 \text{ MHz} = 28 \text{ cycles}$$

**NOTE**

The ADCK frequency must be between  $f_{ADCK}$  minimum and  $f_{ADCK}$  maximum to meet ADC specifications.

**10.4.4 Temperature Sensor**

The ADC module includes a temperature sensor whose output is connected to one of the ADC analog channel inputs. [Equation 10-1](#) provides an approximate transfer function of the temperature sensor.

$$\text{Temp} = 25 - ((V_{TEMP} - V_{TEMP25}) \div m) \quad \text{Eqn. 10-1}$$

where:

—  $V_{TEMP}$  is the voltage of the temperature sensor channel at the ambient temperature.

- $V_{TEMP25}$  is the voltage of the temperature sensor channel at 25°C.
- $m$  is the hot or cold voltage versus temperature slope in V/°C.

For temperature calculations, use the  $V_{TEMP25}$  and  $m$  values from the ADC Electricals table in the *MCF51AG128 Data Sheet* (document MCF51AG128).

In application code, the user reads the temperature sensor channel, calculates  $V_{TEMP}$  and compares to  $V_{TEMP25}$ . If  $V_{TEMP}$  is greater than  $V_{TEMP25}$  the cold slope value is applied in [Equation 10-1](#). If  $V_{TEMP}$  is less than  $V_{TEMP25}$  the hot slope value is applied in [Equation 10-1](#).

For more information on using the temperature sensor, consult *AN3031*.

### 10.4.5 MCU Wait Mode Operation

The WAIT instruction puts the MCU in a lower power-consumption standby mode from which recovery is very fast because the clock sources remain active. If a conversion is in progress when the MCU enters wait mode, it continues until completion. Conversions can be initiated while the MCU is in wait mode by means of the hardware trigger or if continuous or back-to-back conversions are enabled.

The bus clock, bus clock divided by two, and ADACK are available as conversion clock sources while in wait mode. The use of ALTCLK as the conversion clock source in wait is dependent on the definition of ALTCLK for this MCU. Consult the module introduction for information on ALTCLK specific to this MCU.

A conversion complete event sets the COCO and generates an ADC interrupt to wake the MCU from wait mode if the ADC interrupt is enabled ( $AIEN = 1$ ).

### 10.4.6 MCU Stop3 Mode Operation

The STOP instruction is used to put the MCU in a low power-consumption standby mode during which most or all clock sources on the MCU are disabled.

#### 10.4.6.1 Stop Mode With ADACK Disabled

If the asynchronous clock, ADACK, is not selected as the conversion clock, executing a STOP instruction during an ADC conversion could result in unexpected behavior. Do not enter stop mode until all conversions have completed when using any clock other than ADACK.

The contents of ADCR are unaffected by stop mode. After exiting from stop mode, a software or hardware trigger is required to resume conversions.

#### 10.4.6.2 Stop Mode With ADACK Enabled

If ADACK is selected as the conversion clock, the ADC continues operation during stop mode. For guaranteed ADC operation, the MCU's voltage regulator must remain active during stop mode. Consult the module introduction for configuration information for this MCU.

If a conversion is in progress when the MCU enters stop mode, it continues until completion. Conversions can be initiated while the MCU is in stop mode by means of the hardware trigger or if continuous or back-to-back conversions are enabled.

A conversion complete event sets the COCO and generates an ADC interrupt to wake the MCU from stop mode if the ADC interrupt is enabled (AIEN = 1).

## 10.4.7 MCU Partial Power Down Mode Operation

The ADC module is automatically disabled when the MCU enters PPD mode. All module registers contain their reset values following exit from PPD mode. Therefore, the module must be re-enabled and re-configured following exit from PPD.

## 10.5 Initialization Information

This section gives an example that provides some basic direction on how a user would initialize and configure the ADC module. The user has the flexibility of choosing between configuring the module for 8-, 10-, or 12-bit resolution, single or continuous conversion, and a polled or interrupt approach, among many other options. Refer to [Table 10-7](#), [Table 10-8](#), and [Table 10-9](#) for information used in this example.

### NOTE

Hexadecimal values designated by a preceding 0x, binary values designated by a preceding %, and decimal values have no preceding character.

### 10.5.1 ADC Module Initialization Example

#### 10.5.1.1 Initialization Sequence

Before the ADC module can be used to complete conversions, an initialization procedure must be performed. A typical sequence is as follows:

1. Update the configuration register (ADCCFG) to select the input clock source and the divide ratio used to generate the internal clock, ADCK. This register is also used for selecting sample time and low-power configuration.
2. Update status and control register 2 (ADCSC2) to select the conversion trigger (hardware or software).
3. Update status and control register 1 (ADCSC1) to select whether conversions are continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions are performed is also selected here.

#### 10.5.1.2 Pseudo-code Example

In this example, the ADC module is set up with interrupts enabled to perform a single 10-bit conversion at low power with a long sample time on input channel 1, where the internal ADCK clock is derived from the bus clock divided by 1.

Table 10-11. ADC Module Pseudo-code Example

ADC Module	Enabled Interrupts			
ADCCFG = 0x98 (%10011000)	Bit 7	ADLPC	1	Configures for low power (lowers maximum clock speed)
	Bit 6:5	ADIV	00	Sets the ADCK to the input clock $\div$ 1
	Bit 4	ADLSP	1	Configures for long sample time
	Bit 3:2	MODE	10	Sets mode at 10-bit conversions
	Bit 1:0	ADICLK	00	Selects bus clock as input clock source
ADCSC2 = 0x00 (%00000000)	Bit 7	ADACT	0	Flag indicates if a conversion is in progress
	Bit 6	ADTRG	0	Software trigger selected
	Bit 5	—	0	Unimplemented or reserved, always reads zero
	Bit 4	—	0	Unimplemented or reserved, always reads zero
	Bit 3:2	—	00	Unimplemented or reserved, always reads zero
	Bit 1:0	—	00	Reserved for Freescale's internal use; always write zero
ADCSC1A = 0x41 (%01000001)	Bit 7	COCO	0	Read-only flag that is set when a conversion completes
	Bit 6	AIEN	1	Conversion complete interrupt enabled
	Bit 5	ADCO	0	One conversion only (continuous conversions disabled)
	Bit 4–0	ADCH	00001	Input channel 1 selected as ADC input channel
ADCRA = xxxx	—	—	—	Holds results of conversion.

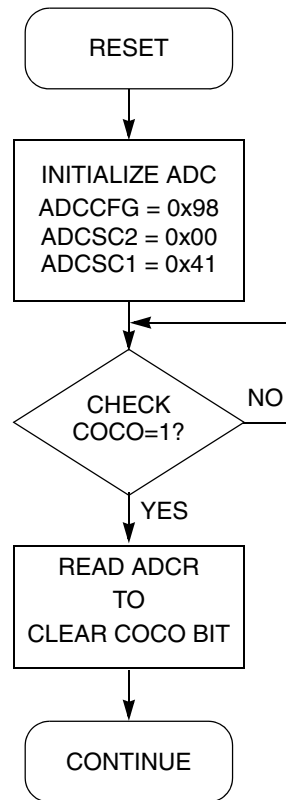


Figure 10-9. Initialization Flowchart for Example

## 10.6 Application Information

This section contains information for using the ADC module in applications. The ADC has been designed to be integrated into a microcontroller for use in embedded control applications requiring an A/D converter.

### 10.6.1 External Pins and Routing

The following sections discuss the external pins associated with the ADC module and how they should be used for best results.

#### 10.6.1.1 Analog Supply Pins

The ADC module has analog power and ground supplies ( $V_{DDAD}$  and  $V_{SSAD}$ ) that are available as separate pins on some devices. On other devices,  $V_{SSAD}$  is shared on the same pin as the MCU digital  $V_{SS}$ , and on others, both  $V_{SSAD}$  and  $V_{DDAD}$  are shared with the MCU digital supply pins. In these cases, there are separate pads for the analog supplies that are bonded to the same pin as the corresponding digital supply so that some degree of isolation between the supplies is maintained.

When available on a separate pin, both  $V_{DDAD}$  and  $V_{SSAD}$  must be connected to the same voltage potential as their corresponding MCU digital supply ( $V_{DD}$  and  $V_{SS}$ ) and must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.



In cases where separate power supplies are used for analog and digital power, the ground connection between these supplies must be at the  $V_{SSAD}$  pin. This should be the only ground connection between these supplies if possible. The  $V_{SSAD}$  pin makes a good single point ground location.

### 10.6.1.2 Analog Reference Pins

In addition to the analog supplies, the ADC module has connections for two reference voltage inputs. The high reference is  $V_{REFH}$ , which may be shared on the same pin as  $V_{DDAD}$  on some devices. The low reference is  $V_{REFL}$ , which may be shared on the same pin as  $V_{SSAD}$  on some devices.

When available on a separate pin,  $V_{REFH}$  may be connected to the same potential as  $V_{DDAD}$ , or may be driven by an external source that is between the minimum  $V_{DDAD}$  spec and the  $V_{DDAD}$  potential ( $V_{REFH}$  must never exceed  $V_{DDAD}$ ). When available on a separate pin,  $V_{REFL}$  must be connected to the same voltage potential as  $V_{SSAD}$ . Both  $V_{REFH}$  and  $V_{REFL}$  must be routed carefully for maximum noise immunity and bypass capacitors placed as near as possible to the package.

AC current in the form of current spikes required to supply charge to the capacitor array at each successive approximation step is drawn through the  $V_{REFH}$  and  $V_{REFL}$  loop. The best external component to meet this current demand is a 0.1  $\mu\text{F}$  capacitor with good high frequency characteristics. This capacitor is connected between  $V_{REFH}$  and  $V_{REFL}$  and must be placed as near as possible to the package pins. Resistance in the path is not recommended because the current causes a voltage drop that could result in conversion errors. Inductance in this path must be minimum (parasitic only).

### 10.6.1.3 Analog Input Pins

The external analog inputs are typically shared with digital I/O pins on MCU devices. The pin I/O control is disabled by setting the appropriate control bit in one of the pin control registers. Conversions can be performed on inputs without the associated pin control register bit set. It is recommended that the pin control register bit always be set when using a pin as an analog input. This avoids problems with contention because the output buffer is in its high impedance state and the pullup is disabled. Also, the input buffer draws DC current when its input is not at either  $V_{DD}$  or  $V_{SS}$ . Setting the pin control register bits for all pins used as analog inputs should be done to achieve lowest operating current.

Empirical data shows that capacitors on the analog inputs improve performance in the presence of noise or when the source impedance is high. Use of 0.01  $\mu\text{F}$  capacitors with good high-frequency characteristics is sufficient. These capacitors are not necessary in all cases, but when used they must be placed as near as possible to the package pins and be referenced to  $V_{SSA}$ .

For proper conversion, the input voltage must fall between  $V_{REFH}$  and  $V_{REFL}$ . If the input is equal to or exceeds  $V_{REFH}$ , the converter circuit converts the signal to 0xFFF (full scale 12-bit representation), 0x3FF (full scale 10-bit representation) or 0xFF (full scale 8-bit representation). If the input is equal to or less than  $V_{REFL}$ , the converter circuit converts it to 0x000. Input voltages between  $V_{REFH}$  and  $V_{REFL}$  are straight-line linear conversions. There is a brief current associated with  $V_{REFL}$  when the sampling capacitor is charging. The input is sampled for 3.5 cycles of the ADCK source when ADLSMP is low, or 23.5 cycles when ADLSMP is high.

For minimal loss of accuracy due to current injection, pins adjacent to the analog input pins must not be transitioning during conversions.

## 10.6.2 Sources of Error

Several sources of error exist for A/D conversions. These are discussed in the following sections.

### 10.6.2.1 Sampling Error

For proper conversions, the input must be sampled long enough to achieve the proper accuracy. Given the maximum input resistance of approximately 7 k $\Omega$  and input capacitance of approximately 5.5 pF, sampling to within 1/4LSB (at 12-bit resolution) can be achieved within the minimum sample window (3.5 cycles @ 8 MHz maximum ADCK frequency) provided the resistance of the external analog source ( $R_{AS}$ ) is kept below 2 k $\Omega$ .

Higher source resistances or higher-accuracy sampling is possible by setting ADLSMP (to increase the sample window to 23.5 cycles) or decreasing ADCK frequency to increase sample time.

### 10.6.2.2 Pin Leakage Error

Leakage on the I/O pins can cause conversion error if the external analog source resistance ( $R_{AS}$ ) is high. If this error cannot be tolerated by the application, keep  $R_{AS}$  lower than  $V_{DDAD} / (2^N \cdot I_{LEAK})$  for less than 1/4LSB leakage error ( $N = 8$  in 8-bit, 10 in 10-bit or 12 in 12-bit mode).

### 10.6.2.3 Noise-Induced Errors

System noise that occurs during the sample or conversion process can affect the accuracy of the conversion. The ADC accuracy numbers are guaranteed as specified only if the following conditions are met:

- There is a 0.1  $\mu$ F low-ESR capacitor from  $V_{REFH}$  to  $V_{REFL}$ .
- There is a 0.1  $\mu$ F low-ESR capacitor from  $V_{DDAD}$  to  $V_{SSAD}$ .
- If inductive isolation is used from the primary supply, an additional 1  $\mu$ F capacitor is placed from  $V_{DDAD}$  to  $V_{SSAD}$ .
- $V_{SSAD}$  (and  $V_{REFL}$ , if connected) is connected to  $V_{SS}$  at a quiet point in the ground plane.
- Operate the MCU in wait or stop3 mode before initiating (hardware triggered conversions) or immediately after initiating (hardware or software triggered conversions) the ADC conversion.
  - For software triggered conversions, immediately follow the write to the ADCSC1 with a WAIT instruction or STOP instruction.
  - For stop3 mode operation, select ADACK as the clock source. Operation in stop3 reduces  $V_{DD}$  noise but increases effective conversion time due to stop recovery.
- There is no I/O switching, input or output, on the MCU during the conversion.

There are some situations where external system activity causes radiated or conducted noise emissions or excessive  $V_{DD}$  noise is coupled into the ADC. In these situations, or when the MCU cannot be placed in wait or stop3 or I/O activity cannot be halted, these recommended actions may reduce the effect of noise on the accuracy:

- Place a 0.01  $\mu$ F capacitor ( $C_{AS}$ ) on the selected input channel to  $V_{REFL}$  or  $V_{SSAD}$  (this improves noise issues but affects sample rate based on the external analog source resistance).

- Average the result by converting the analog input many times in succession and dividing the sum of the results. Four samples are required to eliminate the effect of a 1LSB, one-time error.
- Reduce the effect of synchronous noise by operating off the asynchronous clock (ADACK) and averaging. Noise that is synchronous to ADCK cannot be averaged out.

#### 10.6.2.4 Code Width and Quantization Error

The ADC quantizes the ideal straight-line transfer function into 4096 steps (in 12-bit mode). Each step ideally has the same height (1 code) and width. The width is defined as the delta between the transition points to one code and the next. The ideal code width for an N bit converter (in this case N can be 8, 10 or 12), defined as 1LSB, is:

$$1\text{LSB} = (V_{\text{REFH}} - V_{\text{REFL}}) / 2^N \quad \text{Eqn. 10-2}$$

There is an inherent quantization error due to the digitization of the result. For 8-bit or 10-bit conversions the code transitions when the voltage is at the midpoint between the points where the straight line transfer function is exactly represented by the actual transfer function. Therefore, the quantization error is  $\pm 1/2\text{LSB}$  in 8- or 10-bit mode. As a consequence, however, the code width of the first (0000) conversion is only  $1/2\text{LSB}$  and the code width of the last (FF or 3FF) is  $1.5\text{LSB}$ .

For 12-bit conversions the code transitions only after the full code width is present, so the quantization error is  $-1\text{LSB}$  to  $0\text{LSB}$  and the code width of each step is  $1\text{LSB}$ .

#### 10.6.2.5 Linearity Errors

The ADC may also exhibit non-linearity of several forms. Every effort has been made to reduce these errors but the system should be aware of them because they affect overall accuracy. These errors are:

- Zero-scale error ( $E_{\text{ZS}}$ ) (sometimes called offset) — This error is defined as the difference between the actual code width of the first conversion and the ideal code width ( $1/2\text{LSB}$  in 8-bit or 10-bit modes and  $1\text{LSB}$  in 12-bit mode). Note, if the first conversion is 0x001, then the difference between the actual 0x001 code width and its ideal ( $1\text{LSB}$ ) is used.
- Full-scale error ( $E_{\text{FS}}$ ) — This error is defined as the difference between the actual code width of the last conversion and the ideal code width ( $1.5\text{LSB}$  in 8-bit or 10-bit modes and  $1\text{LSB}$  in 12-bit mode). Note, if the last conversion is 0x3FE, then the difference between the actual 0x3FE code width and its ideal ( $1\text{LSB}$ ) is used.
- Differential non-linearity (DNL) — This error is defined as the worst-case difference between the actual code width and the ideal code width for all conversions.
- Integral non-linearity (INL) — This error is defined as the highest-value the (absolute value of the) running sum of DNL achieves. More simply, this is the worst-case difference of the actual transition voltage to a given code and its corresponding ideal transition voltage, for all codes.
- Total unadjusted error (TUE) — This error is defined as the difference between the actual transfer function and the ideal straight-line transfer function, and therefore includes all forms of error.

### 10.6.2.6 Code Jitter, Non-Monotonicity and Missing Codes

Analog-to-digital converters are susceptible to three special forms of error. These are code jitter, non-monotonicity, and missing codes.

Code jitter is when, at certain points, a given input voltage converts to one of two values when sampled repeatedly. Ideally, when the input voltage is infinitesimally smaller than the transition voltage, the converter yields the lower code (and vice-versa). However, even very small amounts of system noise can cause the converter to be indeterminate (between two codes) for a range of input voltages around the transition voltage. This range is normally around  $\pm 1/2$  LSB in 8-bit or 10-bit mode, or around 2 LSB in 12-bit mode, and will increase with noise.

This error may be reduced by repeatedly sampling the input and averaging the result. Additionally the techniques discussed in [Section 10.6.2.3](#) reduce this error.

Non-monotonicity is defined as when, except for code jitter, the converter converts to a lower code for a higher input voltage. Missing codes are those values which are never converted for any input value.

In 8-bit or 10-bit mode, the ADC is guaranteed to be monotonic and to have no missing codes.

# Chapter 11

## Digital-to-Analog Converter (S08DACV1)

### 11.1 Introduction

#### 11.1.1 Overview

5-bit DAC is 32-tap resistor ladder network that provides a selectable voltage reference for applications where voltage reference is required. The 32-tap resistor ladder network divides the supply reference  $V_{in}$  into 32 voltage level. A 5-bit digital signal input selects output voltage level that varies from  $V_{in}$  to  $V_{in}/32$ .  $V_{in}$  can be selected from two voltage sources,  $V_{in1}$  (internal bandgap supply) and  $V_{in2}$  ( $V_{REFH}$ ).

#### NOTE

- The MCF51AG128 series of MCUs include two DACs.
- Selecting the internal bandgap supply reference requires SPMSC1[BGBE] to be set (see [Section 5.8.6, “System Power Management Status and Control 1 Register \(SPMSC1\)”](#)). See *MCF51AG128 Data Sheet* (document MCF51AG128) for the value of the bandgap voltage.

## 11.1.2 Features

The DAC features include:

- 2.7 V – 5.5 V operating voltage range
- 5-bit resolution
- Selectable supply reference source
- Power down mode to conserve power when it is not being used
- Output can be routed to internal comparator input
- Less than 20 uA power consumption
- Can operation in stop3 mode

## 11.1.3 Modes of Operation

The DAC module is capable of functioning in any mode as long as power down mode is not selected.

## 11.1.4 Block Diagram

The block diagram of the DAC module is shown in [Figure 11-1](#). It contains a 32-tap resistor ladder network and a 32-to-1 multiplexer, which selects an output voltage from one of the 32 distinct levels that outputs from DACO. It is controlled through DAC control register (DACCTRL). Its supply reference source can be selected from two sources,  $V_{in1}$  and  $V_{in2}$ . The module can be powered down (disabled) when it is not used. When in disable mode, DACO is connected to  $V_{SSA}$ .

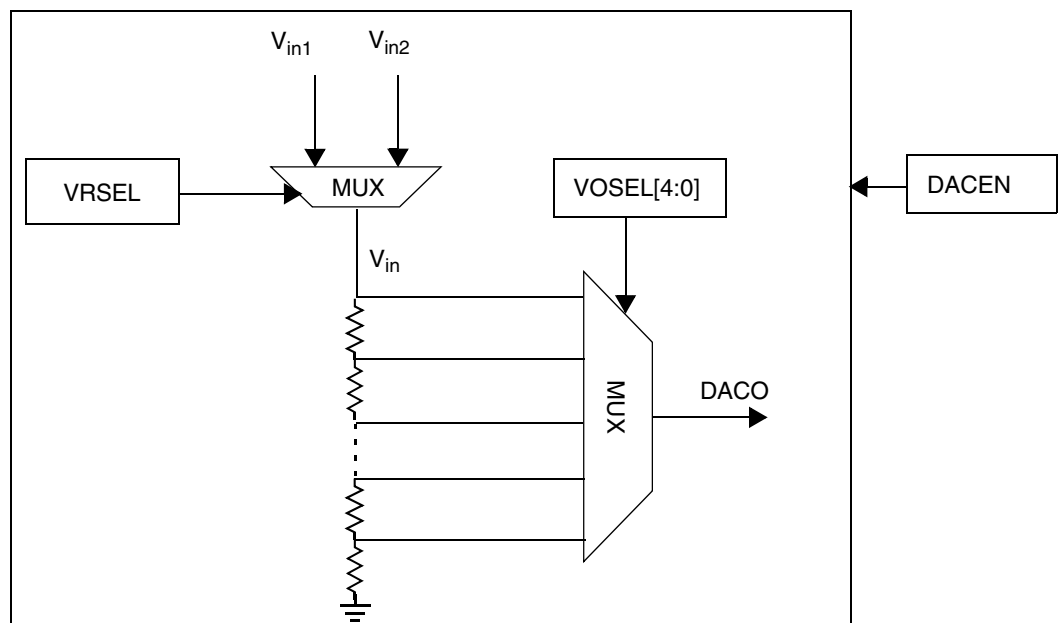


Figure 11-1. 5-bit DAC Block Diagram

## 11.2 Memory Map and Registers

### 11.2.1 Memory Map

Table 11-1. DAC Memory Map

Offset	Register	Description
0x00	DACCTRL	DAC Control Register

### 11.2.2 Registers Descriptions

#### 11.2.2.1 DAC Control Register (DACCTRL)

This register contains control bits for the DAC. The module is enabled if DACEN is set at any operational mode.

Offset: Base + 0x0000

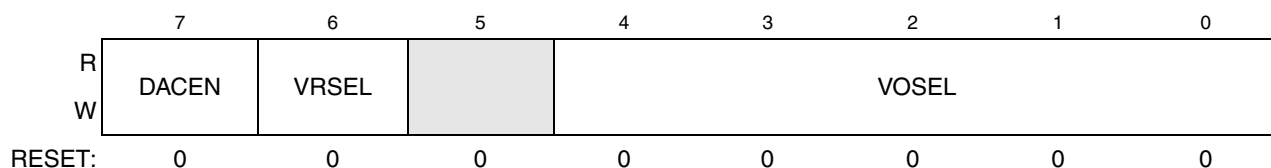


Figure 11-2. DAC Control Register (DACCTRL)

Table 11-2. DACCTRL Register Field Descriptions

Field	Description
7 DACEN	<b>DAC Enable</b> — This bit enables DAC module. When disabled, the module enters power down mode to conserve power. 0 DAC is disabled and the output is $V_{SSA}$ . 1 DAC is operational.
6 VRSEL	<b>Supply Voltage Reference Source Select</b> — This bit selects the supply reference source. 0 $V_{in1}$ is selected as resistor ladder network supply reference $V_{in}$ . 1 $V_{in2}$ is selected as resistor ladder network supply reference $V_{in}$ .
5	Reserved.
[4:0] VOSEL	<b>DAC Output Voltage Select</b> — Selects an output voltage from one of the 32 distinct levels. $DACO = (V_{in}/32) \times (VSEL[4:0] + 1)$ , DACO range is from $V_{in}/32$ to $V_{in}$ .

### 11.2.3 Functional Description

#### 11.2.3.1 Operation

A 32-to-1 multiplexer selects an output voltage from one of the 32 distinct levels that is generated from a 32-tap resistor ladder network. DAC control register (DACCTRL) controls the selection of multiplexer

and resistor ladder network supply reference source. When the module is disabled, its output, DACO, must be equal to  $V_{SSA}$ .

### 11.2.3.2 Voltage Reference Source Select

- $V_{in1}$  should be used to connect to the primary voltage source as supply reference of 32-tap resistor ladder.
- $V_{in2}$  should be used to connect to alternate voltage source (or primary source if alternate voltage source is not available).

### 11.2.3.3 Other Concerns

- The accuracy at worst case is  $\pm 1.5\%$  maximum.
- The settling time must be less than 100 ns.
- The voltage glitch cannot be completely eliminated when changing the output voltage level.

## 11.3 Resets

This module has a single reset input, corresponding to the chip-wide peripheral reset.

## 11.4 Clocks

This module has a single clock input, the bus peripheral clock.

## 11.5 Interrupts

This module has no interrupts.



# Chapter 12

## Cyclic Redundancy Check (S08CRCV3)

### 12.1 Introduction

The CRC block provides hardware acceleration for CRC16-CCITT compliancy with  $x^{16} + x^{12} + x^5 + 1$  polynomial. It is implemented on the 8-bit peripheral bus, and provides a very simple programming interface of only two 8-bit registers. The V1 ColdFire 8-bit peripheral bus bridge serializes 16-bit accesses into two 8-bit accesses, so both registers can be read/written using a single 16-bit read/write instruction.

#### NOTE

The MCF51AG128 series of MCUs do not include stop1 low power mode. Ignore all references to stop1.

#### 12.1.1 CRC Clock Gating

The bus clock to the CRC can be gated on and off using the SCGC2[CRC] bit (see [Section 5.8.10, “System Clock Gating Control 2 Register \(SCGC2\)”](#)). This bit is cleared after any reset that disables the bus clock to this module. The SCGC2[CRC] bit should be set before operation. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

## 12.1.2 Features

Features of the CRC module include:

- Hardware CRC generator circuit using 16-bit shift register
- CRC16-CCITT compliancy with  $x^{16} + x^{12} + x^5 + 1$  polynomial
- Error detection for all single, double, odd, and most multi-bit errors
- Programmable initial seed value
- High-speed CRC calculation
- Optional feature to transpose input data and CRC result via transpose register, required on applications where bytes are in LSB (Least Significant bit) format.

## 12.1.3 Modes of Operation

This section defines the CRC operation in run, wait, and stop modes.

- Run Mode - This is the basic mode of operation.
- Wait Mode - The CRC module is operational.
- Stop 1 and 2 Modes- The CRC module is not functional in these modes and will be put into its reset state upon recovery from stop.
- Stop 3 Mode - In this mode, the CRC module will go into a low power standby state. Any CRC calculations in progress will stop and resume after the CPU goes into run mode.

## 12.1.4 Block Diagram

Figure 12-1 provides a block diagram of the CRC module.

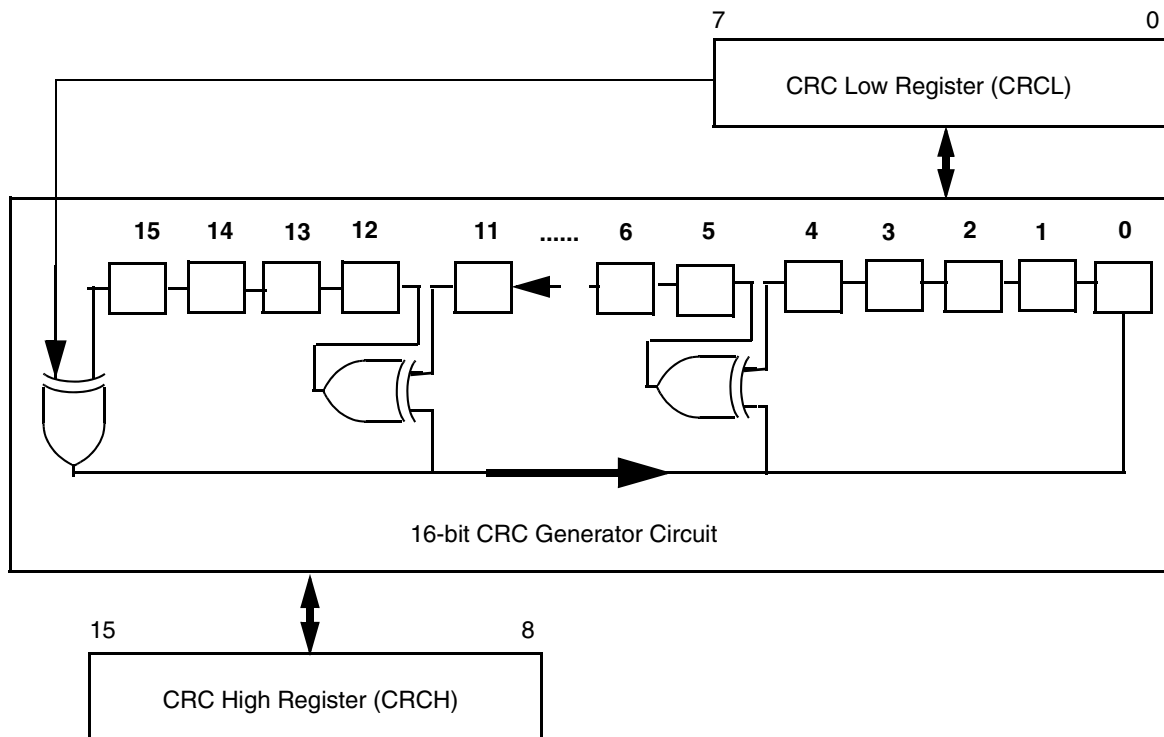


Figure 12-1. Cyclic Redundancy Check (CRC) Module Block Diagram

## 12.2 External Signal Description

There are no CRC signals that connect off chip.

## 12.3 Register Definition

### 12.3.1 Memory Map

Table 12-1. CRC Register Summary

Name		7	6	5	4	3	2	1	0
CRCH (offset=0)	R	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	W								
CRCL (offset=1)	R	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	W								
TRANSPPOSE (offset=2)	R	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	W								

**NOTE**

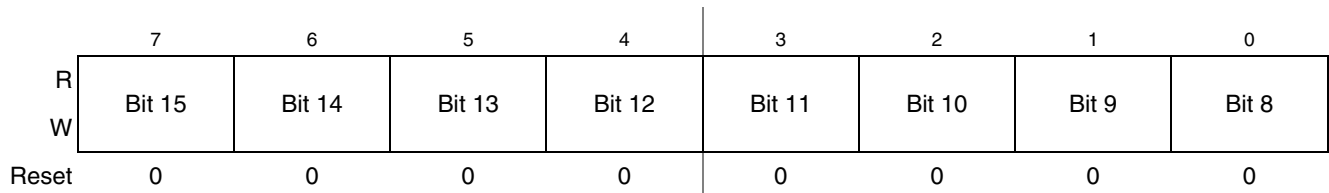
Offsets 4,5,6 and 7 are also mapped onto the CRCL register. This is an *alias* only used on CF1Core (version 1 of ColdFire core) and should be ignored for HCS08 cores. See [Section 12.4.2, “Programming Model Extension for CF1Core”](#) for more details.

**12.3.2 Register Descriptions**

The CRC module includes:

- A 16-bit CRC result and seed register (CRCH:CRCL)
- An 8-bit transpose register to convert from LSB to MSB format (or vice-versa) when required by the application

Refer to the direct-page register summary in the [Chapter 4, “Memory,”](#) for the absolute address assignments for all CRC registers. This section refers to registers only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

**12.3.2.1 CRC High Register (CRCH)****Figure 12-2. CRC High Register (CRCH)****Table 12-2. CRCH Register Field Descriptions**

Field	Description
7:0 CRCH	<b>CRCH</b> -- This is the high byte of the 16-bit CRC register. A write to CRCH will load the high byte of the initial 16-bit seed value directly into bits 15-8 of the shift register in the CRC generator. The CRC generator will then expect the low byte of the seed value to be written to CRCL and loaded directly into bits 7-0 of the shift register. Once both seed bytes written to CRCH:CRCL have been loaded into the CRC generator, and a byte of data has been written to CRCL, the shift register will begin shifting. A read of CRCH will read bits 15-8 of the current CRC calculation result directly out of the shift register in the CRC generator.

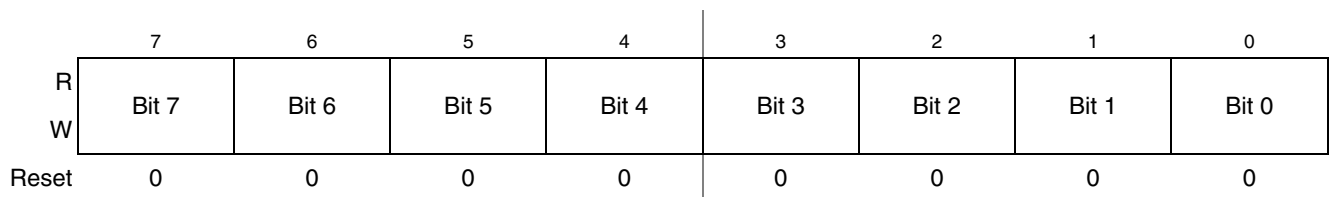
**12.3.2.2 CRC Low Register (CRCL)****Figure 12-3. CRC Low Register (CRCL)**

Table 12-3. CRCL Register Field Descriptions

Field	Description
7:0 CRCL	<b>CRCL</b> -- This is the low byte of the 16-bit CRC register. Normally, a write to CRCL will cause the CRC generator to begin clocking through the 16-bit CRC generator. As a special case, if a write to CRCH has occurred previously, a subsequent write to CRCL will load the value in the register as the low byte of a 16-bit seed value directly into bits 7-0 of the shift register in the CRC generator. A read of CRCL will read bits 7-0 of the current CRC calculation result directly out of the shift register in the CRC generator.

### 12.3.2.3 Transpose Register (TRANPOSE)

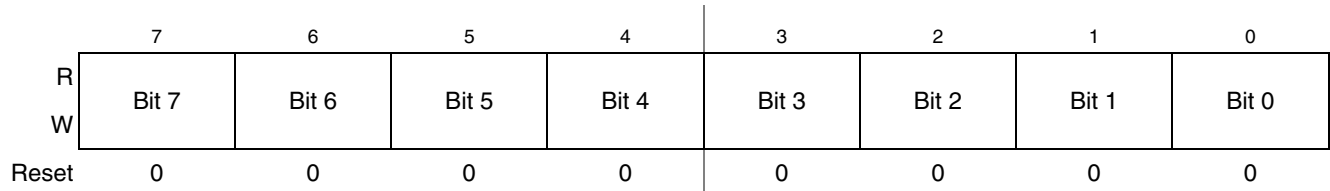


Figure 12-4. Transpose Register (TRANPOSE)

Table 12-4. TRANPOSE Register Field Descriptions

Field	Description
7:0 TRANPOSE	<b>TRANPOSE</b> -- This register is used to transpose data, converting from LSB to MSB (or vice-versa). The byte to be transposed should be first written to TRANPOSE and then subsequent reads from TRANPOSE will return the transposed value of the last written byte (bit 7 becomes bit 0, bit 6 becomes bit 1, and so forth).

## 12.4 Functional Description

To enable the CRC function, a write to the CRCH register will trigger the first half of the seed mechanism which will place the CRCH value directly into bits 15-8 of the CRC generator shift register. The CRC generator will then expect a write to CRCL to complete the seed mechanism.

As soon as the CRCL register is written to, its value will be loaded directly into bits 7-0 of the shift register, and the second half of the seed mechanism will be complete. This value in CRCH:CRCL will be the initial seed value in the CRC generator.

Now the first byte of the data on which the CRC calculation will be applied should be written to CRCL. This write after the completion of the seed mechanism will trigger the CRC module to begin the CRC checking process. The CRC generator will shift the bits in the CRCL register (MSB first) into the shift register of the generator. One Bus cycle after writing to CRCL all 8 bits have been shifted into the CRC generator, and then the result of the shifting, or the value currently in the shift register, can be read directly from CRCH:CRCL, and the next data byte to include in the CRC calculation can be written to the CRCL register.

This next byte will then also be shifted through the CRC generator's 16-bit shift register, and after the shifting has been completed, the result of this second calculation can be read directly from CRCH:CRCL.

After each byte has finished shifting, a new CRC result will appear in CRCH:CRCL, and an additional byte may be written to the CRCL register to be included within the CRC16-CCITT calculation. A new CRC result will appear in CRCH:CRCL each time 8-bits have been shifted into the shift register.

To start a new CRC calculation, write to CRCH, and the seed mechanism for a new CRC calculation will begin again.

### 12.4.1 ITU-T (CCITT) Recommendations and Expected CRC Results

The CRC polynomial  $0x1021 (x^{16} + x^{12} + x^5 + 1)$  is popularly known as *CRC-CCITT* since it was initially proposed by the ITU-T (formerly CCITT) committee.

Although the ITU-T recommendations are very clear about the polynomial to be used, 0x1021, they accept variations in the way the polynomial is implemented:

- ITU-T V.41 implements the same circuit shown in [Figure 12-1](#), but it recommends a SEED = 0x0000.
- ITU-T T.30 and ITU-T X.25 implement the same circuit shown in [Figure 12-1](#), but they recommend the final CRC result to be negated (one-complement operation). Also, they recommend a SEED = 0xFFFF.

Moreover, it is common to find circuits in literature slightly different from the one suggested by the recommendations above, but also known as CRC-CCITT circuits (many variations require the message to be augmented with zeros).

The circuit implemented in the CRC module is exactly the one suggested by the ITU-T V.41 recommendation, with an added flexibility of a programmable SEED. As in ITU-T V.41, no augmentation is needed and the CRC result is not negated. [Table](#) shows some expected results that can be used as a reference. Notice that these are ASCII string messages. For example, message “123456789” is encoded with bytes 0x31 to 0x39 (see ASCII table).

Table 12-5. Expected CRC Results

ASCII String Message	SEED (Initial CRC Value)	CRC Result
"A"	0x0000	0x58e5
"A"	0xffff	0xb915
"A"	0x1d0f <sup>1</sup>	0x9479
"123456789"	0x0000	0x31c3
"123456789"	0xffff	0x29b1
"123456789"	0x1d0f <sup>1</sup>	0xe5cc
A string of 256 upper case "A" characters with no line breaks	0x0000	0xab3
A string of 256 upper case "A" characters with no line breaks	0xffff	0xea0b
A string of 256 upper case "A" characters with no line breaks	0x1d0f <sup>1</sup>	0xe938

<sup>1</sup> One common variation of CRC-CCITT require the message to be augmented with zeros and a SEED=0xFFFF. The CRC module will give the same results of this alternative implementation when SEED=0x1D0F and no message augmentation.

## 12.4.2 Programming Model Extension for CF1Core

The CRC module extends its original programming model to allow faster CRC calculations on CF1 cores. Memory offsets 0x4, 0x5, 0x6 and 0x7 are mapped (aliased) onto the CRCL register, in a way that the CF1Core can execute 32-bit store instructions to write data to the CRC module. The code should use a single *mov.l* store instruction to send four bytes beginning at address 0x4, which will be decomposed by the platform into four sequential/consecutive byte writes to offsets 0x4–0x7 (all aliased to the CRCL position).

In addition, reads from 0x4–0x7 return the contents of the CRCL register.

## 12.4.3 Transpose Feature

The CRC module provides an optional feature to transpose data (invert bit order). This feature is specially useful on applications where the LSB format is used, since the CRC CCITT expects data in the MSB format. In that case, before writing new data bytes to CRCL, these bytes should be transposed as follows:

1. Write data byte to TRANSPOSE register
2. Read data from TRANSPOSE register (subsequent reads will result in the transposed value of the last written data)
3. Write transposed byte to CRCL.

After all data is fed into CRC, the CRCH:CRCL result is available in the MSB format. Then, these two bytes should also be transposed: the values read from CRCH:CRCL should be written/read to/from the TRANSPOSE register.

Although the transpose feature was initially designed to address LSB applications interfacing with the CRC module, it is important to notice that this feature is not necessarily tied to CRC applications. Since it was designed as an independent register, any application should be able to transpose data by writing/reading to/from the TRANSPOSE register (e.g.: Big endian / Little endian conversion in USB)

## 12.5 Initialization Information

To initialize the CRC Module and initiate a CRC16-CCITT calculation, follow this procedure:

1. Write high byte of initial seed value to CRCH.
2. Write low byte of initial seed value to CRCL.
3. Write first byte of data on which CRC is to be calculated to CRCL (an alternative option is offered for CF1Cores. See [Section 12.4.2, “Programming Model Extension for CF1Core”](#)).
4. In the next bus cycle after step 3, if desired, the CRC result from the first byte can be read from CRCH:CRCL.
5. Repeat steps 3–4 until the end of all data to be checked.



# Chapter 13

## FlexTimer Module (S08FTMV3)

### 13.1 Introduction

The FTM is an eight channel timer which supports input capture, output compare, and the generation of PWM signals to control electric motor and power management applications. FTM time reference is a 16-bit counter which can be used as an unsigned or signed counter. FTM target is 8-bit and 32-bit products (high end S08 and ColdFire V1 families primarily). FTM is backwards compatible with the TPM for simple configuration and operation and the high degree of reconfigurability at the design phase.

#### NOTE

MCF51AG128 series MCUs do not support Dual Edge Capture and Quadrature Decoder modes. Please ignore all such references in this chapter.

#### 13.1.1 FTM Module-to-Module Interconnects

##### 13.1.1.1 FTMx Synchronization Trigger

The FTM module supports up to three PWM hardware synchronization signal inputs that, when enabled, update all buffered FTM registers after the rising edge of the trigger and optionally resets the FTM counter. The synchronization trigger is enabled with the FTMxCOMBINEm[SYNCEN] bit and the trigger source is selected in the FTMxSYNC register. [Table 13-1](#) summarizes the FTM trigger sources.

**Table 13-1. FTM Trigger Sources**

Trigger Source	FTMx synchronization
FTMxSYNC bits in SOPT2	FTM1 Trigger0 FTM2 Trigger0
HSCMP1 COUT	FTM1 Trigger1
HSCMP2 COUT	FTM2 Trigger1
iEvent_ch2	FTM1 Trigger2
iEvent_ch1	FTM2 Trigger2

##### 13.1.1.2 FTMx Fault

Each FTM module supports up to four FTM fault sources. On MCF51AG128 series devices, the same source signal is used to signal faults to both the FTM1 and FTM2 in several cases. [Table 13-2](#) summarizes the FTM fault sources.

Table 13-2. FTM Fault Sources

Fault Source	FTMx fault input	FTM filter available
FTM1 FAULT pin	FTM1 Fault1	Yes
FTM2 FAULT pin	FTM2 Fault1	Yes
HSCMP1 COUT	FTM1 Fault2	No
HSCMP2 COUT	FTM2 Fault2	No

### 13.1.1.3 FTM1 Input Capture

The HSCMP1 module can be configured to connect the output of the analog comparator to FTM1 input capture channel 0 by setting the SOPT2[ACIC1] bit. With ACIC1 set, the FTM1CH0 pin is not available externally regardless of the configuration of the FTM1 module.

### 13.1.1.4 FTMx to PDB Hardware Trigger

Each FTM module has the capability to send a hardware trigger to the PDB using the FTMxEXTTRIG register. FTMx can be configured to send a trigger on any channel value match from channels 0-5. Multiple channels can be selected at once to generate multiple triggers to the PDB within one period of the FTM counter.

## 13.1.2 Features

The FTM features include:

- FTM source clock is selectable
  - Source clock can be the system clock, the fixed frequency clock, or an external clock
  - Fixed frequency clock is an additional clock input to allow the selection of an on chip clock source other than the system clock
  - Selecting external clock connects FTM clock to a chip level input pin therefore allowing to synchronize the FTM counter with an off chip clock source
- Prescaler divide-by 1, 2, 4, 8, 16, 32, 64, or 128
- FTM has a 16-bit counter
  - It can be a free-running counter or a counter with initial and final value
  - The counting can be up or up-down
- Each channel can be configured for input capture, output compare, or edge-aligned PWM mode
- In input capture mode
  - the capture can occur on rising edges, falling edges or both edges
  - an input filter can be selected for some channels
- In output compare mode the output signal can be set, cleared, or toggled on match
- All channels can be configured for center-aligned PWM mode
- Each pair of channels can be combined to generate a PWM signal (with independent control of both edges of PWM signal)
- The FTM channels can operate as pairs with equal outputs, pairs with complimentary outputs, or independent channels (with independent outputs)
- The deadtime insertion is available for each complementary pair
- Generation of triggers (match trigger)
- Software control of PWM outputs
- Up to 4 fault inputs for global fault control
- The polarity of each channel is configurable
- The generation of an interrupt per channel
- The generation of an interrupt when the counter overflows
- The generation of an interrupt when the fault condition is detected
- Synchronized loading of write buffered FTM registers
- Write protection for critical registers
- Backwards compatible with TPM
- Testing of input captures for a stuck at zero and one conditions

## 13.1.3 Modes of Operation

When the MCU is in active BDM background or BDM foreground mode, the FTM temporarily suspends all counting until the MCU returns to normal user operating mode. During stop mode, all FTM input clocks

are stopped, so the FTM is effectively disabled until clocks resume. During wait mode, the FTM continues to operate normally. If the FTM does not need to produce a real time reference or provide the interrupt sources needed to wake the MCU from wait mode, the power can then be saved by disabling FTM functions before entering wait mode.

### 13.1.4 Block Diagram

The FTM uses one input/output (I/O) pin per channel, FTMxCHn (FTM channel (n)) where n is the channel number (0–7).

[Figure 13-1](#) shows the FTM structure. The central component of the FTM is the 16-bit counter with programmable initial and final values and its counting can be up or up-down.

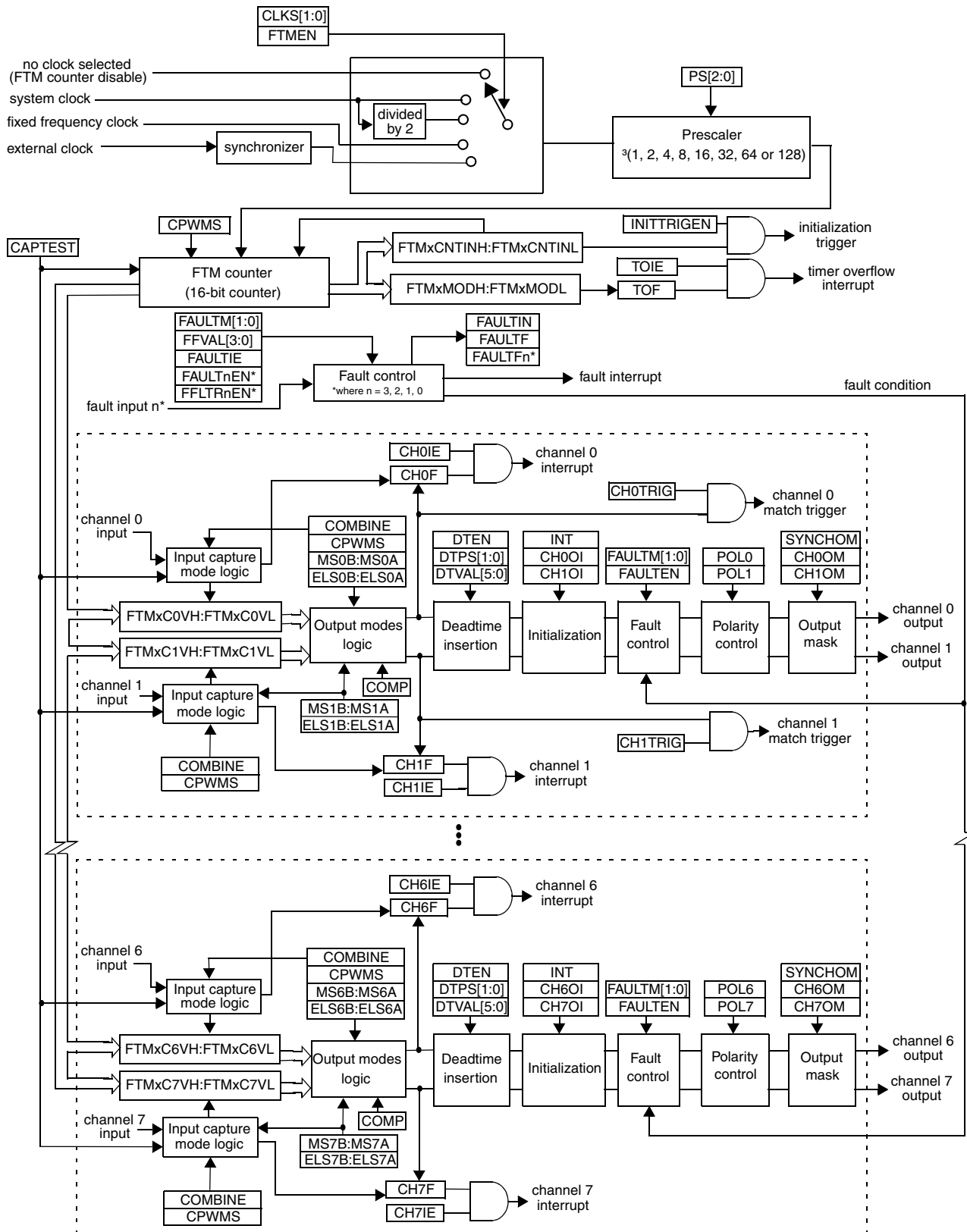


Figure 13-1. FTM Block Diagram

## 13.2 Signal Description

Table 13-3 shows the user-accessible signals for the FTM.

**Table 13-3. Signal Properties**

Name	Function
EXTCLK	FTMx external clock – FTM external clock can be selected to drive the FTM counter.
FTMxCHn <sup>1</sup>	FTMx channel (n) – I/O pin associated with FTM channel (n).
FTMxFAULTj <sup>2</sup>	FTMx fault input (j) – input pin associated with fault input (j).

<sup>1</sup> n = channel number (0 to 7)

<sup>2</sup> j = fault input (0 to 3)

### 13.2.1 EXTCLK — FTM External Clock

The external clock input signal is used as the FTM counter clock if selected by CLKS[1:0] bits in the FTMxSC register. This clock signal must not exceed 1/4 of system clock frequency. The FTM counter prescaler selection and settings are also used when an external clock is selected.

### 13.2.2 FTMxCHn — FTM Channel (n) I/O Pin

Each FTM channel can be configured to operate either as input or output. The direction associated with each channel, input or output, is selected according to the mode assigned for that channel. Please see Table 13-9 for the control bits that define the channel modes.

### 13.2.3 FTMxFAULTj — FTM Fault Input

The fault input signals are used to control the FTMxCHn channel output state. If a fault is detected, the FTMxFAULTj signal is asserted and the channel output is put in a safe state. The behavior of the fault logic is defined by the FAULTM[1:0] control bits in the FTMxMODE register (Section 13.3.10, “FTM Features Mode Selection Register (FTMxMODE)”) and FAULTEN bit in the FTMxCOMBINEm register (Section 13.3.14, “FTM Function for Linked Channels Register (FTMxCOMBINEm)”). Note that each FTMxFAULTj input may affect all channels selectively since FAULTM[1:0] and FAULTEN control bits are defined for each pair of channels. Since there are several FTMxFAULTj inputs, maximum of 4 for the FTM module, each one of these inputs is activated by the FAULTjEN bit in the FTMxFLTCTRL register (Section 13.3.21, “FTM Fault Control Register (FTMxFLTCTRL)”).

## 13.3 Memory Map and Register Definition

This section provides a detailed description of all FTM registers accessible to the end user.

### 13.3.1 Module Memory Map

This section presents a high-level summary of the FTM registers and how they are mapped. [Table 13-4](#) shows the registers for an 8-channel FTM. Register names include the FTM number (x) because it is common for MCU systems to include more than one FTM.

**Table 13-4. 8-channel FTM Module Memory Map**

Address	Use	Access	Section/Page
Base1+0x0000	FTM Status and Control (FTMxSC)	R/W	<a href="#">13.3.3/-9</a>
Base1+0x0001	FTM Counter Register High (FTMxCNTH)	R <sup>1</sup>	<a href="#">13.3.4/-10</a>
Base1+0x0002	FTM Counter Register Low (FTMxCNTL)	R <sup>1</sup>	<a href="#">13.3.4/-10</a>
Base1+0x0003	FTM Modulo Register High (FTMxMODH)	R/W	<a href="#">13.3.5/-11</a>
Base1+0x0004	FTM Modulo Register Low (FTMxMODL)	R/W	<a href="#">13.3.5/-11</a>
Base1+0x0005	FTM Channel 0 Status and Control (FTMxC0SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x0006	FTM Channel 0 Value High (FTMxC0VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0007	FTM Channel 0 Value Low (FTMxC0VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0008	FTM Channel 1 Status and Control (FTMxC1SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x0009	FTM Channel 1 Value High (FTMxC1VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x000A	FTM Channel 1 Value Low (FTMxC1VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x000B	FTM Channel 2 Status and Control (FTMxC2SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x000C	FTM Channel 2 Value High (FTMxC2VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x000D	FTM Channel 2 Value Low (FTMxC2VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x000E	FTM Channel 3 Status and Control (FTMxC3SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x000F	FTM Channel 3 Value High (FTMxC3VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0010	FTM Channel 3 Value Low (FTMxC3VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0011	FTM Channel 4 Status and Control (FTMxC4SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x0012	FTM Channel 4 Value High (FTMxC4VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0013	FTM Channel 4 Value Low (FTMxC4VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0014	FTM Channel 5 Status and Control (FTMxC5SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x0015	FTM Channel 5 Value High (FTMxC5VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0016	FTM Channel 5 Value Low (FTMxC5VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0017	FTM Channel 6 Status and Control (FTMxC6SC)	R/W	<a href="#">13.3.6/-12</a>
Base1+0x0018	FTM Channel 6 Value High (FTMxC6VH)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x0019	FTM Channel 6 Value Low (FTMxC6VL)	R/W	<a href="#">13.3.7/-14</a>
Base1+0x001A	FTM Channel 7 Status and Control (FTMxC7SC)	R/W	<a href="#">13.3.6/-12</a>

**Table 13-4. 8-channel FTM Module Memory Map (continued)**

Base1+0x001B	FTM Channel 7 Value High (FTMxC7VH)	R/W	13.3.7/-14
Base1+0x001C	FTM Channel 7 Value Low (FTMxC7VL)	R/W	13.3.7/-14
<b>FlexTimer specific registers</b>			
Base2+0x0000	FTM Counter Initial Value High (FTMxCNTINH)	R/W	13.3.8/-15
Base2+0x0001	FTM Counter Initial Value Low (FTMxCNTINL)	R/W	13.3.8/-15
Base2+0x0002	FTM Capture and Compare Status Register (FTMxSTATUS)	R/W	13.3.9/-15
Base2+0x0003	FTM Features Mode Selection (FTMxMODE)	R/W	13.3.10/-16
Base2+0x0004	FTM Synchronization (FTMxSYNC)	R/W	13.3.11/-17
Base2+0x0005	FTM Initial State for Channels Output (FTMxOUTINIT)	R/W	13.3.12/-19
Base2+0x0006	FTM Output Mask (FTMxOUTMASK)	R/W	13.3.13/-19
Base2+0x0007	FTM Function For Linked Channels (FTMxCOMBINE0)	R/W	13.3.14/-20
Base2+0x0008	FTM Function For Linked Channels (FTMxCOMBINE1)	R/W	13.3.14/-20
Base2+0x0009	FTM Function For Linked Channels (FTMxCOMBINE2)	R/W	13.3.14/-20
Base2+0x000A	FTM Function For Linked Channels (FTMxCOMBINE3)	R/W	13.3.14/-20
Base2+0x000B	FTM Deadtime Insertion Control (FTMxDEADTIME)	R/W	13.3.15/-21
Base2+0x000C	FTM External Trigger (FTMxEXTTRIG)	R/W	13.3.16/-23
Base2+0x000D	FTM Channels Polarity (FTMxPOL)	R/W	13.3.17/-24
Base2+0x000E	FTM Fault Mode Status (FTMxFMS)	R/W	13.3.18/-25
Base2+0x000F	FTM Input Capture Filter Control (FTMxFILTER0)	R/W	13.3.19/-26
Base2+0x0010	FTM Input Capture Filter Control (FTMxFILTER1)	R/W	13.3.19/-26
Base2+0x0011	FTM Fault Input Filter Control (FTMxFLTFILTER)	R/W	13.3.20/-26
Base2+0x0012	FTM Fault Input Control (FTMxFLTCTRL)	R/W	13.3.21/-27
Base2+0x0013	Reserved	R	–

<sup>1</sup> Read-only for normal access. However, any write to FTMxCNTH or FTMxCNTL causes the 16-bit FTM counter to be updated with the value of FTMxCNTINH:FTMxCNTINL registers.

The FTM memory map can be split into two sets of registers (the first set initial address is Base1 and the second set initial address is Base2) or be only one set (when the second set initial address is Base1 + 0x001D). Refer to the memory map in full-chip specification for the specific chip implementation.

The first set has the original TPM registers. The space of the first set registers not implemented are compressed like in TPM memory map.

The second set has the FTM specific registers and this set can be placed in high memory space. Any second set registers (or bits within these registers) that are used by an unavailable function in the FTM configuration remain in the memory map and in the reset value, so they have no active function.



## NOTE

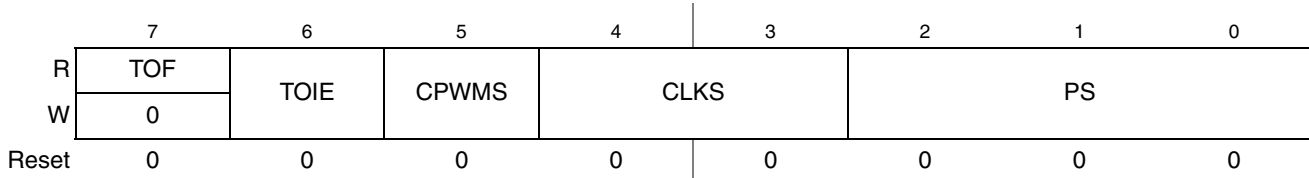
Although, it is possible to write to the FTM specific registers (second set registers) when FTMEN is cleared, it is not recommended and the results can not be guaranteed.

### 13.3.2 Register Descriptions

This section consists of register descriptions in address order. A typical MCU system may contain multiple FTMs and each FTM may have up to eight channels, so register names include placeholder characters to identify which FTM and which channel is being referenced. For example, FTMxCnSC refers to FTM (x) and channel (n). FTM1C2SC is the status and control register for channel 2 of FTM1.

### 13.3.3 FTM Status and Control Register (FTMxSC)

FTMxSC contains the overflow status flag and control bits used to configure the interrupt enable, FTM configuration, clock source, and prescaler factor. These controls relate to all channels within this module.



**Figure 13-2. FTM Status and Control Register (FTMxSC)**

**Table 13-5. FTMxSC Field Descriptions**

Field	Description
7 TOF	<p>Timer overflow flag bit. This read/write bit is set when the FTM counter passes the value in the FTMxMODH:FTMxMODL registers. The TOF bit is cleared by reading FTMxSC register while TOF is set and then writing a logic 0 to TOF bit. If another FTM overflow occurs between the read and write operations, the write operation has no effect, therefore TOF remains set indicating an overflow has occurred. In this case a TOF interrupt request is not lost due to clearing sequence for a previous TOF. TOF bit is cleared out from reset. Writing a logic 1 to TOF has no effect.</p> <p>0 FTM counter has not overflowed. 1 FTM counter has overflowed.</p>
6 TOIE	<p>Timer overflow interrupt enable. This read/write bit enables FTM overflow interrupts. If TOIE is set, an interrupt is generated when TOF equals one. Reset clears TOIE.</p> <p>0 TOF interrupts inhibited (use software polling). 1 TOF interrupts enabled.</p>
5 CPWMS	<p>Center-aligned PWM select. This read/write bit selects CPWM mode. This mode configures the FTM to operate in up-down counting mode. Reset clears CPWMS.</p> <p>CPWMS is write protected, this bit can only be written if WPDIS = 1.</p> <p>0 FTM counter operates in up counting mode. 1 FTM counter operates in up-down counting mode.</p>

**Table 13-5. FTMxSC Field Descriptions (continued)**

Field	Description
4–3 CLKS	Clock source selection. As shown in <a href="#">Table 13-6</a> , this 2-bit field is used to select one of the three FTM counter clock sources. Note that if CLKS[1:0] = 0:0, no clock is selected to the FTM counter which is equivalent as disabling the counter. Note also that if CLKS[1:0] = 0:1 and FTMEN = 0, the system clock is selected with an additional divided by 2 prescaler. CLKS[1:0] bits are write protected, these bits can only be written if WPDIS = 1.
2–0 PS	Prescale factor selection. This 3-bit field selects one of 8 division factors for the clock source selected by CLKS[1:0] bits as shown in <a href="#">Table 13-7</a> . The new prescaler factor affects the clock source on the next system clock cycle after the new value is updated into the register bits. PS[2:0] bits are write protected, these bits can only be written if WPDIS = 1.

**Table 13-6. FTM Clock Source Selection**

CLKS	FTMEN	FTM Clock Source to Prescaler Input
00	X	No clock selected (FTM counter disable)
01	0	System clock divided by 2
	1	System clock
10	X	Fixed frequency clock
11	X	External clock

**Table 13-7. Prescale Factor Selection**

PS	FTM Clock Source Divided-by
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

### 13.3.4 FTM Counter Registers (FTMxCNTH:FTMxCNTL)

The two read-only FTM counter registers contain the high and low bytes of the value in the FTM counter. Reading either byte (FTMxCNTH or FTMxCNTL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This allows coherent 16-bit reads in either big-endian or little-endian order which makes this more friendly to various compiler implementations. The coherency mechanism is automatically restarted by an MCU reset or any write to the FTM status and control register (FTMxSC).

Reset clears the FTM counter registers. Writing any value to FTMxCNTH or FTMxCNTL updates the FTM counter (FTMxCNTH:FTMxCNTL) with its initial value (FTMxCNTINH:FTMxCNTINL) and resets the read coherency mechanism, regardless of the data involved in the write.

	7	6	5	4	3	2	1	0
R	Bit 15	14	13	12	11	10	9	Bit 8
W	Any write to FTMxCNTH updates the 16-bit counter with its initial value (FTMxCNTINH:FTMxCNTINL)							
Reset	0	0	0	0	0	0	0	0

**Figure 13-3. FTM Counter Register High (FTMxCNTH)**

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W	Any write to FTMxCNTL updates the 16-bit counter with its initial value (FTMxCNTINH:FTMxCNTINL)							
Reset	0	0	0	0	0	0	0	0

**Figure 13-4. FTM Counter Register Low (FTMxCNTL)**

When BDM is active, the FTM counter is frozen (this is the value that you may read); the read coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both counter halves are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

### 13.3.5 FTM Counter Modulo Registers (FTMxMODH:FTMxMODL)

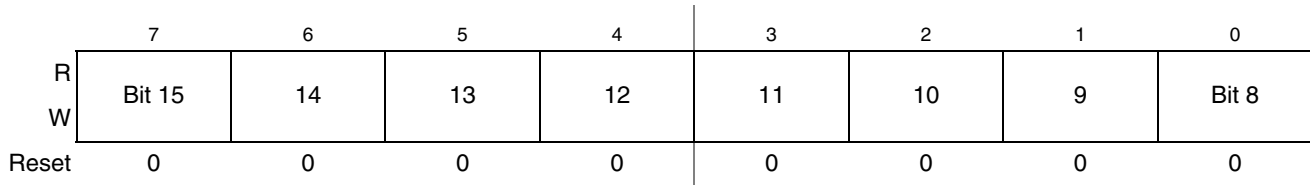
The read/write FTM modulo registers contain the modulo value for the FTM counter. After the FTM counter reaches the modulo value, the overflow flag (TOF) becomes set at the next clock, and the next value of FTM counter depends on the selected counting method ([Section 13.4.3, “Counter”](#)).

Reset sets the FTM counter modulo registers to 0x0000.

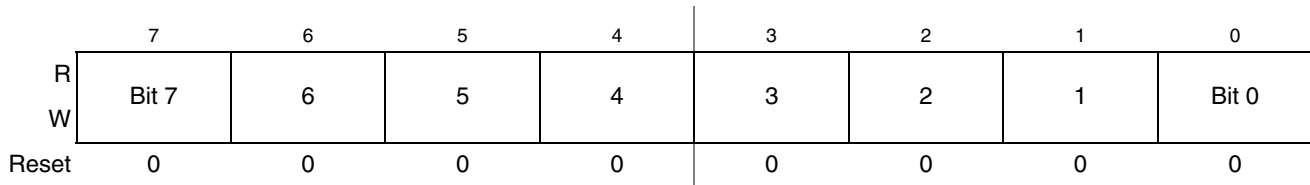
Writing to either byte (FTMxMODH or FTMxMODL) latches the value into a buffer. The registers are updated with the value of their write buffer according to [Section 13.4.10, “Load of the Registers With Write Buffers.”](#)

If FTMMEN = 0, then this write coherency mechanism may be manually reset by writing to the FTMxSC register (whether BDM is active or not).

When BDM is active, this write coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any write to the modulo registers bypasses the buffer latches and directly writes to the modulo register while BDM is active.



**Figure 13-5. FTM Counter Modulo Register High (FTMxMODH)**

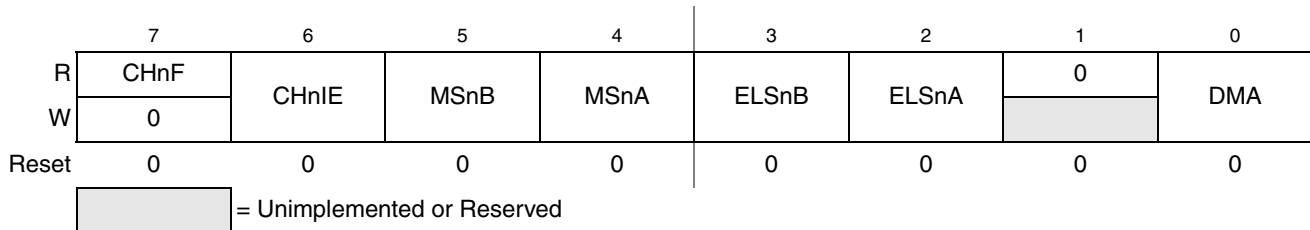


**Figure 13-6. FTM Counter Modulo Register Low (FTMxMODL)**

It is recommended to initialize the FTM counter (write to FTMxCNTH or FTMxCNTL) before writing to the FTM modulo registers to avoid confusion about when the first counter overflow will occur.

### 13.3.6 FTM Channel (n) Status and Control Register (FTMxCnSC)

FTMxCnSC contains the channel-interrupt-status flag and control bits used to configure the interrupt enable, channel configuration, and pin function.



**Figure 13-7. FTM Channel (n) Status and Control Register (FTMxCnSC)**

**Table 13-8. FTMxCnSC Field Descriptions**

Field	Description
7 CHnF	Channel (n) flag bit. This read/write bit is set when an event occurs on channel (n). The CHnF bit is cleared by reading FTMxCnSC register while CHnF is set and then writing a logic 0 to CHnF bit. If another event occurs between the read and write operations, the write operation has no effect, therefore CHnF remains set indicating an event has occurred. In this case a CHnF interrupt request is not lost due to clearing sequence for a previous CHnF. CHnF bit is cleared out from reset. Writing a logic 1 to CHnF has no effect. 0 No channel (n) event occurred. 1 Channel (n) event occurred.
6 CHnIE	Channel (n) interrupt enable. This read/write bit enables interrupts from channel (n). Reset clears CHnIE. 0 Channel (n) interrupt requests disabled (use software polling). 1 Channel (n) interrupt requests enabled.
5 MSnB	Mode select B bit for FTM channel (n). This bit is used for further selections in the channel (n) logic. Its functionality is dependent on the channel mode. Refer to the <a href="#">Table 13-9</a> for details. MSnB is write protected, this bit can only be written if WPDIS = 1.

**Table 13-8. FTMxCnSC Field Descriptions (continued)**

Field	Description
4 MSnA	Mode select A bit for FTM channel (n). This bit is used for further selections in the channel (n) logic. Its functionality is dependent on the channel mode. Refer to the <a href="#">Table 13-9</a> for details. MSnA is write protected, this bit can only be written if WPDIS = 1.
3–2 ELSnB ELSnA	Edge or level selection bits. The functionality of ELSnB and ELSnA bits depends on the channel (n) mode as shown in <a href="#">Table 13-9</a> . ELSnB and ELSnA bits are write protected, these bits can only be written if WPDIS = 1.
0 DMA	DMA enable. The DMA bit enables the channel (n) DMA protocol. 0 DMA transfer request and done are disabled. 1 DMA transfer request and done are enabled.

**Table 13-9. Mode, Edge, and Level Selection**

COMBINE	CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	X	XX	00	Pin not used for FTM — revert the channel pin to general purpose I/O or other peripheral control	
0	0	00	01	Input capture	Capture on rising edge only
			10		Capture on falling edge only
			11		Capture on rising or falling edge
		01	01	Output compare	Toggle output on match
			10		Clear output on match
			11		Set output on match
		1X	10	Edge-aligned PWM	High-true pulses (clear output on match)
			X1		Low-true pulses (set output on match)
		1	XX	10	Center-aligned PWM
	X1			Low-true pulses (set output on match-up)	
1	0	XX	10	Combine PWM	High-true pulses (set on channel (n) match, and clear on channel (n+1) match)
			X1		Low-true pulses (clear on channel (n) match, and set on channel (n+1) match)

### 13.3.7 FTM Channel Value Registers (FTMxCnVH:FTMxCnVL)

These read/write registers contain the captured FTM counter value of the input capture function or the match value for the output modes. The channel registers are cleared by reset.

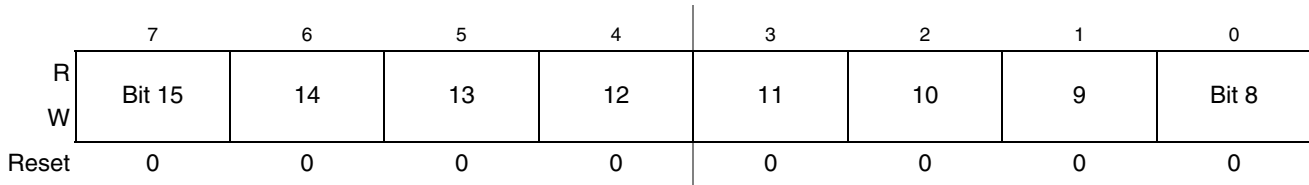


Figure 13-8. FTM Channel Value Register High (FTMxCnVH)

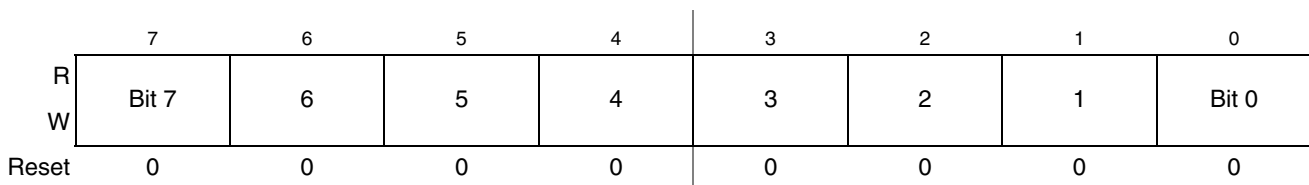


Figure 13-9. FTM Channel Value Register Low (FTMxCnVL)

In input capture mode, reading either byte (FTMxCnVH or FTMxCnVL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This latching mechanism also resets (becomes unlatched) when the FTMxCnSC register is written (whether BDM mode is active or not). Any write to the channel registers is ignored during the input capture mode.

When BDM is active, the read coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the channel value register are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. Any read of the FTMxCnVH and FTMxCnVL registers in BDM mode bypasses the buffer latches and returns the value of these registers and not the value of their read buffer.

In output modes, writing to either byte (FTMxCnVH or FTMxCnVL) latches the value into a buffer. The registers are updated with the value of their write buffer according to [Section 13.4.10, “Load of the Registers With Write Buffers.”](#)

If FTMEN = 0, then this write coherency mechanism may be manually reset by writing to the FTMxCnSC register (whether BDM mode is active or not). This latching mechanism allows coherent 16-bit writes in either big-endian or little-endian order which is friendly to various compiler implementations.

When BDM is active, the write coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active even if one or both halves of the channel value register are written while BDM is active. Any write to the FTMxCnVH and FTMxCnVL registers bypasses the buffer latches and writes directly to the register while BDM is active. The values written to the channel value registers while BDM is active are used in output modes operation once normal execution resumes. Writes to the channel value registers while BDM is active do not interfere with the partial completion of a

coherency sequence. After the write coherency mechanism has been fully exercised, the channel value registers are updated using the buffered values (while BDM was not active).

### 13.3.8 FTM Counter Initial Value Registers (FTMxCNTINH:FTMxCNTINL)

The read/write FTM counter initial value registers contain the initial value for the FTM counter.

Writing to either byte (FTMxCNTINH or FTMxCNTINL) latches the value into a buffer and the FTMxCNTINH:FTMxCNTINL registers are updated when the second byte is written.

When BDM is active, the write coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the counter initial value register are written while BDM is active. Any write to the counter initial value registers bypasses the buffer latches and writes directly to the counter initial value register while BDM is active.

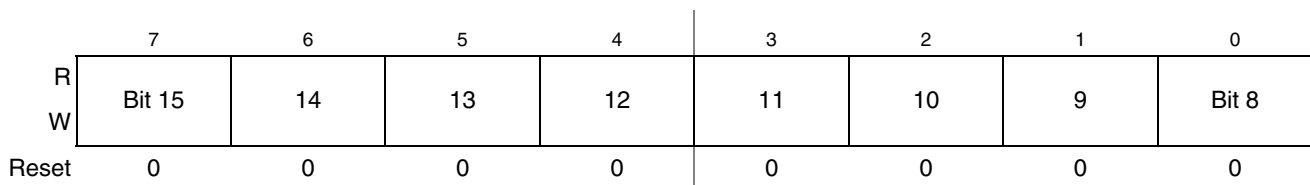


Figure 13-10. FTM Counter Initial Value Register High (FTMxCNTINH)

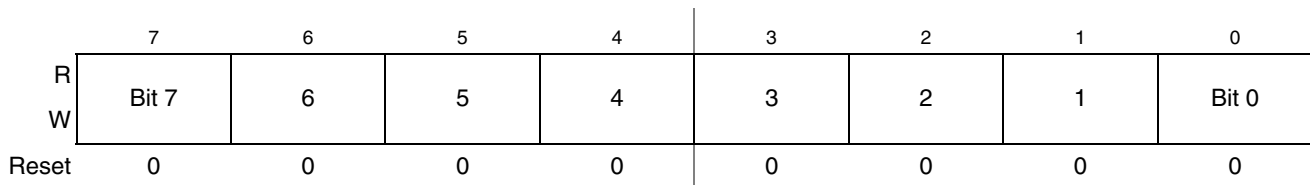


Figure 13-11. FTM Counter Initial Value Register Low (FTMxCNTINL)

The first time that the FTM clock is selected (first write to change the CLKS[1:0] bits to a non-zero value), FTM counter starts with the value 0x0000. To avoid this behavior, before the first write to select the FTM clock, write the new value to the FTM counter initial value registers (FTMxCNTINH:FTMxCNTINL) and then initialize the FTM counter (write a value to FTMxCNTH or FTMxCNTL).

### 13.3.9 FTM Capture and Compare Status Register (FTMxSTATUS)

FTMxSTATUS contains a copy of the status flag CHnF bit (in FTMxCnSC register) for each FTM channel for simpler software driver.

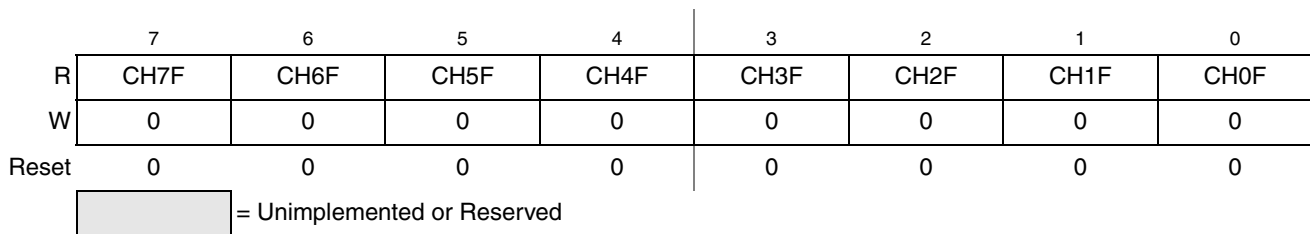


Figure 13-12. FTM Capture and Compare Status Register (FTMxSTATUS)

**Table 13-10. FTMxSTATUS Field Descriptions**

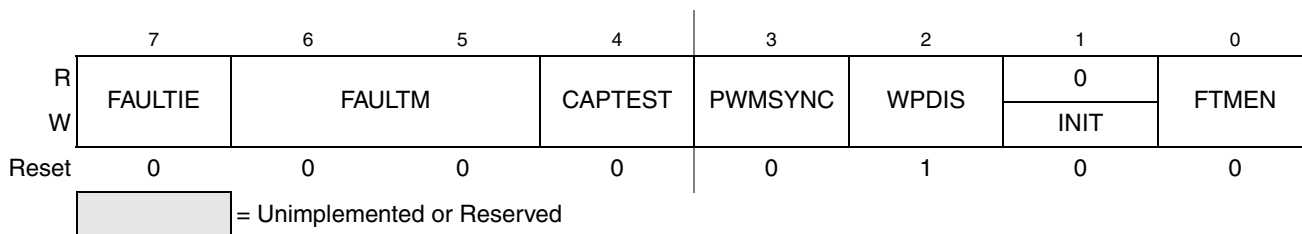
Field	Description
7-0 CHnF	<p>Channel (n) flag bit. This read/write bit is set when an event occurs on channel (n). The CHnF bit is cleared by reading FTMxSTATUS register while CHnF is set and then writing a logic 0 to CHnF bit. If another event occurs between the read and write operations, the write operation has no effect, therefore CHnF remains set indicating an event has occurred. In this case a CHnF interrupt request is not lost due to clearing sequence for a previous CHnF. CHnF bit is cleared out from reset. Writing a logic 1 to CHnF has no effect.</p> <p>0 No channel (n) event occurred. 1 Channel (n) event occurred.</p> <p><b>Note:</b> Each CHnF bit in FTMxSTATUS register is a mirror of CHnF bit in FTMxCnSC register. All CHnF bits can be checked using only one read of FTMxSTATUS register. All CHnF bits can be cleared by reading of FTMxSTATUS register followed by writing 0x00 to FTMxSTATUS register.</p>

**NOTE**

- The use of FTMxSTATUS register is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- The use of this register with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

**13.3.10 FTM Features Mode Selection Register (FTMxMODE)**

This read/write register contains the control bits used to configure the fault interrupt and fault control, capture test mode, PWM synchronization, write protection, channel output initialization, and enable the enhanced features of the FTM. These controls relate to all channels within this module.



**Figure 13-13. FTM Features Mode Selection Register (FTMxMODE)**

**Table 13-11. FTMxMODE Field Descriptions**

Field	Description
7 FAULTIE	<p>Fault interrupt enable. This read/write bit enables the generation of an interrupt when a fault is detected by FTM and the FTM fault control is enabled.</p> <p>0 Fault control interrupt is disabled. 1 Fault control interrupt is enabled.</p>
6-5 FAULTM	<p>Fault control mode. These bits define the FTM fault control mode. The fault control mode is selected according to the <a href="#">Table 13-12</a>.</p> <p>FAULTM[1:0] bits are write protected, these bits can only be written if WPDIS = 1.</p>



Field	Description
4 CAPTEST	Capture test mode enable. This read/write bit enables the capture test mode. CAPTEST bit is write protected, this bit can only be written if WPDIS = 1. 0 Capture test mode is disabled. 1 Capture test mode is enabled.
3 PWMSYNC	PWM synchronization mode. This read/write bit selects which triggers can be used by FTMxMODH:L, FTMxCnVH:L, CHnOM, and FTM counter synchronization (Section 13.4.11, "PWM Synchronization"). 0 No restrictions. Software and hardware triggers can be used by FTMxMODH:L, FTMxCnVH:L, CHnOM, and FTM counter synchronization. 1 Software trigger can only be used by FTMxMODH:L and FTMxCnVH:L synchronization, and hardware triggers can only be used by CHnOM and FTM counter synchronization.
2 WPDIS	Write protection disable. When write protection is enabled (WPDIS = 0), write protected bits can not be written. When write protection is disabled (WPDIS = 1), write protected bits can be written. The WPDIS bit is the negation of the WPEN bit. WPDIS is cleared when 1 is written to WPEN. WPDIS is set when WPEN bit is read as a logic 1 and then 1 is written to WPDIS. Write 0 to WPDIS has no effect. 0 Write protection is enabled. 1 Write protection is disabled.
1 INIT	Initialize the output channels. When a logic 1 is written to INIT bit the output channels are initialized according to the state of their corresponding bit in the FTMxOUTINIT register. Writing a logic 0 to INIT bit has no effect. 0 The INIT bit is always read as logic 0.
0 FTMEN	FTM enable. When FTMEN = 1, all registers including the FTM specific registers (second set registers) are available for use with no restrictions. When FTMEN = 0, only the TPM compatible registers (first set registers) can be used without any restriction. The use of the FTM specific registers when FTMEN = 0 is not recommended and can result in unpredictable behavior. FTMEN bit is write protected, this bit can only be written if WPDIS = 1. 0 Only TPM compatible registers (first set registers) are available. 1 All FTM registers are available.

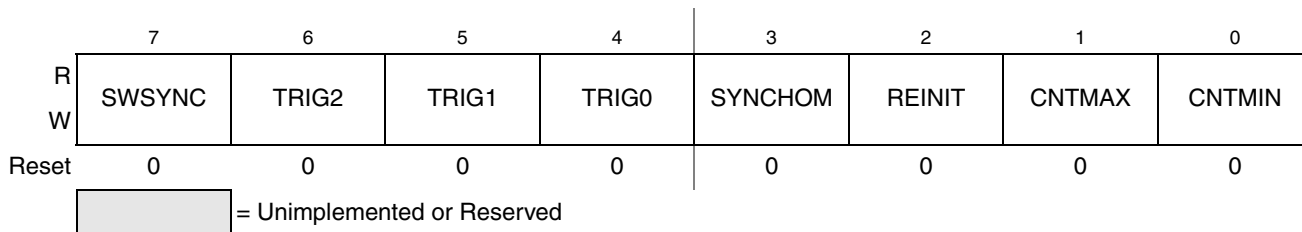
**Table 13-12. Fault Control Mode Selection**

FAULTM	Fault Control Mode
00	Fault control is disabled for all channels
01	Fault control is enabled for even channels only (channels 0, 2, 4, and 6) and the selected mode is the manual fault clearing
10	Fault control is enabled for all channels and the selected mode is the manual fault clearing
11	Fault control is enabled for all channels and the selected mode is the automatic fault clearing

### 13.3.11 FTM Synchronization Register (FTMxSYNC)

This read/write register configures the PWM synchronization.

A synchronization event can perform the synchronized update of FTMxMODH:FTMxMODL, FTMxCnVH:FTMxCnVL, and FTMxOUTMASK registers with the value of their write buffer and the FTM counter initialization.



**Figure 13-14. FTM Synchronization Register (FTMxSYNC)**

**Table 13-13. FTMxSYNC Field Descriptions**

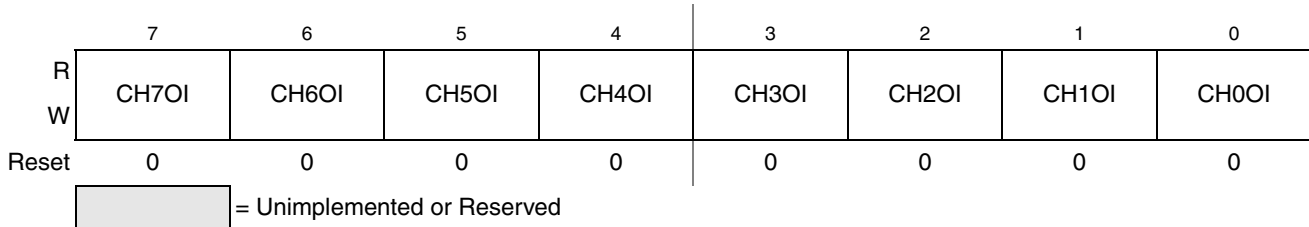
Field	Description
7 SWSYNC	PWM synchronization software trigger. SWSYNC bit selects the software trigger as the PWM synchronization trigger. The software trigger happens when a logic 1 is written to SWSYNC bit. 0 Software trigger is not selected. 1 Software trigger is selected.
6 TRIG2	PWM synchronization external trigger 2. TRIG2 bit selects external trigger 2 as the PWM synchronization trigger. External trigger 2 happens when the FTM detects a rising edge in the trigger 2 input signal. 0 External trigger 2 is not selected. 1 External trigger 2 is selected.
5 TRIG1	PWM synchronization external trigger 1. TRIG1 bit selects external trigger 1 as the PWM synchronization trigger. External trigger 1 happens when the FTM detects a rising edge in the trigger 1 input signal. 0 External trigger 1 is not selected. 1 External trigger 1 is selected.
4 TRIG0	PWM synchronization external trigger 0. TRIG0 bit selects external trigger 0 as the PWM synchronization trigger. External trigger 0 happens when the FTM detects a rising edge in the trigger 0 input signal. 0 External trigger 0 is not selected. 1 External trigger 0 is selected.
3 SYNCHOM	Output mask synchronization. SYNCHOM bit selects when the CHnOM bits in register FTMxOUTMASK are updated with the value of their write buffer. 0 CHnOM bits are updated with the value of the FTMxOUTMASK write buffer in all rising edges of the system clock. 1 CHnOM bits are updated with the value of the FTMxOUTMASK write buffer by the PWM synchronization.
2 REINIT	Reinitialization of the FTM by PWM synchronization ( <a href="#">Section 13.4.11, "PWM Synchronization"</a> ). REINIT bit determines if the FTM is initialized when the selected trigger for the PWM synchronization is detected. 0 FTM counter continues to count normally. 1 FTM counter is updated with its initial value when the selected trigger is detected.
1 CNTMAX	Maximum boundary cycle enable. The CNTMAX bit determines when the FTMxMODH:L and FTMxCnVH:L registers are updated with their write buffer contents following a PWM synchronization event. If CNTMAX is enabled, the registers are updated when the FTM counter reaches its maximum value FTMxMODH:L. 0 The maximum boundary cycle is disabled. 1 The maximum boundary cycle is enabled.
0 CNTMIN	Minimum boundary cycle enable. The CNTMIN bit determines when the FTMxMODH:L and FTMxCnVH:L registers are updated with their write buffer contents following a PWM synchronization event. If CNTMIN is enabled, the registers are updated when the FTM counter reaches its minimum value FTMxCNTINH:L. 0 The minimum boundary cycle is disabled. 1 The minimum boundary cycle is enabled.

## NOTE

- The software trigger (SWSYNC bit) and hardware triggers (TRIG0, TRIG1, and TRIG2 bits) have a potential conflict if used together. It is recommended using only hardware or software triggers but not both at the same time, otherwise unpredictable behavior is likely to happen.
- The selection of the boundary cycle (CNTMAX and CNTMIN bits) is intended to provide the update of FTMxCnVH:FTMxCnVL across all enabled channels simultaneously. The use of the boundary cycle selection together with TRIG0, TRIG1, or TRIG2 bits is likely to result in an unpredictable behavior.
- The PWMSYNC bit in the FTMxMODE register determines which type of trigger event controls the functions enabled by the FTMxSYNC register. Please refer to [Section 13.3.10, “FTM Features Mode Selection Register \(FTMxMODE\)”](#) for more information on the PWMSYNC bit.

### 13.3.12 FTM Initial State for Channels Output Register (FTMxOUTINIT)

This read/write register defines the value that is forced in the channels output by the initialization feature (when a logic 1 is written to the INIT bit in register FTMxMODE).



**Figure 13-15. FTM Initial State for Channels Output Register (FTMxOUTINIT)**

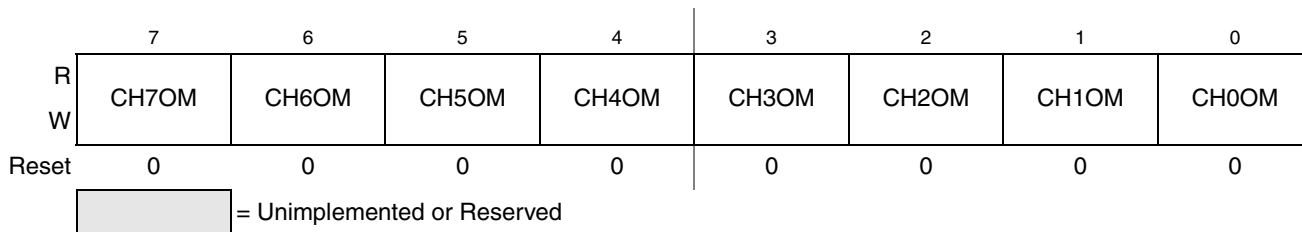
**Table 13-14. FTMxOUTINIT Field Descriptions**

Field	Description
7-0 CHnOI	Channel (n) output initialization value. CHnOI bit selects the value that is forced into channel (n) output when the initialization occurs. 0 The initialization value is the logical value 0. 1 The initialization value is the logical value 1.

### 13.3.13 FTM Output Mask Register (FTMxOUTMASK)

This read/write register provides a mask for each FTM channel. The mask of a channel determines if its output responds (that is, it is masked or not) when a match occurs. This feature is used for BLDC control where the PWM signal is presented to an electric motor at specific times to provide electronic commutation.

Any write to the FTMxOUTMASK register, stores the value into a write buffer. The register is updated with the value of its write buffer according to [Section 13.4.11, “PWM Synchronization.”](#)



**Figure 13-16. FTM Output Mask Register (FTMxOUTMASK)**

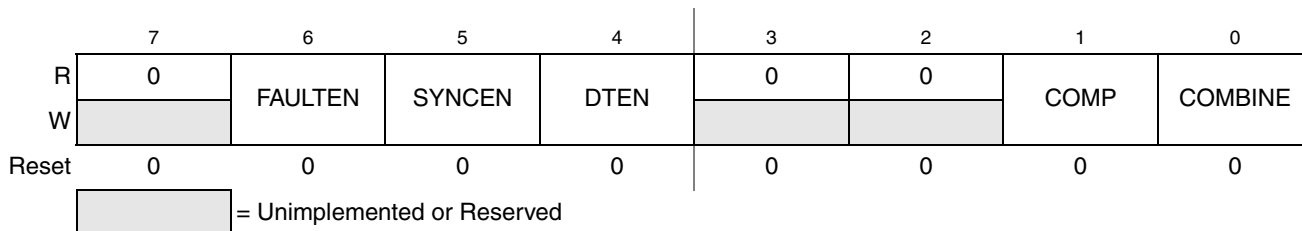
**Table 13-15. FTMxOUTMASK Field Descriptions**

Field	Description
7-0 CHnOM	Channel (n) output mask. CHnOM bit defines if the channel (n) output is masked (forced to its inactive state) or unmasked (it continues to operate normally). 0 Channel (n) output is not masked. It continues to operate normally. 1 Channel (n) output is masked. It is forced to its inactive state.

### 13.3.14 FTM Function for Linked Channels Register (FTMxCOMBINEm)

This read/write register contains the control bits used to configure the fault control, synchronization, deadtime, dual edge capture mode, complementary, and combine features of channels (n) and (n+1).

The FTMxCOMBINE0 register configures the control bits for channels 0 and 1; FTMxCOMBINE1 for channels 2 and 3; FTMxCOMBINE2 for channels 4 and 5; and FTMxCOMBINE3 for channels 6 and 7.



**Figure 13-17. FTM Function for Linked Channels Register (FTMxCOMBINEm)**

**Table 13-16. FTMxCOMBINEm Field Descriptions**

Field	Description
6 FAULTEN	Fault control enable. The FAULTEN bit enables the fault control in channels (n) and (n+1). FAULTEN is write protected, this bit can only be written if WPDIS = 1. 0 The fault control in this pair of channels is disabled. 1 The fault control in this pair of channels is enabled.
5 SYNCEN	Synchronization enable. The SYNCEN bit enables PWM synchronization of registers FTMxC(n)VH:FTMxC(n)VL and FTMxC(n+1)VH:FTMxC(n+1)VL. 0 The PWM synchronization in this pair of channels is disabled. 1 The PWM synchronization in this pair of channels is enabled.
4 DTEN	Deadtime enable. The DTEN bit enables the deadtime insertion in the channels (n) and (n+1). DTEN is write protected, this bit can only be written if WPDIS = 1. 0 The deadtime insertion in this pair of channels is disabled. 1 The deadtime insertion in this pair of channels is enabled.

Field	Description
1 COMP	Complement of channel (n). The COMP bit enables complementary mode for the combined channels. In the complementary mode channel (n+1) output is the inverse of channel (n) output. COMP is write protected, this bit can only be written if WPDIS = 1. 0 The channel (n+1) output is the same as the channel (n) output. 1 The channel (n+1) output is the complement of the channel (n) output.
0 COMBINE	Combine channels (n) and (n+1). The COMBINE bit enables the combine feature for channels (n) and (n+1). COMBINE is write protected, this bit can only be written if WPDIS = 1. 0 Channels (n) and (n+1) are independent. 1 Combine mode is enabled for channels (n) and (n+1).

### NOTE

The channel (n) is the even channel and the channel (n+1) is the odd channel of a pair of channels.

## 13.3.15 FTM Deadtime Insertion Control Register (FTMxDEADTIME)

This read/write register selects the deadtime prescaler factor and deadtime value. All FTM channels use this clock prescaler and this deadtime value for the deadtime insertion.

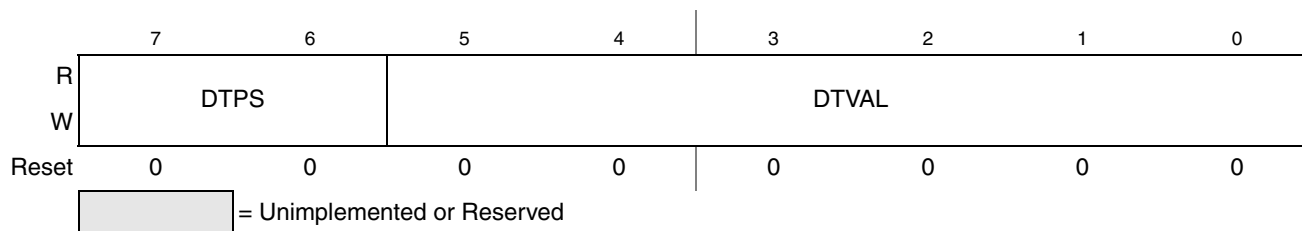


Figure 13-18. FTM Deadtime Insertion Control Register (FTMxDEADTIME)

Table 13-17. FTMxDEADTIME Field Descriptions

Field	Description
7–6 DTPS	Deadtime prescaler value. DTPS[1:0] bits select the division factor of the system clock (as shown in <a href="#">Table 13-18</a> ). This prescaled clock is used by the deadtime counter. DTPS[1:0] bits are write protected, these bits can only be written if WPDIS = 1.
5–0 DTVAL	Deadtime value. DTVAL[5:0] bits select the deadtime value using the deadtime prescaled clock (as shown in <a href="#">Table 13-19</a> ). The deadtime insertion in all channels is disabled when DTVAL[5:0] is zero. DTVAL[5:0] bits are write protected, these bits can only be written if WPDIS = 1.

Table 13-18. Deadtime Prescale Factor Selection

DTPS	System Clock Divided-by
0X	1
10	4
11	16

**Table 13-19. Deadtime Value (Number of System Clock Periods)**

DTVAL	DTPS = 0X	DTPS = 10	DTPS = 11
0	Deadtime insertion is disabled		
1	1	4	16
2	2	8	32
3	3	12	48
4	4	16	64
5	5	20	80
6	6	24	96
7	7	28	112
8	8	32	128
9	9	36	144
10	10	40	160
11	11	44	176
12	12	48	192
13	13	52	208
14	14	56	224
15	15	60	240
16	16	64	256
17	17	68	272
18	18	72	288
19	19	76	304
20	20	80	320
21	21	84	336
22	22	88	352
23	23	92	368
24	24	96	384
25	25	100	400
26	26	104	416
27	27	108	432
28	28	112	448
29	29	116	464
30	30	120	480
31	31	124	496
32	32	128	512
33	33	132	528
34	34	136	544

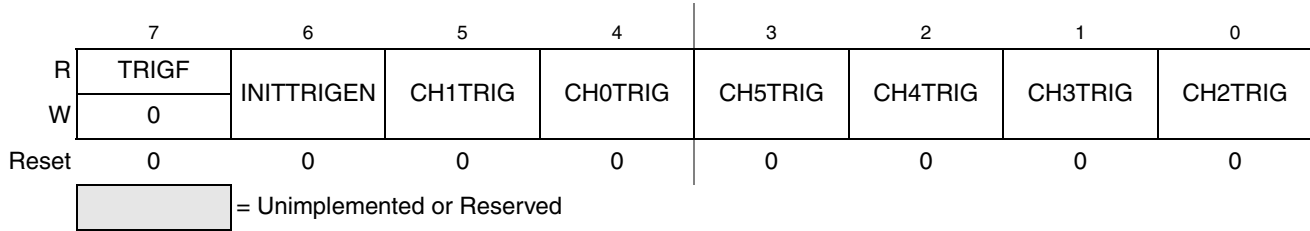
**Table 13-19. Deadtime Value (Number of System Clock Periods) (continued)**

DTVAL	DTPS = 0X	DTPS = 10	DTPS = 11
35	35	140	560
36	36	144	576
37	37	148	592
38	38	152	608
39	39	156	624
40	40	160	640
41	41	164	656
42	42	168	672
43	43	172	688
44	44	176	704
45	45	180	720
46	46	184	736
47	47	188	752
48	48	192	768
49	49	196	784
50	50	200	800
51	51	204	816
52	52	208	832
53	53	212	848
54	54	216	864
55	55	220	880
56	56	224	896
57	57	228	912
58	58	232	928
59	59	236	944
60	60	240	960
61	61	244	976
62	62	248	992
63	63	252	1008

### 13.3.16 FTM External Trigger Register (FTMxEXTTRIG)

This read/write register indicates when a channel trigger was generated, enables the generation of a trigger when the FTM counter is equal to its initial value, and selects which channels are used in the generation of the channel triggers.

Channels 6 and 7 are not used to generate channel triggers.



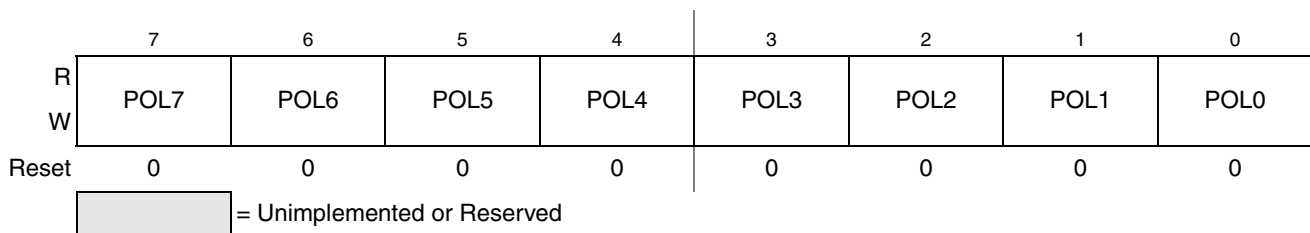
**Figure 13-19. FTM External Trigger Register (FTMxEXTTRIG)**

**Table 13-20. FTMxEXTTRIG Field Descriptions**

Field	Description
7 TRIGF	Channel trigger flag. This read/write bit is set when a channel trigger is generated. Clear TRIGF bit by reading FTMxEXTTRIG while TRIGF is set and then writing a logic 0 to TRIGF. If another channel trigger is generated before the clearing sequence is completed, the sequence is reset so TRIGF remains set after the clear sequence is completed for the earlier TRIGF. Reset clears TRIGF. Writing a logic 1 to TRIGF has no effect. 0 No channel trigger was generated. 1 A channel trigger was generated.
6 INITTRIGEN	Initialization trigger enable. This read/write bit enables the generation of the trigger when the FTM counter is equal to its initial value. 0 The generation of initialization trigger is disabled. 1 The generation of initialization trigger is enabled.
5-0 CHjTRIG where j = 1, 0, 5, 4, 3, 2	Channel j trigger enable. These read/write bits enable the generation of a channel trigger when the FTM counter is equal to the FTMxC(j)VH:FTMxC(j)VL registers. Several FTM channels can be selected to generate multiple triggers in one PWM period. 0 The generation of channel (j) trigger is disabled. 1 The generation of channel (j) trigger is enabled.

### 13.3.17 FTM Channels Polarity Register (FTMxPOL)

This read/write register defines the output polarity of the FTM channels.



**Figure 13-20. FTM Channels Polarity Register (FTMxPOL)**

**Table 13-21. FTMxPOL Field Descriptions**

Field	Description
7-0 POLn	Channel (n) polarity. The POLn bit defines the polarity of the channel (n) output. POLn is write protected, this bit can only be written if WPDIS = 1. 0 The channel (n) polarity is high (the active state is logical one and the inactive state is logical zero). 1 The channel (n) polarity is low (the active state is logical zero and the inactive state is logical one).



## NOTE

The safe value that is driven in the channel (n) output when the fault control is enabled and a fault condition is detected is the inactive state of the channel (n) polarity. The channel (n) safe value is the value of its POL bit.

### 13.3.18 FTM Fault Mode Status Register (FTMxFMS)

This read/write register contains the fault detection flags, write protection enable bit, and the logic OR of the enable fault inputs.

	7	6	5	4	3	2	1	0
R	FAULTF	WPEN	FAULTIN	0	FAULTF3	FAULTF2	FAULTF1	FAULTF0
W	0				0	0	0	0
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 13-21. FTM Fault Mode Status Register (FTMxFMS)**

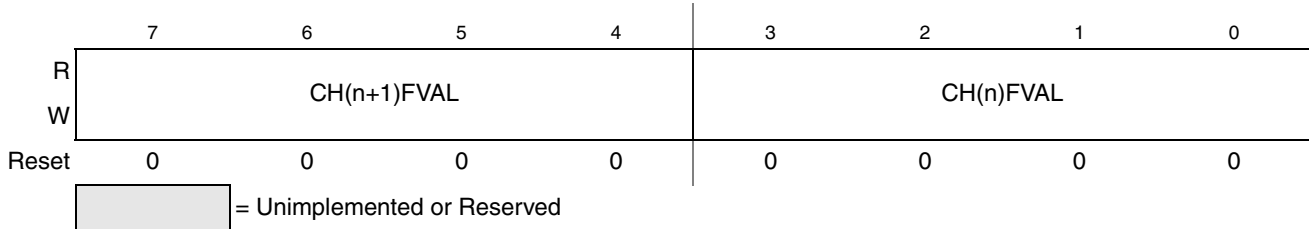
**Table 13-22. FTMxFMS Field Descriptions**

Field	Description
7 FAULTF	Fault detection flag. The FAULTF bit is the logic OR of the FAULTFn bits, where n = 3, 2, 1, 0. Clear FAULTF by reading the FTMxFMS register while FAULTF is set and then writing a logic 0 to FAULTF. If another fault condition is detected in an enabled fault input before the clearing sequence is completed, the sequence is reset so FAULTF remains set after the clearing sequence is completed for the earlier fault condition. FAULTF is also cleared when FAULTFn bits are cleared individually. Reset clears FAULTF. Writing a logic 1 to FAULTF has no effect. 0 No fault condition was detected. 1 A fault condition was detected
6 WPEN	Write protection enable. When write protection is enabled (WPEN = 1), write protected bits can not be written. When write protection is disabled (WPEN = 0), write protected bits can be written. The WPEN bit is the negation of the WPDIS bit. WPEN is set when 1 is written to it. WPEN is cleared when WPEN bit is read as a logic 1 and then 1 is written to WPDIS. Write 0 to WPEN has no effect. 0 Write protection is disabled. 1 Write protection is enabled.
5 FAULTIN	Fault inputs. The FAULTIN bit is the logic OR of the enabled fault input after its filter (if its filter is enabled) when fault control is enabled. Reset clears FAULTIN. Writing a logic 0 or a logic 1 to FAULTIN has no effect. 0 The value of the fault input is logic 0. 1 The value of the fault input is logic 1.
3-0 FAULTFn where n = 3, 2, 1, 0	Fault detection flag n. The FAULTFn bit is set when fault control is enabled, the fault input n is enabled and a fault condition is detected in the fault input n. Clear FAULTFn by reading the FTMxFMS register while FAULTFn is set and then writing a logic 0 to FAULTFn when the fault input n is low. If another fault condition is detected at fault input n before the clearing sequence is completed, the sequence is reset so FAULTFn remains set after the clearing sequence is completed for the earlier fault condition. FAULTFn bit is also cleared when FAULTF bit is cleared. Reset clears FAULTFn. Writing a logic 1 to FAULTFn has no effect. 0 No fault condition was detected in the fault input n. 1 A fault condition was detected in the fault input n.

### 13.3.19 FTM Input Capture Filter Control Register (FTMxFILTERm)

This read/write register selects the filter value for the inputs of channels (n) and (n+1).

The FTMxFILTER0 register selects the filter value for the inputs of channels 0 and 1; the FTMxFILTER1 register selects the filter value for the inputs of channels 2 and 3. Channels 4, 5, 6 and 7 do not have an input filter.



**Figure 13-22. FTM Input Capture Filter Control Register (FTMxFILTERm)**

**Table 13-23. FTMxFILTERm Field Descriptions**

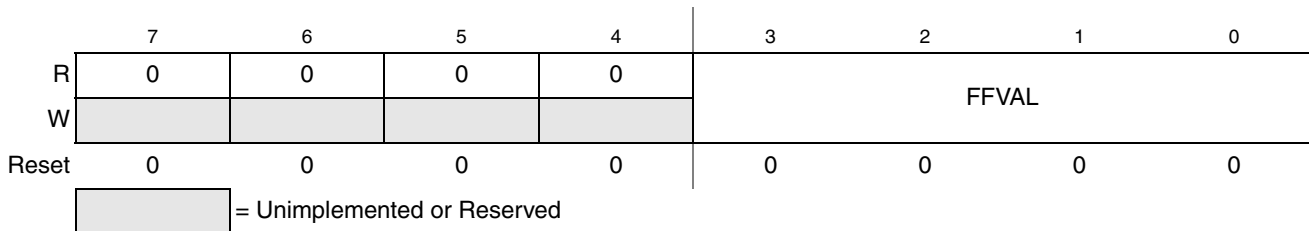
Field	Description
7-4 CH(n+1)FVAL	Channel (n+1) input filter. CH(n+1)FVAL[3:0] bits select the filter value for the channel (n+1) input. The channel (n+1) filter is disabled when CH(n+1)FVAL[3:0] is zero.
3-0 CH(n)FVAL	Channel (n) input filter. CH(n)FVAL[3:0] bits select the filter value for the channel (n) input. The channel (n) filter is disabled when CH(n)FVAL[3:0] is zero.

#### NOTE

Writing to this register has immediate effect and must only be done when the input capture modes of channels (n) and (n+1) are disabled. Failure to do this could result in a missing valid signal.

### 13.3.20 FTM Fault Input Filter Control Register (FTMxFLTFILTER)

This read/write register selects the filter value for the fault inputs.



**Figure 13-23. FTM Fault Input Filter Control Register (FTMxFLTFILTER)**

**Table 13-24. FTMxFLTFILTER Field Descriptions**

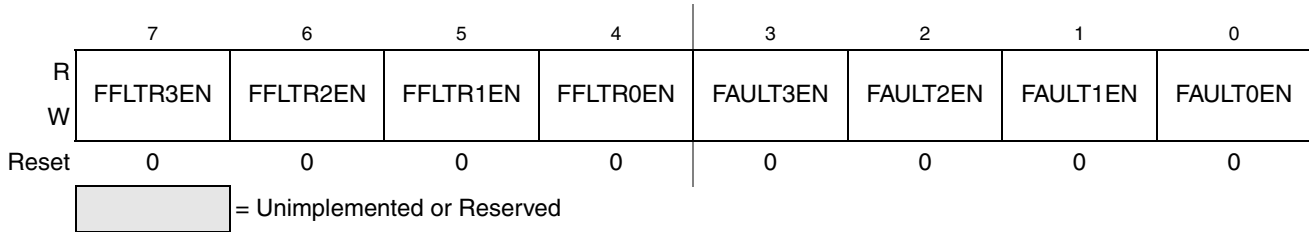
Field	Description
3-0 FFVAL	Fault input filter. FFVAL[3:0] bits select the filter value for the fault inputs. The fault filter is disabled when FFVAL[3:0] is zero.

## NOTE

Writing to this register has immediate effect and must only be done when the fault control or the fault input is disabled. Failure to do so could result in a missing fault detection.

### 13.3.21 FTM Fault Control Register (FTMxFLTCTRL)

This read/write register selects the fault inputs and enables the fault input filter.



**Figure 13-24. FTM Fault Control Register (FTMxFLTCTRL)**

**Table 13-25. FTMxFLTCTRL Field Descriptions**

Field	Description
7-4 FFLTRnEN where n = 3, 2, 1, 0	Fault input n filter enable. This read/write bit enables the filter for the fault input n. FFLTRnEN is write protected, this bit can only be written if WPDIS = 1. 0 Fault input n filter is disabled. 1 Fault input n filter is enabled.
3-0 FAULTnEN where n = 3, 2, 1, 0	Fault input n enable. This read/write bit enables the fault input n. FAULTnEN is write protected, this bit can only be written if WPDIS = 1. 0 Fault input n is disabled. 1 Fault input n is enabled.

## 13.4 Functional Description

The following sections describe the FTM features.

The notation used in this document to represent the counters and the generation of the signals is shown in [Figure 13-25](#).

Channel (n) - high-true EPWM

PS[2:0] = 001  
 FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x0004  
 FTMxCnVH:L = 0x0002

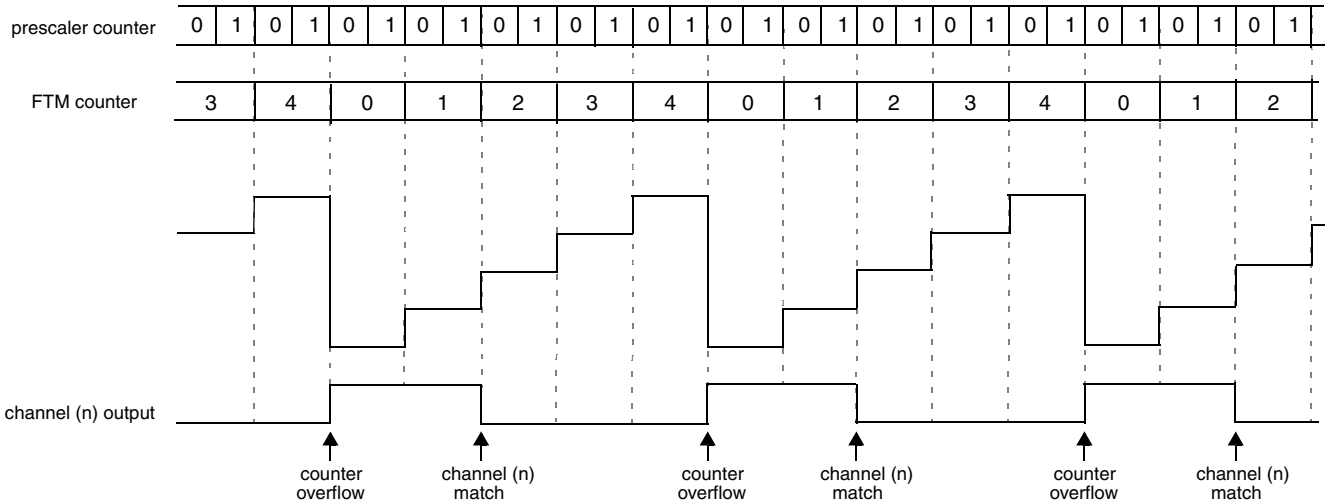


Figure 13-25. Notation Used in the FTM Block Guide

## 13.4.1 Clock Source

FTM module has only one clock domain that is the system clock.

### 13.4.1.1 Counter Clock Source

The CLKS[1:0] bits in the FTMxSC register select one of three possible clock sources for the FTM counter or disable the FTM counter, see [Table 13-6](#). After any MCU reset, CLKS[1:0] = 0:0 so no clock source is selected.

The CLKS[1:0] bits may be read or written at any time. Disabling the FTM counter by writing 0:0 to the CLKS[1:0] bits does not affect the FTM counter value or other registers.

The fixed frequency clock is an alternative clock source for the FTM counter that allows the selection of a clock other than the system clock or an external clock. This clock input is defined by chip integration. Refer the chip specific documentation for further information. Due to FTM hardware implementation limitations, the frequency of the fixed frequency clock must not exceed the system clock frequency. The fixed frequency clock has no limitations for lower frequency operation.

The external clock passes through a synchronizer clocked by the system clock to assure that counter transitions are properly aligned to system clock transitions. Therefore, to meet Nyquist criteria considering also jitter, the frequency of the external clock source must not exceed 1/4 of the system clock frequency.

## 13.4.2 Prescaler

The selected counter clock source passes through a prescaler that is a 7-bit counter. The value of the prescaler is selected by the PS[2:0] bits (Table 13-7). Figure 13-26 shows an example of the prescaler counter and FTM counter.

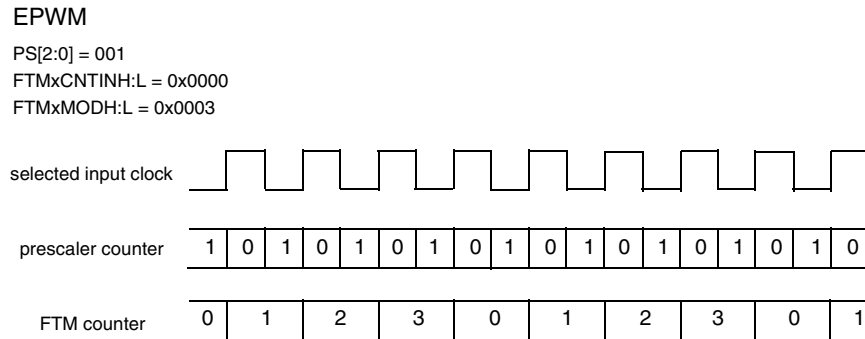


Figure 13-26. Example of the Prescaler Counter

## 13.4.3 Counter

The FTM has a 16-bit counter that is used by the channels either for input or output modes. The FTM counter clock is the selected clock divided by the prescaler (Section 13.4.2, “Prescaler”).

The FTM counter has two modes of operation: up or up-down counter according to the CPWMS bit.

### 13.4.3.1 Up Counting

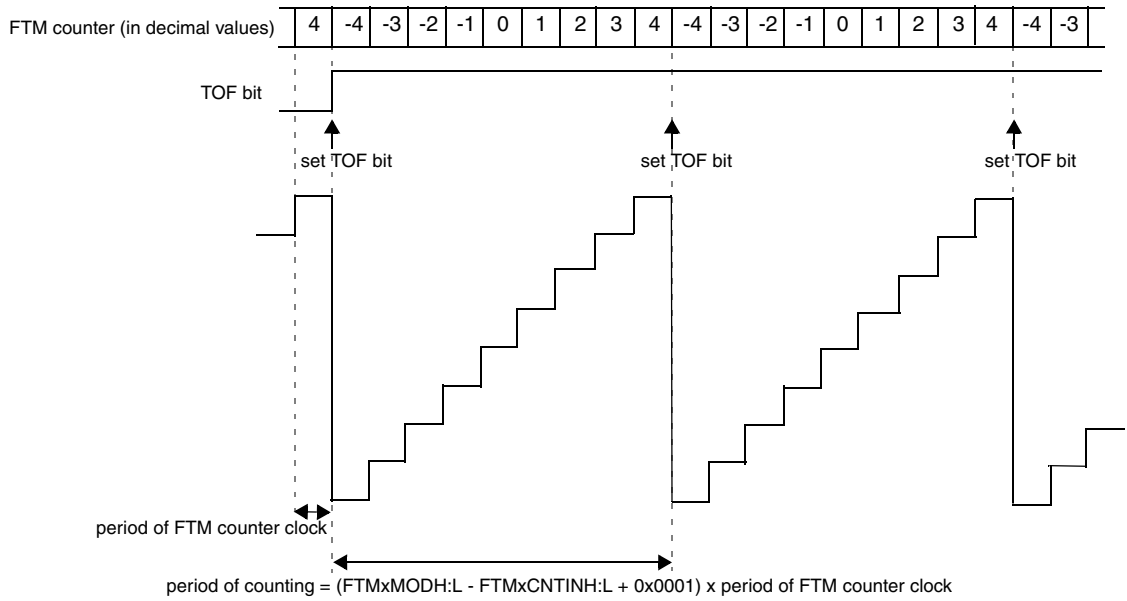
Up counting is selected when (CPWMS = 0).

FTMxCNTINH:FTMxCNTINL defines the starting value of the count and FTMxMODH:FTMxMODL defines the final value of the count (Figure 13-27). The value of FTMxCNTINH:FTMxCNTINL is loaded into the FTM counter, and the counter increments until the value of FTMxMODH:FTMxMODL is reached, at which point the counter is reloaded with the contents of FTMxCNTINH:FTMxCNTINL.

The FTM period when using up counting is  $(\text{FTMxMODH:FTMxMODL} - \text{FTMxCNTINH:FTMxCNTINL} + 0x0001) \times \text{period of the FTM counter clock}$ .

The TOF bit is set when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxCNTINH:FTMxCNTINL.

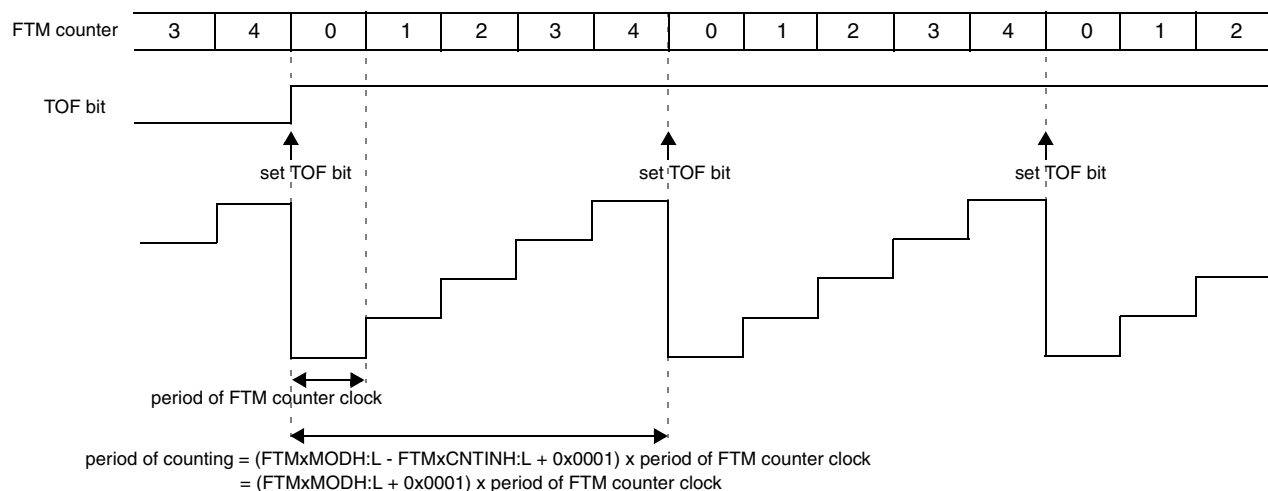
FTM counting is up (CPWMS = 0)  
 FTMxCNTINH:L = 0xFFFC (in two's complement is equal to -4)  
 FTMxMODH:L = 0x0004



**Figure 13-27. Example of FTM Up and Signed Counting**

If (FTMxCNTINH:FTMxCNTINL = 0x0000), the FTM counting is equivalent to TPM up counting (that is, up and unsigned counting) (Figure 13-28). If (FTMxCNTINH[7] = 1), then the initial value of the FTM counter is a negative number in two's complement, so the FTM counting is up and signed. Conversely if (FTMxCNTINH[7] = 0 and FTMxCNTINH:L ≠ 0x0000), then the initial value of the FTM counter is a positive number, so the FTM counting is up and unsigned.

FTM counting is up (CPWMS = 0)  
 FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x0004

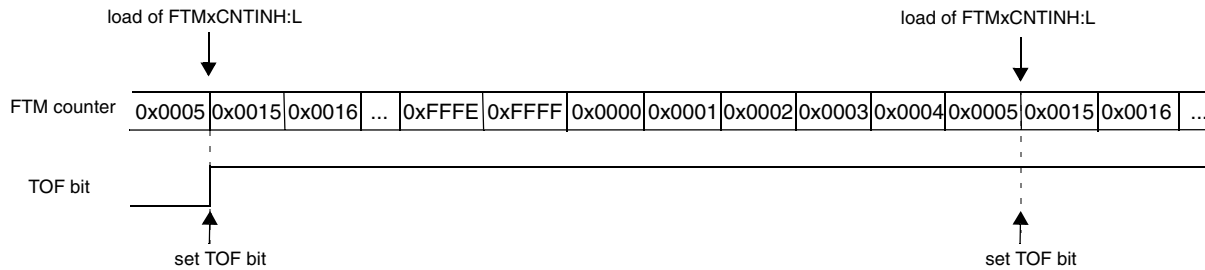


**Figure 13-28. Example of FTM Up Counting with FTMxCNTINH:FTMxCNTINL = 0x0000**

#### NOTE

- FTM operation is only valid when the value of the FTMxCNTINH:FTMxCNTINL registers is less than the value of the FTMxMODH:FTMxMODL registers (either in the unsigned counting or signed counting). It is the responsibility of the software to ensure that the values in the FTMxCNTINH:FTMxCNTINL and FTMxMODH:FTMxMODL registers meet this requirement. Any values of FTMxCNTINH:FTMxCNTINL and FTMxMODH:FTMxMODL that do not satisfy this criteria can result in unpredictable behavior.
- FTMxMODH:FTMxMODL = FTMxCNTINH:FTMxCNTINL is a redundant condition. In this case, the FTM counter is always equal to FTMxMODH:FTMxMODL and the TOF bit is set in each rising edge of the FTM counter clock.
- When FTMxMODH:FTMxMODL = 0x0000, FTMxCNTINH:FTMxCNTINL = 0x0000 (for example after reset), and FTMEN = 1, the FTM counter remains stopped at 0x0000 until a non-zero value is written into the FTMxMODH:FTMxMODL or FTMxCNTINH:FTMxCNTINL registers.
- Setting FTMxCNTINH:L to be greater than the value of FTMxMODH:L is not recommended as this unusual setting may make the FTM operation difficult to comprehend. However, there is no restriction on this configuration, and an example is shown in [Figure 13-29](#).

FTM counting is up (CPWMS = 0)  
 FTMxMODH:L = 0x0005  
 FTMxCNTINH:L = 0x0015



**Figure 13-29. Example of Up Counting When the Value of FTMxCNTINH:FTMxCNTINL Registers Is Greater Than the Value of FTMxMODH:FTMxMODL Registers**

### 13.4.3.2 Up-Down Counting

Up-down counting is selected when (FTMEN = 0) and (CPWMS = 1).

FTMxCNTINH:FTMxCNTINL defines the starting value of the count and FTMxMODH:FTMxMODL defines the final value of the count (Figure 13-30). The value of FTMxCNTINH:FTMxCNTINL is loaded into the FTM counter, and the counter increments until the value of FTMxMODH:FTMxMODL is reached, at which point the counter is decremented until it returns to the value of FTMxCNTINH:FTMxCNTINL and the up-down counting restarts.

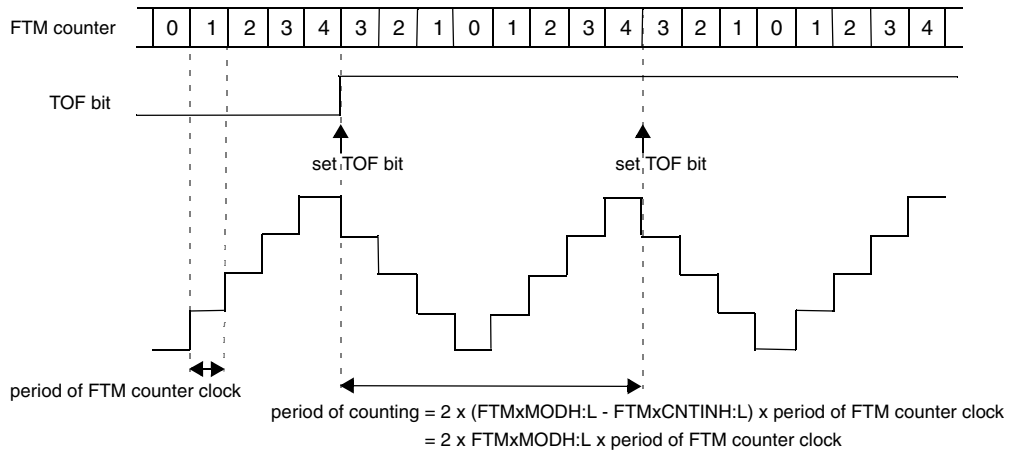
The FTM period when using up-down counting is  $2 \times (\text{FTMxMODH:FTMxMODL} - \text{FTMxCNTINH:FTMxCNTINL}) \times \text{period of the FTM counter clock}$ .

The TOF bit is set when the FTM counter changes from FTMxMODH:FTMxMODL to (FTMxMODH:FTMxMODL - 1).

If (FTMxCNTINH:FTMxCNTINL = 0x0000), the FTM counting is equivalent to TPM up-down counting (that is, up-down and unsigned counting) (Figure 13-30).



FTM counting is up-down (CPWMS = 1)  
 FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x0004



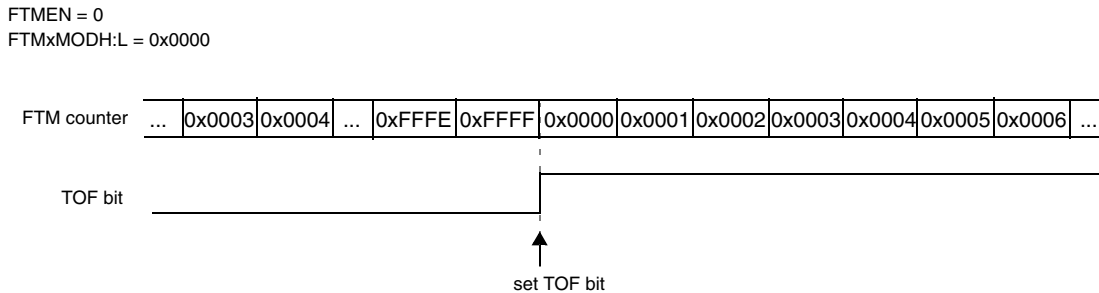
**Figure 13-30. Example of Up-Down Counting When FTMxCNTINH:FTMxCNTINL = 0x0000**

**NOTE**

- The up-down counting is only available when (FTMEN = 0) and (FTMxCNTINH:FTMxCNTINL = 0x0000).
- The configuration with (FTMEN = 1) or (FTMxCNTINH:FTMxCNTINL ≠ 0x0000) when (CPWMS = 1) is not recommended and its results are not guaranteed.

**13.4.3.3 Free Running Counter**

If (FTMEN = 0) and (FTMxMODH:FTMxMODL = 0x0000 or FTMxMODH:FTMxMODL = 0xFFFF), the FTM counter is a free running counter. In this case, the FTM counter runs free from 0x0000 through 0xFFFF and the TOF bit is set when the FTM counter changes from 0xFFFF to 0x0000 (Figure 13-31).



**Figure 13-31. Example When the FTM Counter Is a Free Running**

If (FTMEN = 1), (CPWMS = 0), (FTMxCNTINH:FTMxCNTINL = 0x0000), and (FTMxMODH:FTMxMODL = 0xFFFF), the FTM counter is a free running counter. In this case, the FTM counter runs free from 0x0000 through 0xFFFF and the TOF bit is set when the FTM counter changes from 0xFFFF to 0x0000.

### 13.4.3.4 Counter Reset

Any write to FTMxCNTH or FTMxCNTL register resets the FTM counter to the value in the FTMxCNTINH:FTMxCNTINL registers and the channels output to their initial value (except for channels in output compare mode).

PWM synchronization can also be used to force the value of FTMxCNTINH:FTMxCNTINL into the FTM counter and the channels output to their initial value (except for channels in output compare mode) (please see [Section 13.4.11, “PWM Synchronization”](#)){ftm\_ipi\_slv\_pwm\_sync}.

## 13.4.4 Input Capture Mode

The input capture mode is selected when (COMBINE = 0), (CPWMS = 0), (MSnB:MSnA = 0:0), and (ELSnB:ELSnA ≠ 0:0).

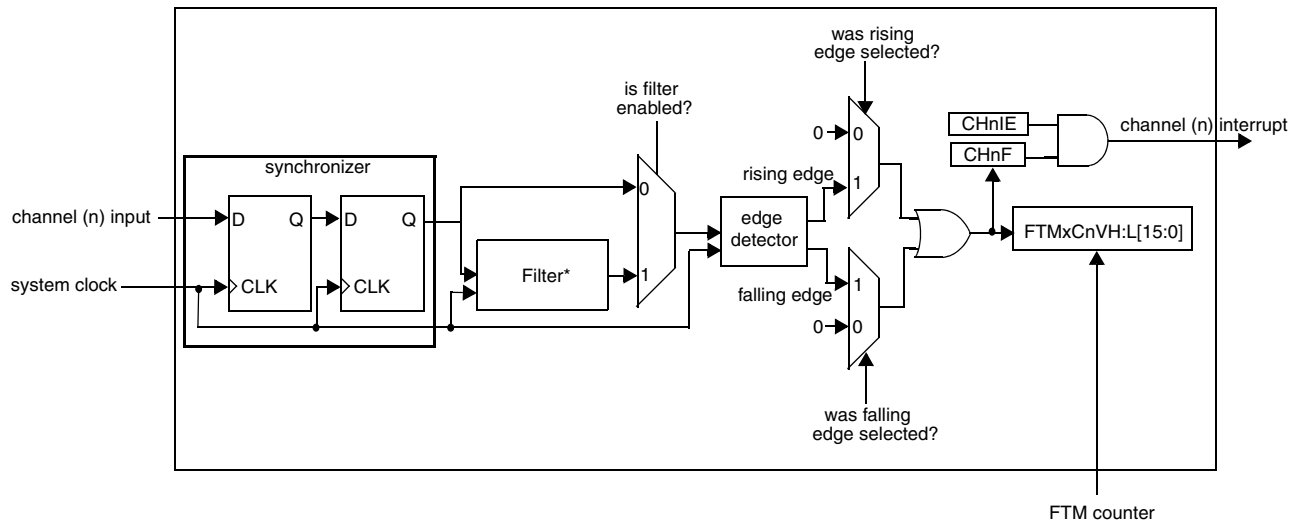
When a selected edge occurs on the channel input, the current value of the FTM counter is captured into the FTMxCnVH:FTMxCnVL registers, at the same time the CHnF bit is set and the channel interrupt is generated if enabled by CHnIE = 1 ([Figure 13-32](#)).

When a channel is configured for input capture, the FTMxCHn pin is an edge-sensitive input. ELSnB:ELSnA control bits determine which edge, falling or rising, triggers input-capture event. Note that the maximum frequency for the channel input signal to be detected correctly is system clock divided by 4, which is required to meet Nyquist criteria for signal sampling.

When either half of the 16-bit capture register (FTMxCnVH:FTMxCnVL) is read, the other half is latched into a buffer to support coherent 16-bit access in big-endian or little-endian order. This read coherency mechanism can be manually reset by writing to FTMxCnSC register.

Writes to the FTMxCnVH:FTMxCnVL registers are ignored in input capture mode.

While in BDM, the input capture function works as configured. When a selected edge event occurs, the FTM counter value (which is frozen because of BDM) is captured into the FTMxCnVH:FTMxCnVL registers and the CHnF bit is set.



\* Filtering function is only available in the inputs of channel 0, 1, 2, and 3

**Figure 13-32. Input Capture Mode**

If the channel input does not have a filter enabled, then the input signal is always delayed 3 rising edges of the system clock (two rising edges to the synchronizer plus one more rising edge to the edge detector). In other words, the CHnF bit is set on the third rising edge of the system clock after a valid edge occurs on the channel input.

#### NOTE

- Input capture mode is only available with (FTMxCNTINH:FTMxCNTINL = 0x0000).
- Input capture mode with (FTMxCNTINH:FTMxCNTINL ≠ 0x0000) is not recommended and its results are not guaranteed.

#### 13.4.4.1 Filter for Input Capture Mode

The filter function is only available on channels 0, 1, 2, and 3.

Firstly the input signal is synchronized by the system clock (synchronizer block in Figure 13-32). Following synchronization, the input signal enters the filter block (Figure 13-33). When there is a state change in the input signal, the 5-bit counter is reset and starts counting up. As long as the new state is stable on the input, the counter continues to increment. If the 5-bit counter overflows (the counter exceeds the value of the CHnFVAL[3:0] bits), the state change of the input signal is validated. It is then transmitted as a pulse edge to the edge detector.

If the opposite edge appears on the input signal before validation (counter overflow), the counter is reset. At the next input transition, the counter starts counting again. Any pulse that is shorter than the minimum value selected by CHnFVAL[3:0] bits (× 4 system clocks) is regarded as a glitch and is not passed on to the edge detector. A timing diagram of the input filter is shown in Figure 13-34.

The filter function is disabled when CHnFVAL[3:0] bits are zero. In this case, the input signal is delayed 3 rising edges of the system clock. If (CHnFVAL[3:0] ≠ 0000), then the input signal is delayed by the

minimum pulse width ( $CHnFVAL[3:0] \times 4$  system clocks) plus a further 4 rising edges of the system clock (two rising edges to the synchronizer, one rising edge to the filter output plus one more to the edge detector). In other words,  $CHnF$  is set  $(4 + 4 \times CHnFVAL[3:0])$  system clock periods after a valid edge occurs on the channel input.

The clock for the 5-bit counter in the channel input filter is the system clock divided by 4.

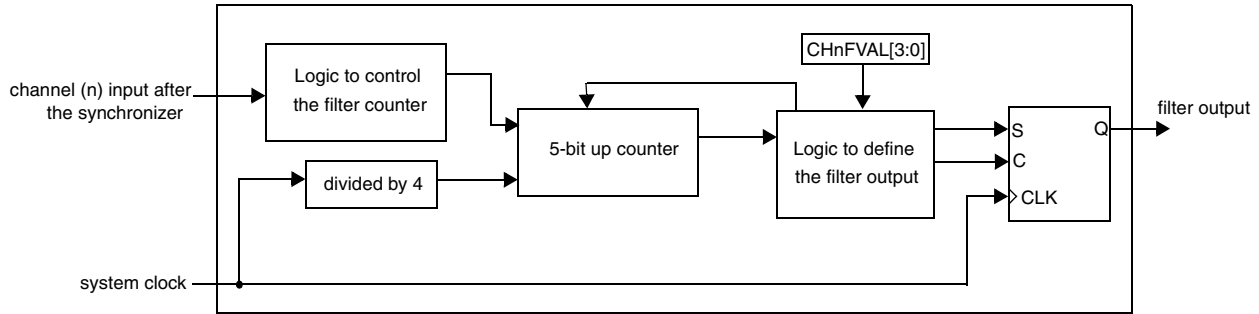


Figure 13-33. Channel Input Filter

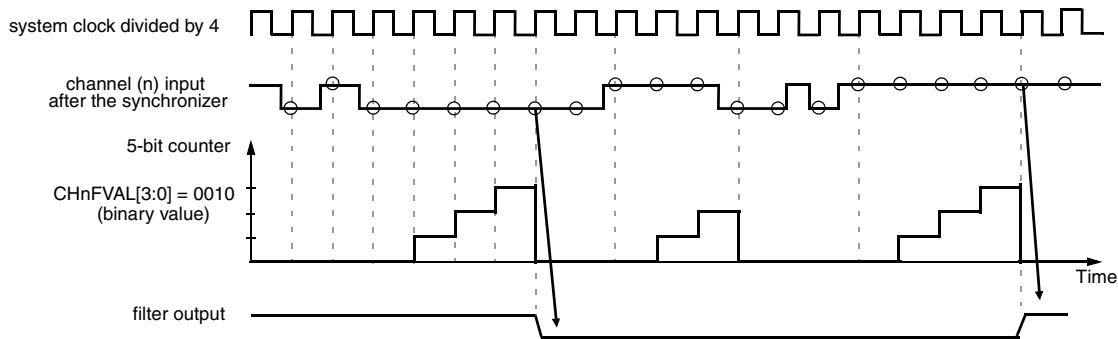


Figure 13-34. Channel Input Filter Example

### 13.4.5 Output Compare Mode

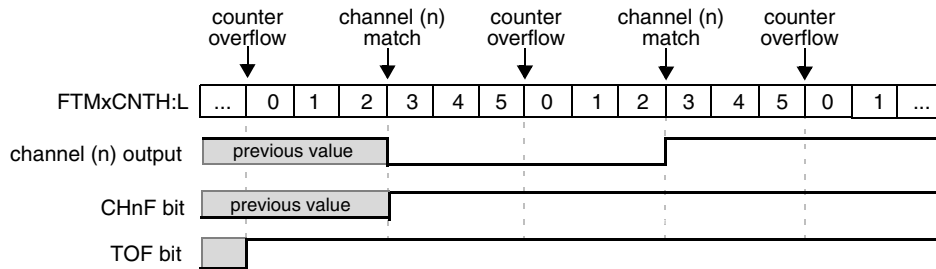
The output compare mode is selected when  $(COMBINE = 0)$ ,  $(CPWMS = 0)$ , and  $(MSnB:MSnA = 0:1)$ .

In output compare mode, the FTM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter matches the value in the  $FTMxCnVH:FTMxCnVL$  registers of an output compare channel, the channel (n) output can be set, cleared, or toggled.

When a channel is initially configured to toggle mode, the previous value of the channel output is held until the first output compare event occurs.

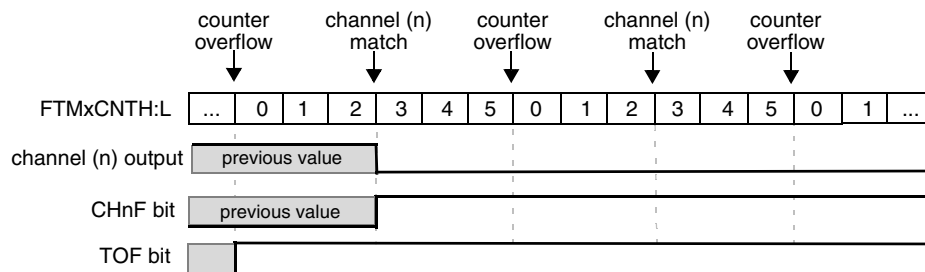
The  $CHnF$  bit is set and the channel (n) interrupt is generated (if  $CHnIE = 1$ ) at the channel (n) match ( $FTM$  counter =  $FTMxCnVH:FTMxCnVL$ ).

FTMxMODH:L = 0x0005  
 FTMxCnVH:L = 0x0003



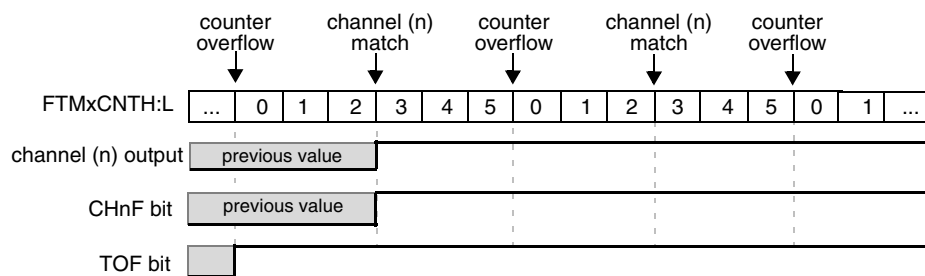
**Figure 13-35. Example of the Output Compare Mode when the Match Toggles the Channel Output**

FTMxMODH:L = 0x0005  
 FTMxCnVH:L = 0x0003



**Figure 13-36. Example of the Output Compare Mode when the Match Clears the Channel Output**

FTMxMODH:L = 0x0005  
 FTMxCnVH:L = 0x0003



**Figure 13-37. Example of the Output Compare Mode when the Match Sets the Channel Output**

It is possible to use the output compare mode with (ELSnB:ELSnA = 0:0). In this case, when the counter reaches the value in the FTMxCnVH:FTMxCnVL registers, the CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1), however the channel (n) output is not modified and controlled by FTM.

**NOTE**

- Output compare mode is only available with (FTMxCNTINH:FTMxCNTINL = 0x0000).

- Output compare mode with (FTMxCNTINH:FTMxCNTINL  $\neq$  0x0000) is not recommended and its results are not guaranteed.

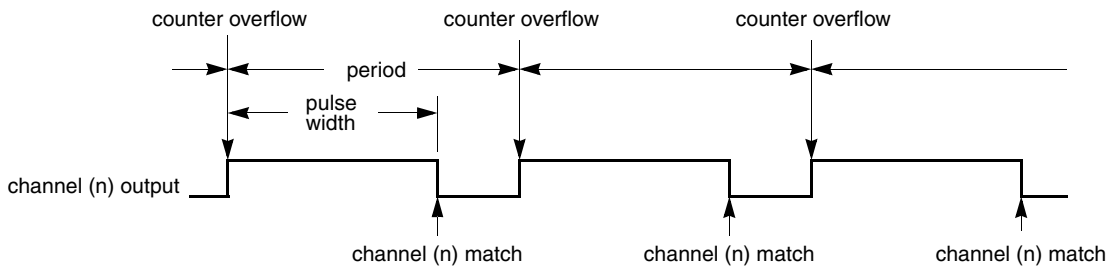
### 13.4.6 Edge-Aligned PWM (EPWM) Mode

The edge-aligned mode is selected when (COMBINE = 0), (CPWMS = 0), and (MSnB = 1).

The EPWM period is determined by (FTMxMODH:FTMxMODL – FTMxCNTINH:FTMxCNTINL + 0x0001) and the pulse width (duty cycle) is determined by (FTMxCnVH:FTMxCnVL – FTMxCNTINH:FTMxCNTINL).

The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = FTMxCnVH:FTMxCnVL), that is, at the end of the pulse width.

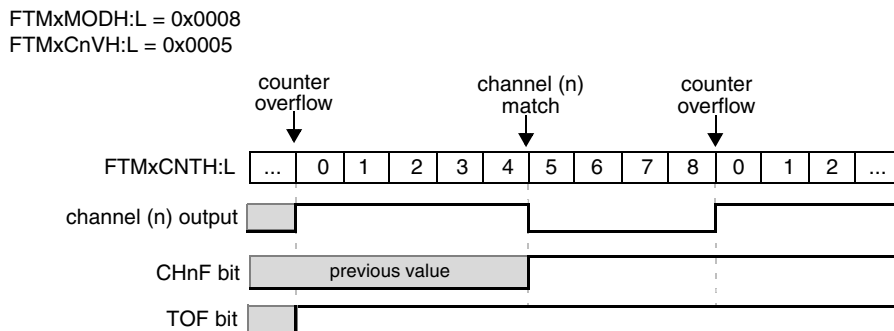
This type of PWM signal is called edge-aligned because the leading edges of all PWM signals are aligned with the beginning of the period, which is the same for all channels within an FTM.



**Figure 13-38. EPWM Period and Pulse Width with ELSnB:ELSnA = 1:0**

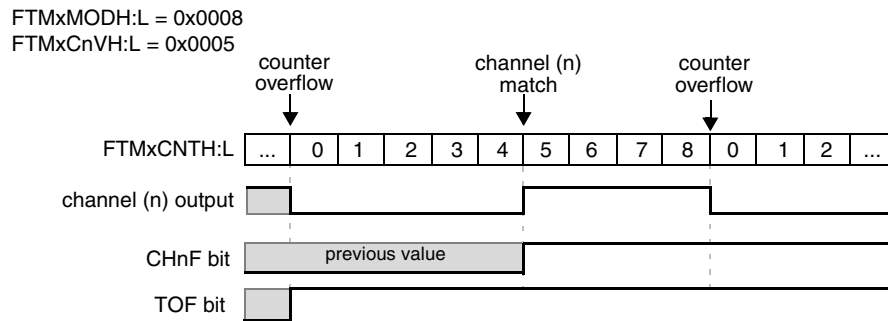
If (ELSnB:ELSnA = 0:0) when the counter reaches the value in the FTMxCnVH:FTMxCnVL registers, the CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1), however the channel (n) output is not controlled by FTM.

If (ELSnB:ELSnA = 1:0), then the channel (n) output is forced high at the counter overflow (when the FTMxCNTINH:FTMxCNTINL register contents are loaded into the FTM counter), and it is forced low at the channel (n) match (when the FTM counter = FTMxCnVH:FTMxCnVL) (Figure 13-39).



**Figure 13-39. EPWM Signal with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = X:1), then the channel (n) output is forced low at the counter overflow (when the FTMxCNTINH:FTMxCNTINL register contents are loaded into the FTM counter), and it is forced high at the channel (n) match (when the FTM counter = FTMxCnVH:FTMxCnVL) (Figure 13-40).



**Figure 13-40. EPWM Signal with ELSnB:ELSnA = X:1**

If (FTMxCnVH:FTMxCnVL = 0x0000), then the channel (n) output is a 0% duty cycle EPWM signal and CHnF bit is not set even when there is the channel (n) match. If (FTMxCnVH:FTMxCnVL > FTMxMODH:FTMxMODL), then the channel (n) output is a 100% duty cycle EPWM signal and CHnF bit is not set even when there is the channel (n) match. Therefore, FTMxMODH:FTMxMODL must be less than 0xFFFF in order to get a 100% duty cycle EPWM signal.

#### NOTE

- EPWM mode is only available with (FTMxCNTINH:FTMxCNTINL = 0x0000).
- EPWM mode with (FTMxCNTINH:FTMxCNTINL ≠ 0x0000) is not recommended and its results are not guaranteed.

### 13.4.7 Center-Aligned PWM (CPWM) Mode

The center-aligned mode is selected when (COMBINE = 0), and (CPWMS = 1).

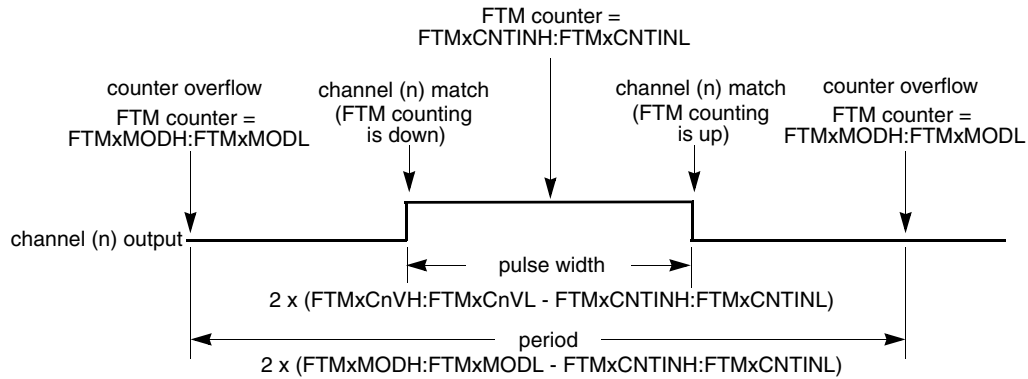
The CPWM pulse width (duty cycle) is determined by  $2 \times (\text{FTMxCnVH:FTMxCnVL} - \text{FTMxCNTINH:FTMxCNTINL})$  and the period is determined by  $2 \times (\text{FTMxMODH:FTMxMODL} - \text{FTMxCNTINH:FTMxCNTINL})$  (Figure 13-41). FTMxMODH:FTMxMODL must be kept in the range of 0x0001 to 0x7FFF because values outside this range can produce ambiguous results.

In the CPWM mode, the FTM counter counts up until it reaches FTMxMODH:FTMxMODL and then counts down until it reaches FTMxCNTINH:FTMxCNTINL.

The CHnF bit is set and channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = FTMxCnVH:FTMxCnVL) when the FTM counting is down (at the begin of the pulse width) and when the FTM counting is up (at the end of the pulse width).

This type of PWM signal is called center-aligned because the pulse width centers for all channels are aligned with the value of FTMxCNTINH:FTMxCNTINL.

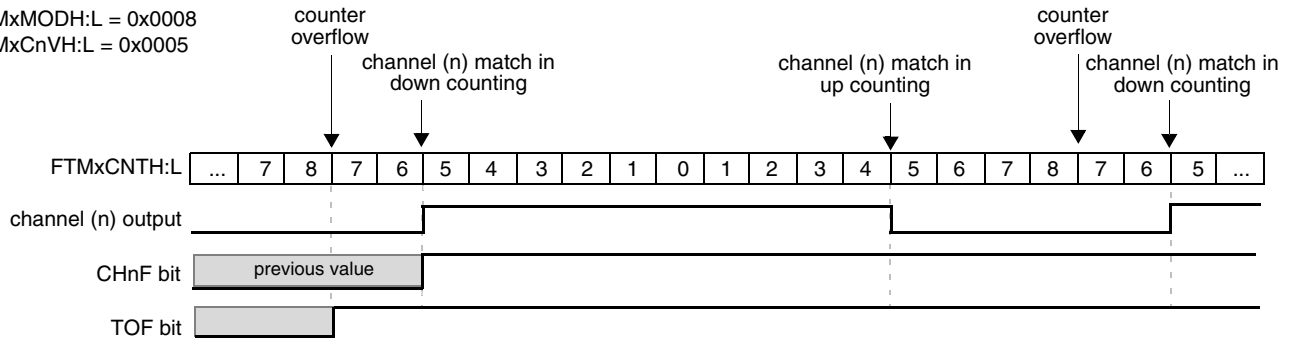
The other channel modes are not compatible with the up-down counter (CPWMS = 1). Therefore, all FTM channels must be used in CPWM mode when (CPWMS = 1).



**Figure 13-41. CPWM Period and Pulse Width with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = 0:0) when the counter reaches the value in the FTMxCnVH:FTMxCnVL registers, the CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1), however the channel (n) output is not controlled by FTM.

If (ELSnB:ELSnA = 1:0), then the channel (n) output is forced high at the channel (n) match (FTM counter = FTMxCnVH:FTMxCnVL) when counting down, and it is forced low at the channel (n) match when counting up (Figure 13-42).

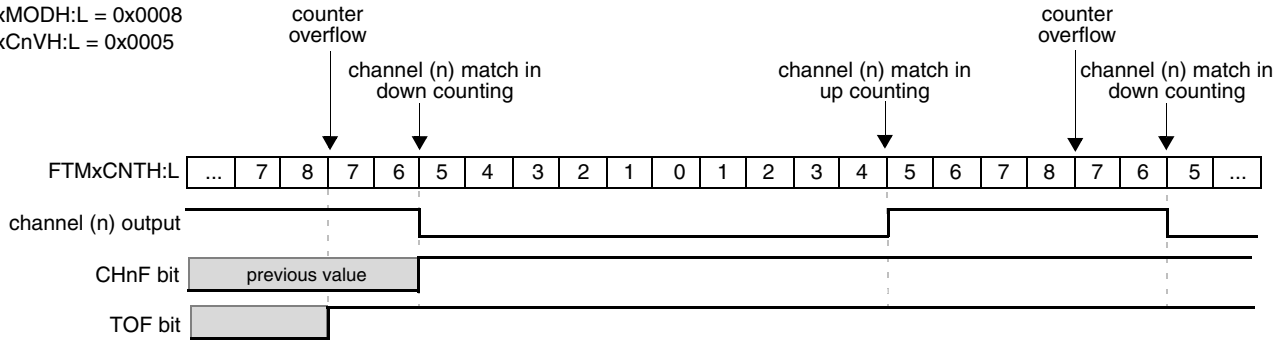


**Figure 13-42. CPWM Signal with ELSnB:ELSnA = 1:0**

If (ELSnB:ELSnA = X:1), then the channel (n) output is forced low at the channel (n) match (FTM counter = FTMxCnVH:FTMxCnVL) when counting down, and it is forced high at the channel (n) match when counting up (Figure 13-43).



FTMxMODH:L = 0x0008  
 FTMxCnVH:L = 0x0005



**Figure 13-43. CPWM Signal with ELSnB:ELSnA = X:1**

If (FTMxCnVH:FTMxCnVL = 0x0000) or (FTMxCnVH:FTMxCnVL is a negative value, that is, FTMxCnVH[7] = 1) then the channel (n) output is a 0% duty cycle CPWM signal and CHnF bit is not set even when there is the channel (n) match.

If (FTMxCnVH:FTMxCnVL is a positive value, that is, FTMxCnVH[7] = 0), (FTMxCnVH:FTMxCnVL ≥ FTMxMODH:FTMxMODL), and (FTMxMODH:FTMxMODL ≠ 0x0000), then the channel (n) output is a 100% duty cycle CPWM signal and CHnF bit is not set even when there is the channel (n) match. This implies that the usable range of periods set by FTMxMODH:FTMxMODL is 0x0001 through 0x7FFE (0x7FFF if you do not need to generate a 100% duty cycle CPWM signal). This is not a significant limitation because the resulting period is much longer than required for normal applications.

The CPWM mode must not be used when the FTM counter is a free running counter.

#### NOTE

- CPWM mode is only available with (FTMxCNTINH:FTMxCNTINL = 0x0000).
- CPWM mode with (FTMxCNTINH:FTMxCNTINL ≠ 0x0000) is not recommended and its results are not guaranteed.

### 13.4.8 Combine Mode

The combine mode is selected when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).

In combine mode, the channel (n) (an even channel) and channel (n+1) (the adjacent odd channel) are combined to generate a PWM signal in the channel (n) output.

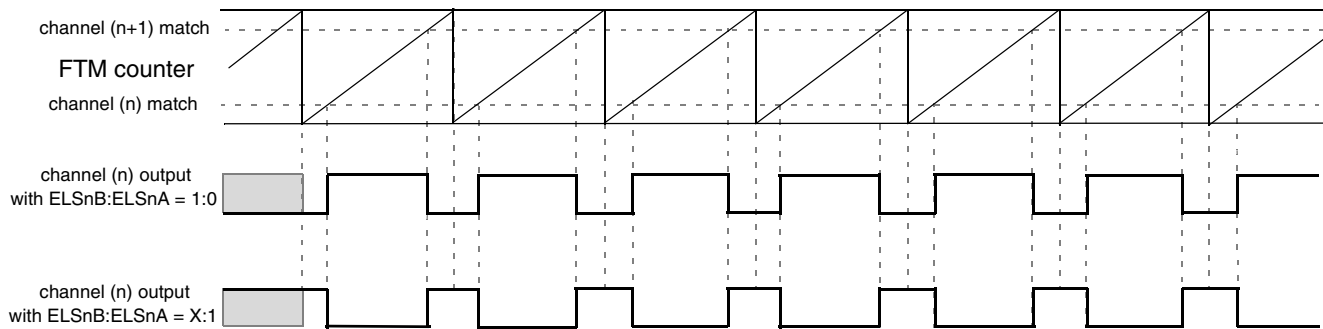
In the combine mode, the PWM period is determined by (FTMxMODH:FTMxMODL – FTMxCNTINH:FTMxCNTINL + 0x0001) and the PWM pulse width (duty cycle) is determined by ((FTMxC(n+1)VH:FTMxC(n+1)VL – FTMxC(n)VH:FTMxC(n)VL).

The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (FTM counter = FTMxC(n)VH:FTMxC(n)VL). The CH(n+1)F bit is set and the channel (n+1) interrupt is generated (if CH(n+1)IE = 1) at the channel (n+1) match (FTM counter = FTMxC(n+1)VH:FTMxC(n+1)VL).

If (ELSnB:ELSnA = 1:0), then the channel (n) output is forced low at the beginning of the period (FTM counter = FTMxCNTINH:FTMxCNTINL) and at the channel (n+1) match (FTM counter = FTMxC(n+1)VH:FTMxC(n+1)VL). It is forced high at the channel (n) match (FTM counter = FTMxC(n)VH:FTMxC(n)VL)(Figure 13-44).

If (ELSnB:ELSnA = X:1), then the channel (n) output is forced high at the beginning of the period (FTM counter = FTMxCNTINH:FTMxCNTINL) and at the channel (n+1) match (FTM counter = FTMxC(n+1)VH:FTMxC(n+1)VL). It is forced low at the channel (n) match (FTM counter = FTMxC(n)VH:FTMxC(n)VL)(Figure 13-44).

In combine mode, the ELS(n+1)B and ELS(n+1)A bits are not used in the generation of the channels (n) and (n+1) output.

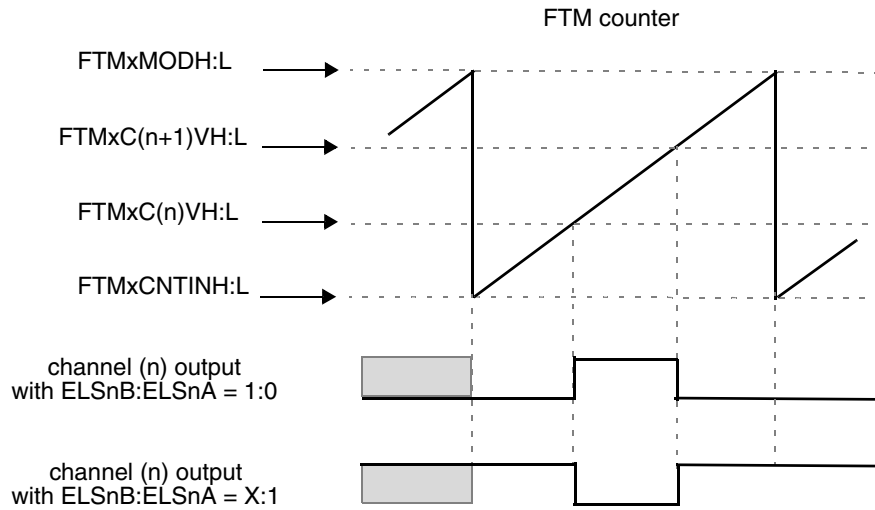


**Figure 13-44. Combine Mode**

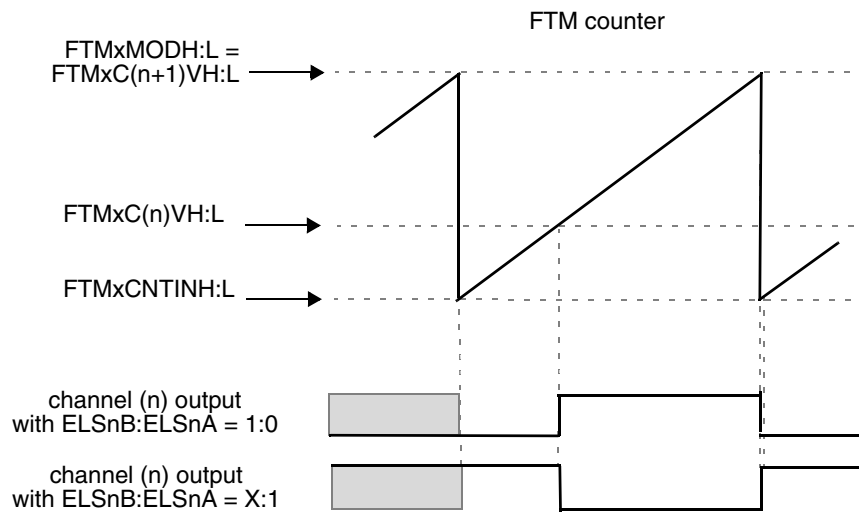
**NOTE**

- Combine mode is only available when (FTMEN = 1) and (CPWMS = 0).
- Combine mode with (FTMEN = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

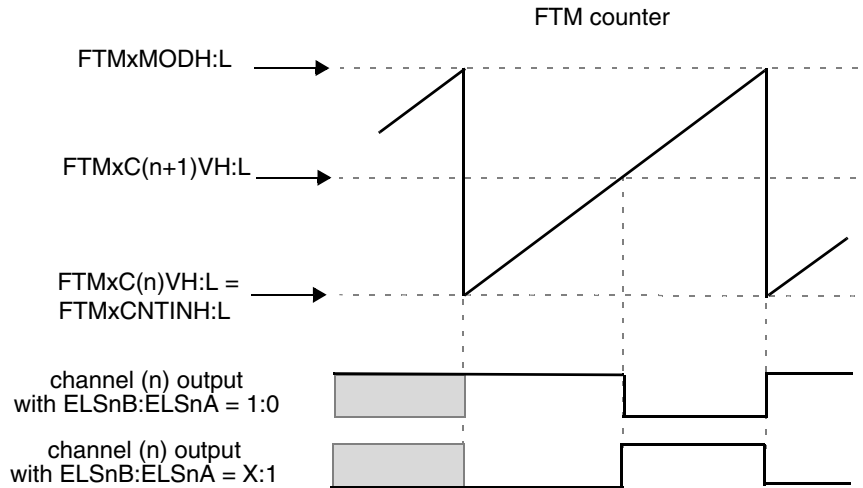
The following figures illustrate the generation of signals using combine mode.



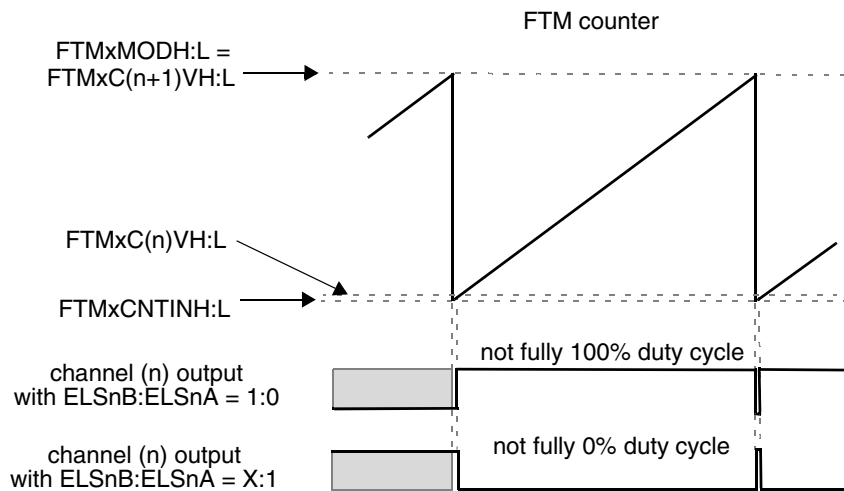
**Figure 13-45. Channel (n) Output If  $(FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)$  and  $(FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)$  and  $(FTMxC(n)VH:L < FTMxC(n+1)VH:L)$**



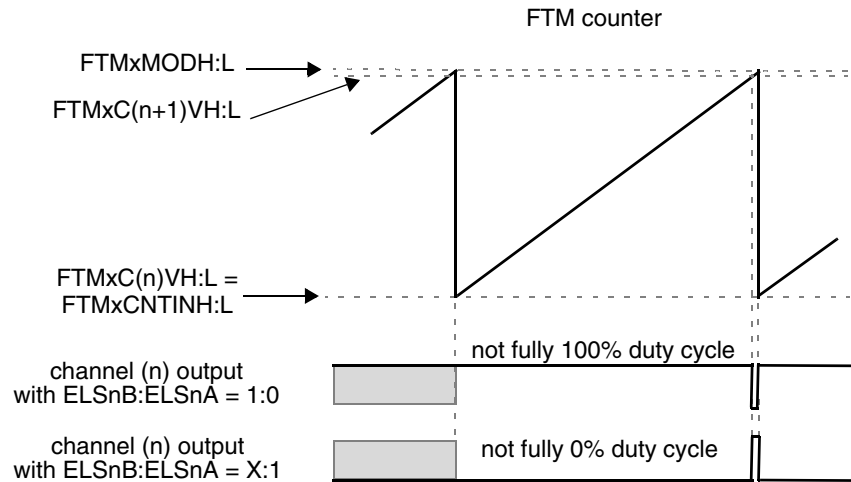
**Figure 13-46. Channel (n) Output If  $(FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)$  and  $(FTMxC(n+1)VH:L = FTMxMODH:L)$**



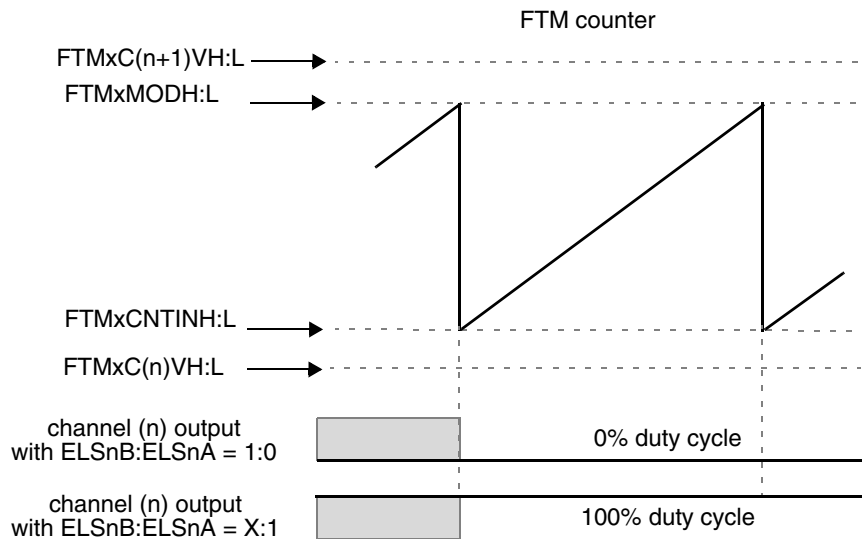
**Figure 13-47. Channel (n) Output If  $(FTMxC(n)VH:L = FTMxCNTINH:L)$  and  $(FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)$**



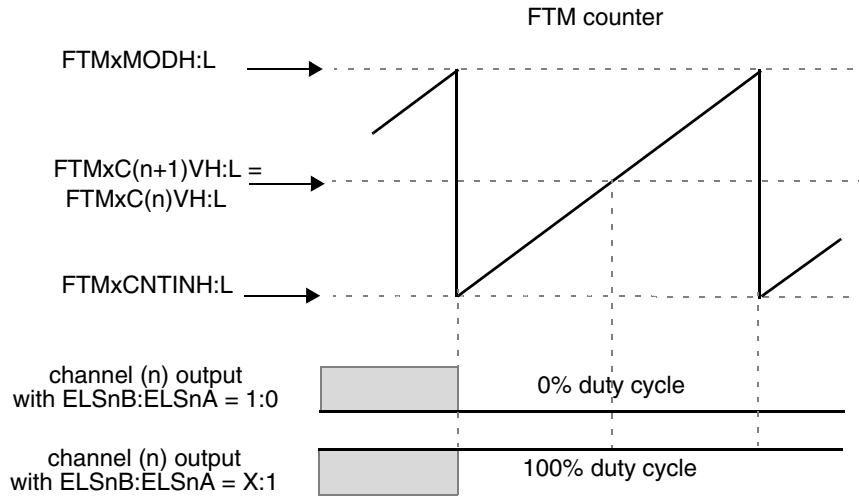
**Figure 13-48. Channel (n) Output If  $(FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)$  and  $(FTMxC(n)VH:L$  is Almost Equal to  $FTMxCNTINH:L$ ) and  $(FTMxC(n+1)VH:L = FTMxMODH:L)$**



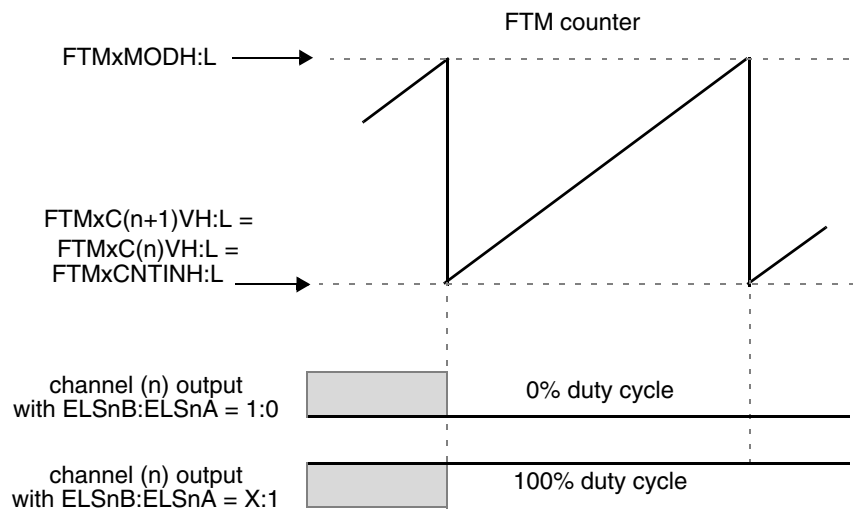
**Figure 13-49. Channel (n) Output If (FTMxC(n)VH:L = FTMxCNTINH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L) and (FTMxC(n+1)VH:L is Almost Equal to FTMxMODH:L)**



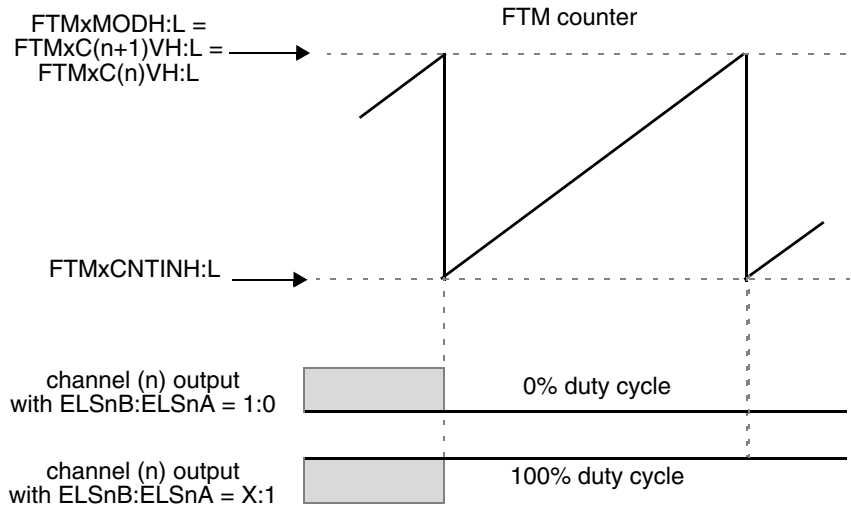
**Figure 13-50. Channel (n) Output If FTMxC(n)VH:L and FTMxC(n+1)VH:L Are Not Between FTMxCNTINH:L and FTMxMODH:L**



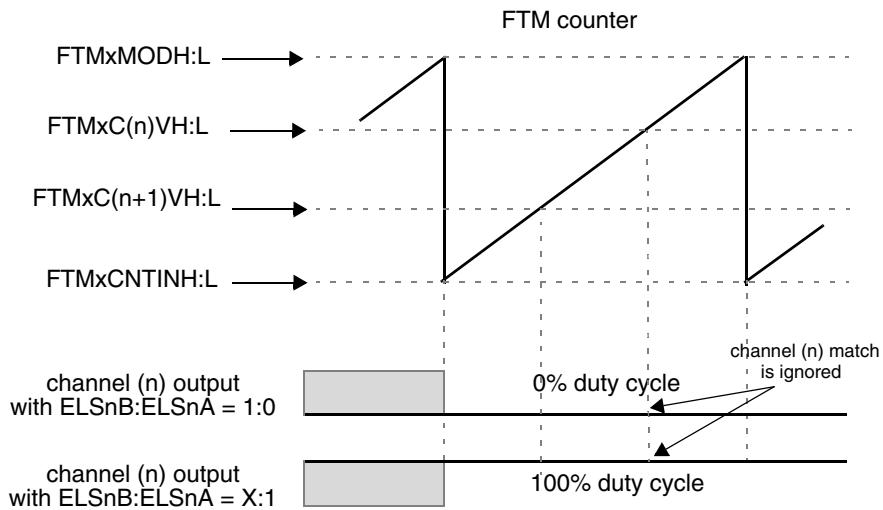
**Figure 13-51. Channel (n) Output If ( $FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L$ ) and ( $FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L$ ) and ( $FTMxCnVH:L = FTMxC(n+1)VH:L$ )**



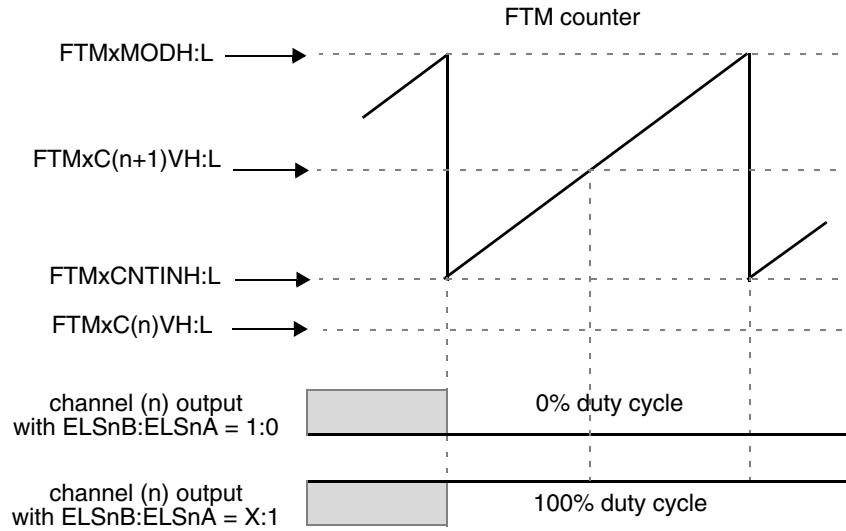
**Figure 13-52. Channel (n) Output If ( $FTMxC(n)VH:L = FTMxC(n+1)VH:L = FTMxCNTINH:L$ )**



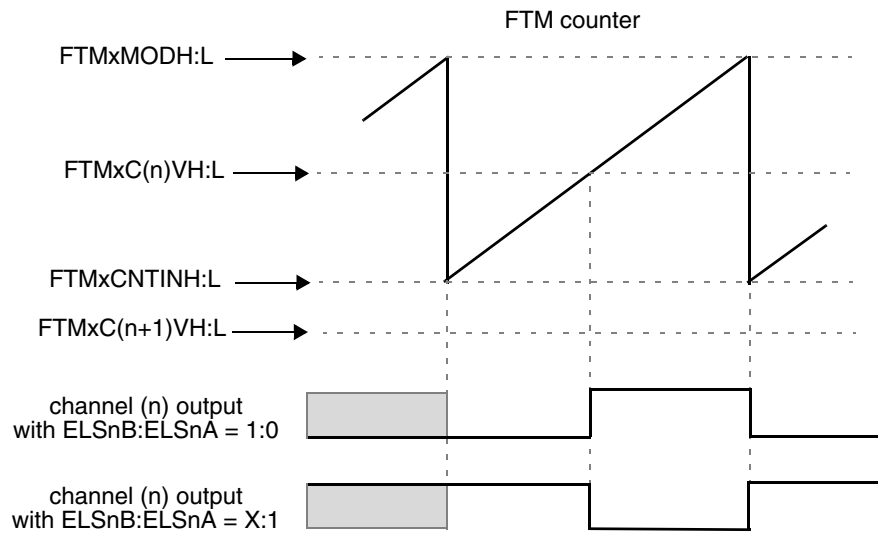
**Figure 13-53. Channel (n) Output If ( $FTMxC(n)VH:L = FTMxC(n+1)VH:L = FTMxMODH:L$ )**



**Figure 13-54. Channel (n) Output If ( $FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L$ ) and ( $FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L$ ) and ( $FTMxC(n)VH:L > FTMxC(n+1)VH:L$ )**

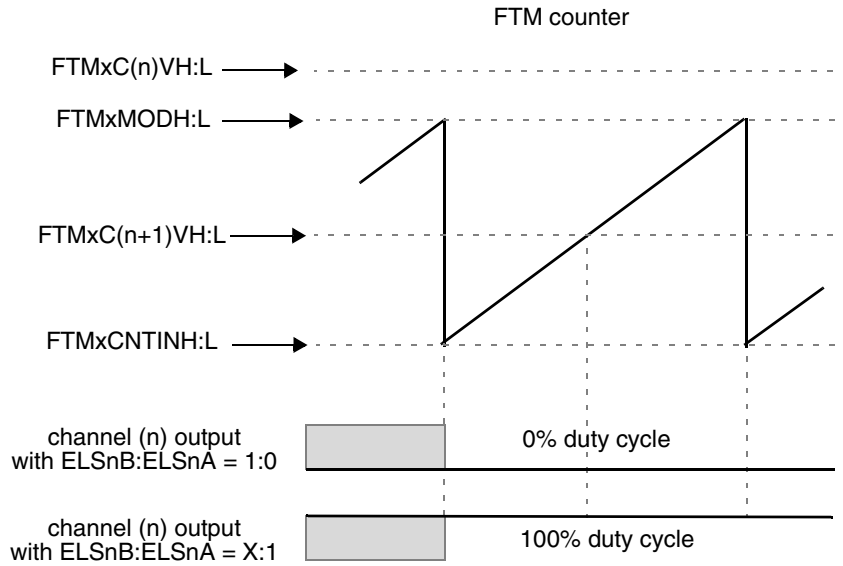


**Figure 13-55. Channel (n) Output If  $(FTMxC(n)VH:L < FTMxCNTINH:L)$  and  $(FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)$**

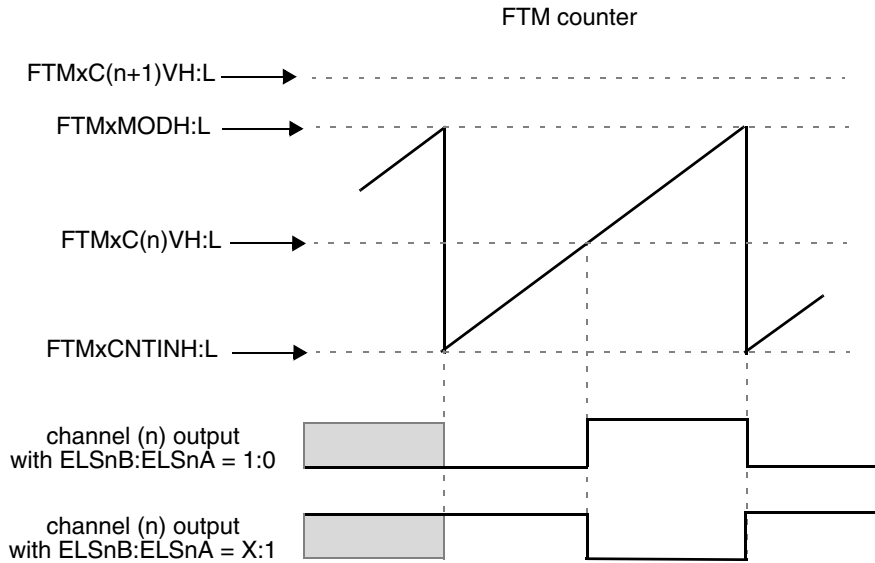


**Figure 13-56. Channel (n) Output If  $(FTMxC(n+1)VH:L < FTMxCNTINH:L)$  and  $(FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)$**

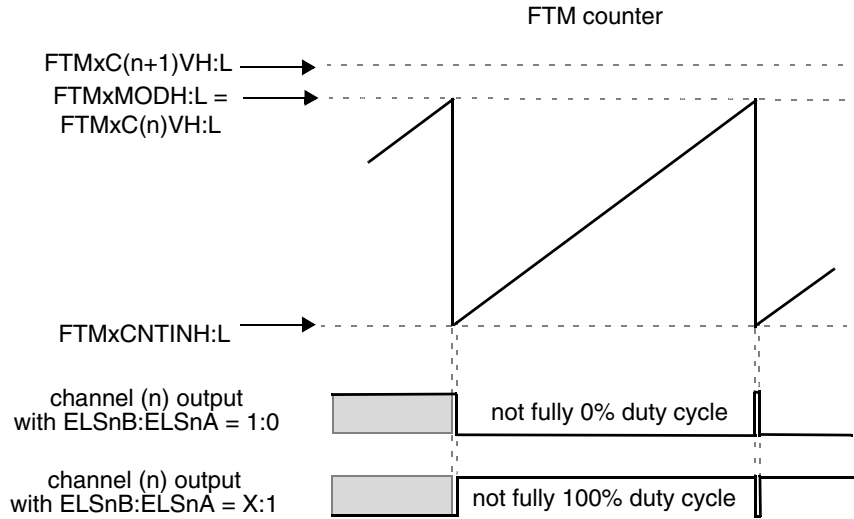




**Figure 13-57. Channel (n) Output If (FTMxC(n)VH:L > FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)**



**Figure 13-58. Channel (n) Output If (FTMxC(n+1)VH:L > FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)**



**Figure 13-59. Channel (n) Output If  $(FTMxC(n+1)VH:L > FTMxMODH:L)$  and  $(FTMxCNTINH:L < FTMxC(n)VH:L = FTMxMODH:L)$**

### 13.4.8.1 Asymmetrical PWM

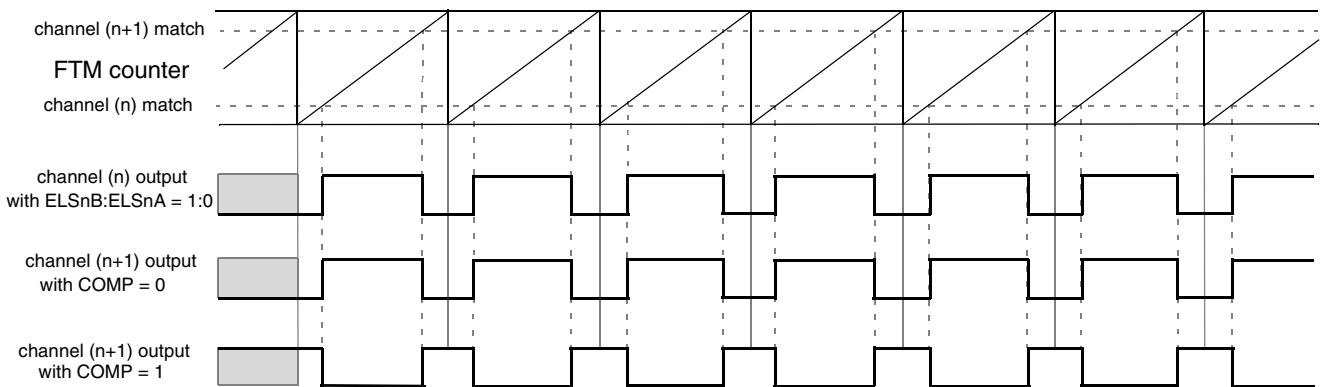
The combine mode can be used to generate an asymmetrical PWM signal when  $FTMxC(n)VH:L$  is different from  $FTMxC(n+1)VH:L$ .

### 13.4.9 Complementary Mode

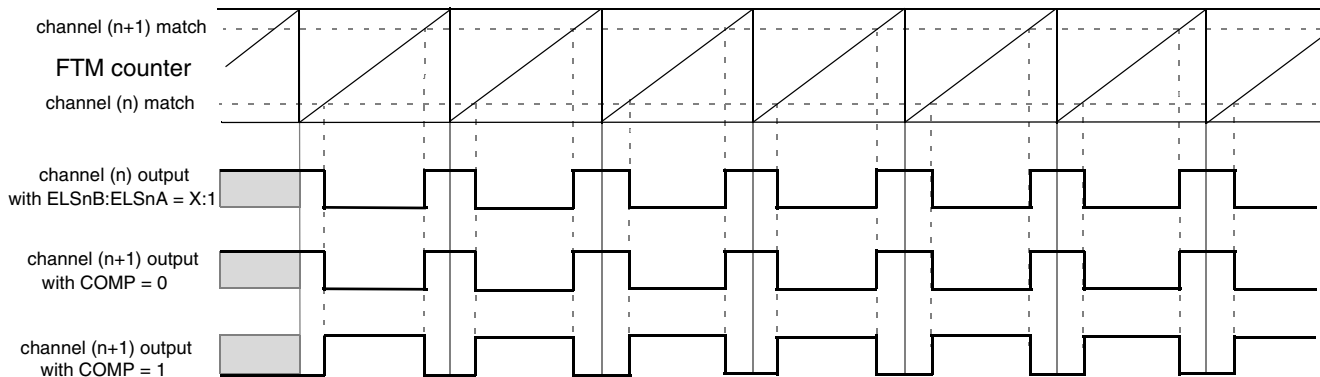
The complementary mode is selected when  $(FTMEN = 1)$ ,  $(COMBINE = 1)$ ,  $(CPWMS = 0)$ , and  $(COMP = 1)$ .

In complementary mode the channel (n+1) output is the inverse of the channel (n) output.

If  $(FTMEN = 1)$ ,  $(COMBINE = 1)$ ,  $(CPWMS = 0)$ , and  $(COMP = 0)$ , then the channel (n+1) output is the same as the channel (n) output.



**Figure 13-60. Channel (n+1) Output in Complementary Mode with  $(ELSnB:ELSnA = 1:0)$**



**Figure 13-61. Channel (n+1) Output in Complementary Mode with (ELSnB:ELSnA = X:1)**

**NOTE**

- Complementary mode is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- Complementary mode with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

## 13.4.10 Load of the Registers With Write Buffers

### 13.4.10.1 FTMxCNTINH:L Registers

FTMxCNTINH:L registers are always updated with their write buffer after both bytes have been written.

### 13.4.10.2 FTMxMODH:L Registers

If (CLKS[1:0] = 0:0), then FTMxMODH:L registers are updated when their second byte is written (independent of FTMEN bit).

If (CLKS[1:0] ≠ 0:0 and FTMEN = 0), then FTMxMODH:L registers are updated according to the CPWMS bit, that is:

- If the selected mode is not CPWM mode, then FTMxMODH:L registers are updated after both bytes have been written and the FTM counter changes from (FTMxMODH:L) to (FTMxCNTINH:L). If the FTM counter is a free-running counter, then this update is made when the FTM counter changes from 0xFFFF to 0x0000.
- If the selected mode is CPWM mode, then FTMxMODH:L registers are updated after both bytes have been written and the FTM counter changes from FTMxMODH:L to (FTMxMODH:L – 0x0001).

If (CLKS[1:0] ≠ 0:0 and FTMEN = 1), then FTMxMODH:L registers are updated by PWM synchronization (Section 13.4.11, “PWM Synchronization”).

### 13.4.10.3 FTMxCnVH:L Registers

If (CLKS[1:0] = 0:0), then FTMxCnVH:L registers are updated when their second byte is written (independent of FTMEN bit).

If (CLKS[1:0] ≠ 0:0 and FTMEN = 0), then FTMxCnVH:L registers are updated according to the selected mode, that is:

- If the selected mode is output compare mode, then FTMxCnVH:L registers are updated after their second byte is written and on the next change of the FTM counter (end of the prescaler counting).
- If the selected mode is EPWM mode, then FTMxCnVH:L registers are updated after both bytes have been written and the FTM counter changes from FTMxMODH:L to FTMxCNTINH:L. If the FTM counter is a free running counter, then this update is made when the FTM counter changes from 0xFFFF to 0x0000.
- If the selected mode is CPWM mode, then FTMxCnVH:L registers are updated after both bytes have been written and the FTM counter changes from FTMxMODH:L to (FTMxMODH:L – 0x0001).

If (CLKS[1:0] ≠ 0:0 and FTMEN = 1), then FTMxCnVH:L registers are updated according to the selected mode, that is:

- If the selected mode is output compare mode, then FTMxCnVH:L registers are updated according to the SYNCEN bit. If (SYNCEN = 0), then FTMxCnVH:L registers are updated after their second byte is written and on the next change of the FTM counter (end of the prescaler counting). If (SYNCEN = 1), then FTMxCnVH:L registers are updated by PWM synchronization ([Section 13.4.11, “PWM Synchronization”](#)).
- If the selected mode is not output compare mode and (SYNCEN = 1), then FTMxCnVH:L registers are updated by PWM synchronization ([Section 13.4.11, “PWM Synchronization”](#)).

### 13.4.11 PWM Synchronization

PWM synchronization provides an opportunity to update registers with the contents of their write buffers. It can also be used to synchronize two or more FlexTimer modules on the same MCU.

PWM synchronization updates the FTMxMODH:L and FTMxCnVH:L registers with their write buffers. It is also possible to force the FTM counter to its initial value and update the CHnOM bits in FTMxOUTMASK using PWM synchronization.

#### NOTE

- PWM synchronization is only available when (COMBINE = 1) and (CPWMS = 0).
- PWM synchronization with (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

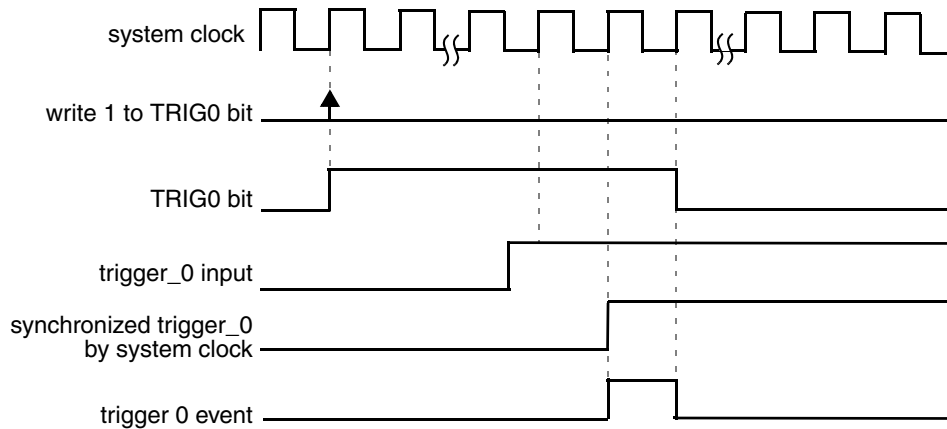
#### 13.4.11.1 Hardware Trigger

Each hardware trigger (input signals: trigger\_0, trigger\_1, and trigger\_2) is synchronized by the system clock.

A rising edge on the selected hardware trigger input (trigger n event) initiates PWM synchronization. A hardware trigger is selected when its enable bit is set ( $TRIGN = 1$  where  $n = 0, 1,$  or  $2$ ). The  $TRIGN$  bit is cleared when 0 is written to it or when the trigger n event is detected.

If two or more hardware triggers are enabled ( $TRIG0$  and  $TRIG1 = 1$ ) and only the trigger 1 event occurs, then only the  $TRIG1$  bit is cleared.

If a trigger n event occurs together with a write to set the  $TRIGN$  bit, then the synchronization is made, but the  $TRIGN$  bit remains set because of the last write.



Note

All hardware trigger (input signals: trigger\_0, trigger\_1, and trigger\_2) have this same behavior

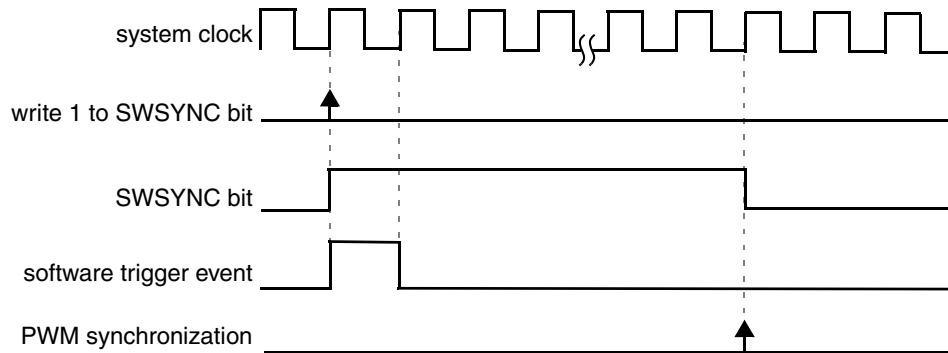
**Figure 13-62. Hardware Trigger Event**

### 13.4.11.2 Software Trigger

A software trigger event occurs when 1 is written to the  $SWSYNC$  bit. The  $SWSYNC$  bit is cleared when 0 is written to it or when the PWM synchronization (initiated by the software event) is completed.

If the software trigger event occurs together with the event that clears the  $SWSYNC$  bit, then the synchronization is made using this trigger event and the  $SWSYNC$  bit remains set because of the last write.

For example, if  $PWMSYNC = 0$  and  $REINIT = 0$  and there is a software trigger event, then the load of  $FTMxMODH:L$  and  $FTMxCnVH:L$  registers is only made at the boundary cycle ( $CNTMIN$  and  $CNTMAX$ ). In this case, the  $SWSYNC$  bit is cleared only at the boundary cycle, so you do not know when this bit is cleared. Therefore, it is possible a new write to set  $SWSYNC$  happens when FTM is clearing the  $SWSYNC$  because it is the selected boundary cycle of PWM synchronization that was started previously by the software trigger event.



**Figure 13-63. Software Trigger Event**

### 13.4.11.3 Boundary Cycle

The CNTMAX and CNTMIN bits select the boundary cycle when the FTMxMODH:L and FTMxCnVH:L registers are updated with the value of their write buffer by PWM synchronization, except if (PWMSYNC = 0 and REINIT = 1).

If CNTMIN = 1, then the boundary cycle is the FTMxCNTINH:L value. FTMxMODH:L and FTMxCnVH:L registers are updated when the FTM counter reaches the FTMxCNTINH:L value. If CPWMS = 0, then FTMxCNTINH:L is reached when the FTM counter changes from FTMxMODH:L to FTMxCNTINH:L. If CPWMS = 1, then FTMxCNTINH:L is reached when the FTM counter changes from (FTMxCNTINH:L + 0x0001) to FTMxCNTINH:L.

If CNTMAX = 1, then the boundary cycle is the FTMxMODH:L value. FTMxMODH:L and FTMxCnVH:L registers are updated when the FTM counter reaches the FTMxMODH:L value. FTMxMODH:L is reached when the FTM counter changes from (FTMxMODH:L - 0x0001) to FTMxMODH:L, regardless of the CPWMS configuration.

If no boundary cycle was selected (that is, CNTMAX = 0 and CNTMIN = 0), then the update of the FTMxMODH:L and FTMxCnVH:L registers is not made, except if (PWMSYNC = 0 and REINIT = 1).

If both boundary cycles were selected (that is, CNTMAX = 1 and CNTMIN = 1), then the update of the FTMxMODH:L and FTMxCnVH:L registers is made in the first boundary cycle that occurs with valid conditions for FTMxMODH:L or FTMxCnVH:L synchronization, except if (PWMSYNC = 0 and REINIT = 1).

The CNTMAX and CNTMIN bits are cleared only by software.

#### NOTE

- PWM synchronization boundary cycle is only available when (CNTMIN = 1).
- PWM synchronization with (CNTMAX = 1) is not recommended and its results are not guaranteed.

### 13.4.11.4 FTMxMODH:FTMxMODL Synchronization

The FTMxMODH:L synchronization occurs when the FTMxMODH:L registers are updated with the value of their write buffer.

The synchronization requires both bytes of FTMxMODH:L to have been written in one of the following situations.

- If  $PWMSYNC = 0$  and  $REINIT = 0$ , then the synchronization is made on the next selected boundary cycle after an enabled trigger event takes place. If the trigger event was a software trigger, then the SWSYNC bit is cleared on the next selected boundary cycle (Figure 13-64). If the trigger event was a hardware trigger, then the trigger enable bit (TRIGN) is cleared when the trigger event is detected (Figure 13-65).

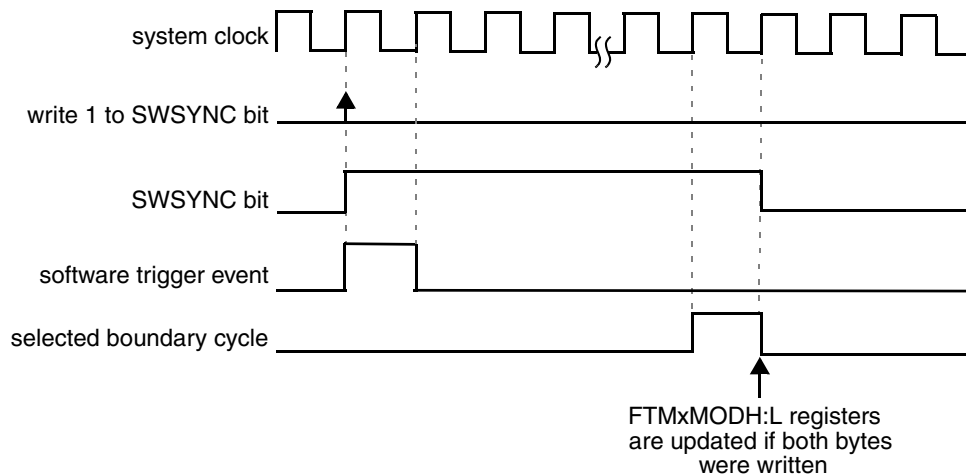
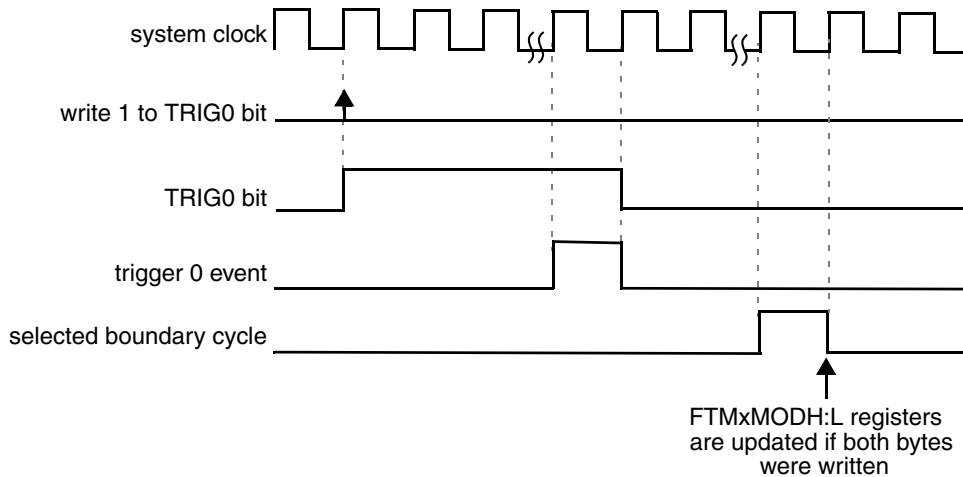
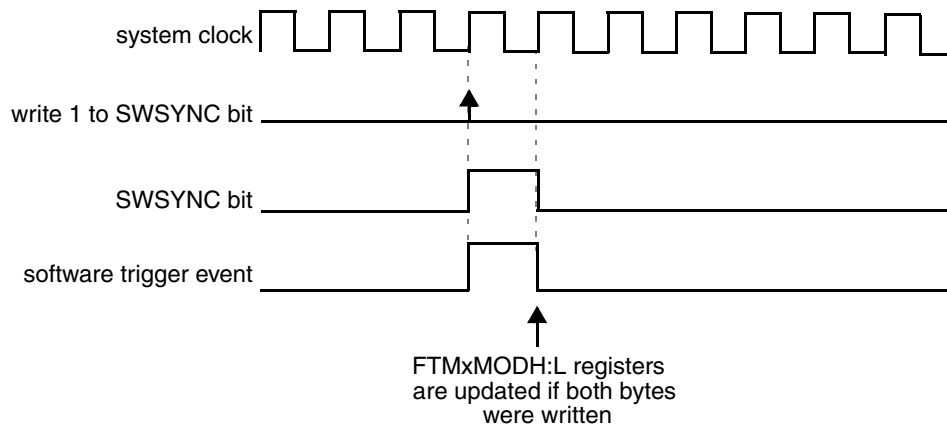


Figure 13-64. FTMxMODH:L Synchronization when ( $PWMSYNC = 0$ ), ( $REINIT = 0$ ), and (Software Trigger Was Used)



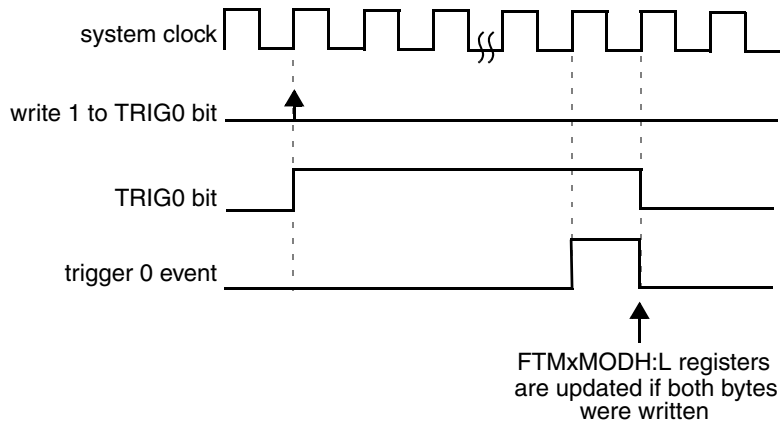
**Figure 13-65. FTMxMODH:L Synchronization when (PWMSYNC = 0), (REINIT = 0), and (a Hardware Trigger Was Used)**

- If PWMSYNC = 0 and REINIT = 1, then the synchronization is made on the next enabled trigger event. If the trigger event was a software trigger, then the SWSYNC bit is cleared (Figure 13-66). If the trigger event was a hardware trigger, then the TRIGn bit is cleared (Figure 13-67).



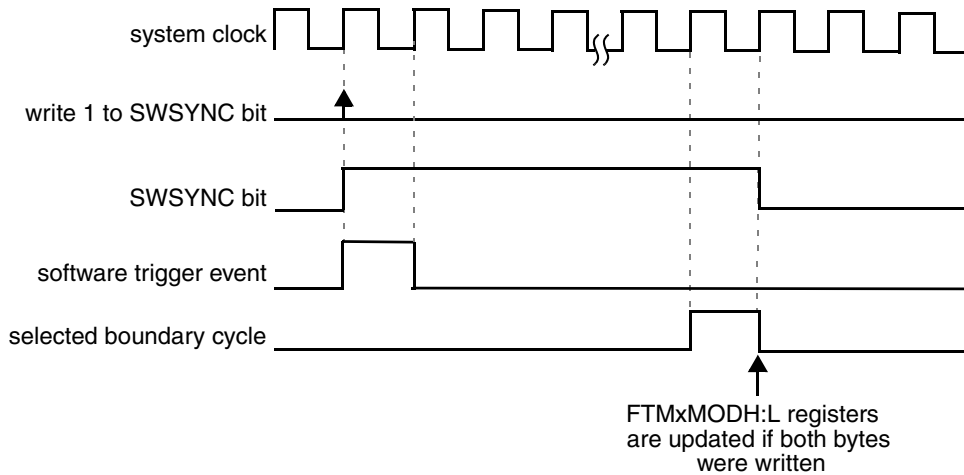
**Figure 13-66. FTMxMODH:L Synchronization when (PWMSYNC = 0), (REINIT = 1), and (Software Trigger Was Used)**





**Figure 13-67. FTMxMODH:L Synchronization when (PWMSYNC = 0), (REINIT = 1), and (a Hardware Trigger Was Used)**

- If PWMSYNC = 1, then the synchronization is made on the next selected boundary cycle after the enabled software trigger event takes place. The SWSYNC bit is cleared on the next selected boundary cycle (Figure 13-68).



**Figure 13-68. FTMxMODH:L Synchronization when (PWMSYNC = 1)**

### 13.4.11.5 FTMxCnVH:FTMxCnVL Synchronization

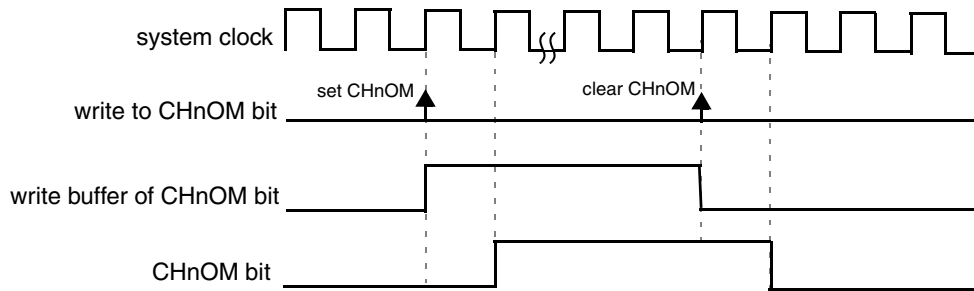
The FTMxCnVH:L synchronization occurs when the FTMxCnVH:L registers are updated with the value of their write buffer.

The synchronization requires both bytes of FTMxCnVH:L to have been written, SYNCEN = 1 and either a hardware or software trigger event as per FTMxMODH:L synchronization (Section 13.4.11.4, “FTMxMODH:FTMxMODL Synchronization”).

### 13.4.11.6 CHnOM Synchronization

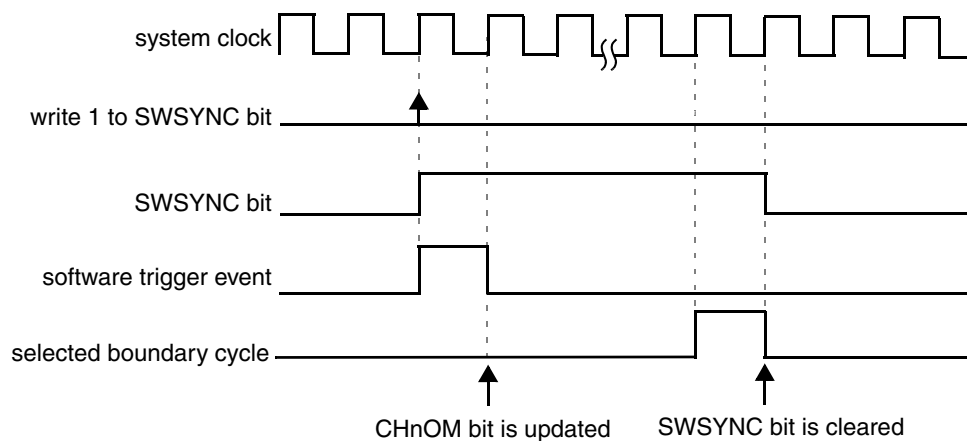
Any write to a CHnOM bit updates the FTMxOUTMASK write buffer. The CHnOM bit is updated with the value of its corresponding bit in the FTMxOUTMASK write buffer according to SYNCHOM and PWMSYNC bits.

If SYNCHOM = 0, then the CHnOM bit is updated with the value of its write buffer equivalent in all rising edges of the system clock.

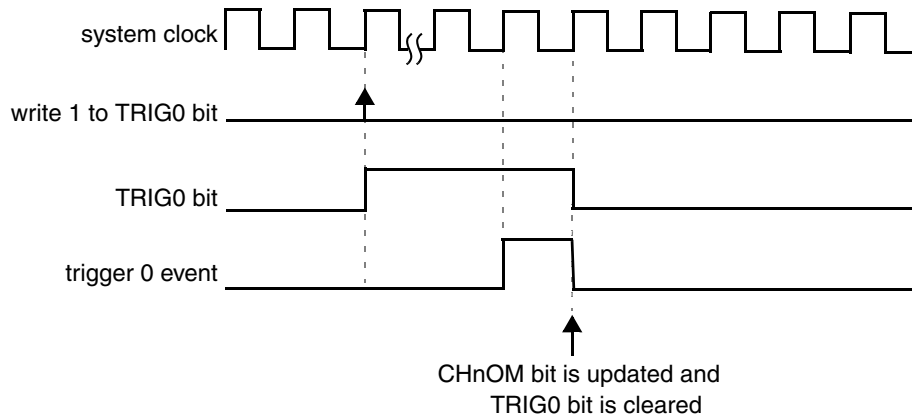


**Figure 13-69. CHnOM Synchronization when (SYNCHOM = 0)**

If SYNCHOM = 1 and PWMSYNC = 0, then this synchronization is made on the next enabled trigger event. If the trigger event was a software trigger, then the SWSYNC bit is cleared on the next selected boundary cycle (Figure 13-70). If the trigger event was a hardware trigger, then the trigger enable bit (TRIGn) is cleared when the trigger n event is detected (Figure 13-71).

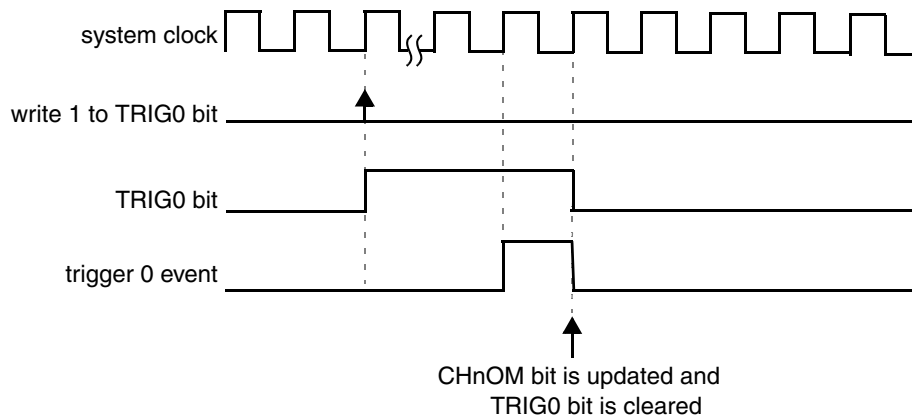


**Figure 13-70. CHnOM Synchronization when (SYNCHOM = 1), (PWMSYNC = 0) and (Software Trigger Was Used)**



**Figure 13-71. CHnOM Synchronization when (SYNCHOM = 1), (PWMSYNC = 0), and (a Hardware Trigger Was Used)**

If SYNCHOM = 1 and PWMSYNC = 1, then this synchronization is made on the next enabled hardware trigger event. The trigger enable bit (TRIGn) is cleared when the enabled hardware trigger n event is detected (Figure 13-72).

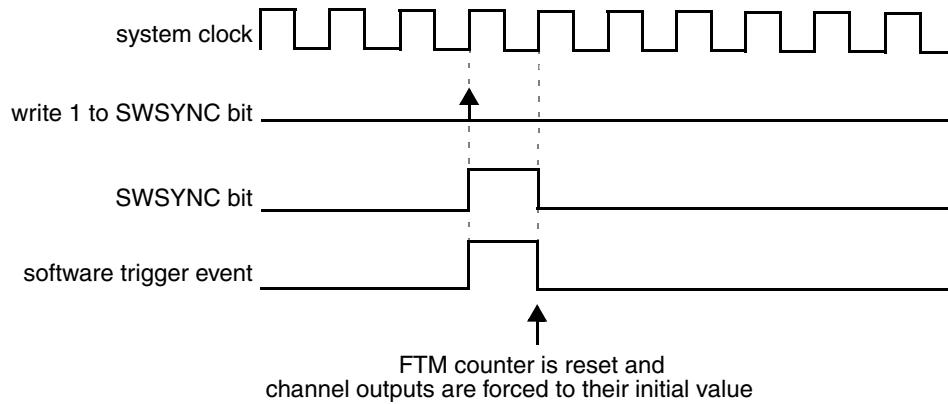


**Figure 13-72. CHnOM Synchronization when (SYNCHOM = 1), (PWMSYNC = 1), and (a Hardware Trigger Was Used)**

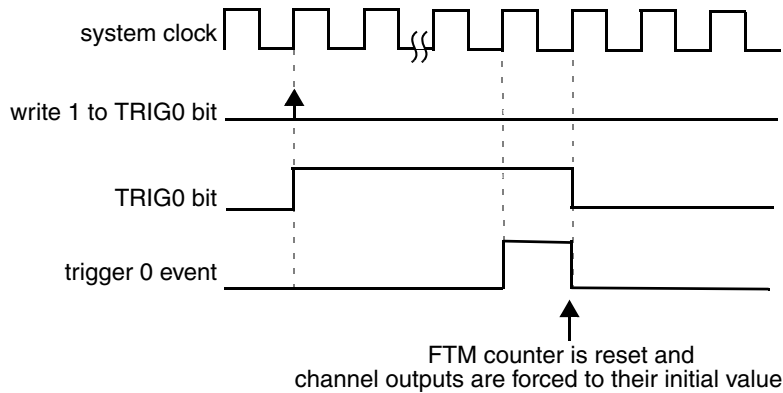
### 13.4.11.7 FTM Counter Synchronization

The FTM counter synchronization occurs when the FTM counter is updated with the value of the FTMxCNTINH:L registers and the channel outputs are forced to their initial value as defined by the channel configuration.

- If REINIT = 0, then this synchronization is made when the FTM counter changes from FTMxMODH:L to FTMxCNTINH:L.
- If REINIT = 1 and PWMSYNC = 0, then this synchronization is made on the next enabled trigger event. If the trigger event was a software trigger, then the SWSYNC bit is cleared (Figure 13-73). If the trigger event was a hardware trigger, then the TRIGn bit is cleared (Figure 13-74).

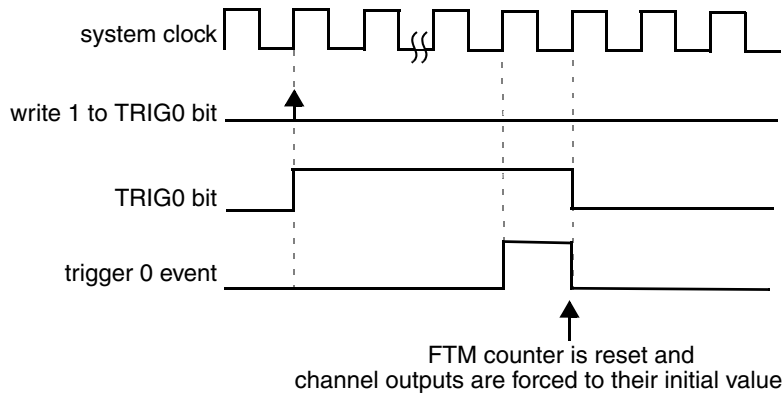


**Figure 13-73. FTM Counter Synchronization when (REINIT = 1), (PWMSYNC = 0), and (Software Trigger Was Used)**



**Figure 13-74. FTM Counter Synchronization when (REINIT = 1), (PWMSYNC = 0), and (a Hardware Trigger Was Used)**

- If REINIT = 1 and PWMSYNC = 1, then this synchronization is made on the next enabled hardware trigger event. The trigger enable bit (TRIGn) is cleared when the enabled hardware trigger n event is detected (Figure 13-75).



**Figure 13-75. FTM Counter Synchronization when (REINIT = 1), (PWMSYNC = 1), and (a Hardware Trigger Was Used)**

### 13.4.11.8 Summary of PWM Synchronization

Table 13-26 shows the summary of PWM synchronization.

**Table 13-26. Summary of PWM Synchronization**

Register or bit	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	Description
FTMxCNTINH:L	X	X	X	X	X	X	Changes take effect after the second byte is written. Effect is seen after the next TOF or PWM synchronization.
FTMxMODH:L	0	0	X	1	0	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled (hardware or software) trigger has occurred.
	0	0	X	0	1	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled (hardware or software) trigger has occurred.
	0	1	X	X	X	X	FTMxMODH:L are updated with their write buffer contents when the enabled (hardware or software) trigger occurs.
	1	X	X	1	0	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled software trigger has occurred.
	1	X	X	0	1	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled software trigger has occurred.

**Table 13-26. Summary of PWM Synchronization (continued)**

Register or bit	PWMSYNC	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	Description
FTMxCnVH:L	0	0	X	1	0	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled (hardware or software) trigger has occurred.
	0	0	X	0	1	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled (hardware or software) trigger has occurred.
	0	1	X	X	X	1	FTMxCnVH:L are updated with their write buffer contents when the enabled (hardware or software) trigger occurs.
	1	X	X	1	0	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its maximum value after the enabled software trigger has occurred.
	1	X	X	0	1	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its minimum value after the enabled software trigger has occurred.
FTMxCNTH:L	0	1	X	X	X	X	FTMxCNTH:L are forced to the FTM counter initial value when the enabled (hardware or software) trigger occurs.
	1	1	X	X	X	X	FTMxCNTH:L are forced to the FTM counter initial value when the enabled hardware trigger occurs.
FTMxOUTMASK	X	X	0	X	X	X	Changes to FTMxOUTMASK take effect on the next rising edge of the system clock.
	0	X	1	X	X	X	FTMxOUTMASK is updated with its write buffer contents when the enabled (hardware or software) trigger occurs.
	1	X	1	X	X	X	FTMxOUTMASK is updated with its write buffer contents when the enabled hardware trigger occurs.
SWSYNC bit	0	0	X	1	0	X	SWSYNC bit is cleared when the counter reaches its maximum value after the enabled software trigger has occurred.
	0	0	X	0	1	X	SWSYNC bit is cleared when the counter reaches its minimum value after the enabled software trigger has occurred.
	0	1	X	X	X	X	SWSYNC bit is cleared when the enabled software trigger occurs.
	1	X	X	1	0	X	SWSYNC bit is cleared when the counter reaches its maximum value after the enabled software trigger has occurred.
	1	X	X	0	1	X	SWSYNC bit is cleared when the counter reaches its minimum value after the enabled software trigger has occurred.
TRIGn bit	X	X	X	X	X	X	TRIGn bit is cleared when the enabled hardware trigger has occurred.

### 13.4.12 Deadtime Insertion

The deadtime insertion is enabled when (DTEN = 1) and (DTVAL[5:0] is non-zero).

FTM<sub>x</sub>DEADTIME register defines the deadtime delay that can be used for all FTM channels. The DT<sub>PS</sub>[1:0] bits define the prescaler for the system clock and the DT<sub>VAL</sub>[5:0] bits define the deadtime modulo (number of the deadtime prescaler clocks).

The deadtime delay insertion ensures that no two complementary signals (channel (n) and (n+1)) drive the active state at the same time.

For POL(n) = 0, POL(n+1) = 0, and deadtime enabled, a rising edge on the output of channel (n) remains low for the duration of the deadtime delay, after which the rising edge appears on the output. Similarly, when a falling edge is due on the output of channel (n), the channel (n+1) output remains low for the duration of the deadtime delay, after which the channel (n+1) output will have a rising edge.

For POL(n) = 1, POL(n+1) = 1, and deadtime enabled, a falling edge on the output of channel (n) remains high for the duration of the deadtime delay, after which the falling edge appears on the output. Similarly, when a rising edge is due on the output of channel (n), the channel (n+1) output remains high for the duration of the deadtime delay, after which the channel (n+1) output will have a falling edge.

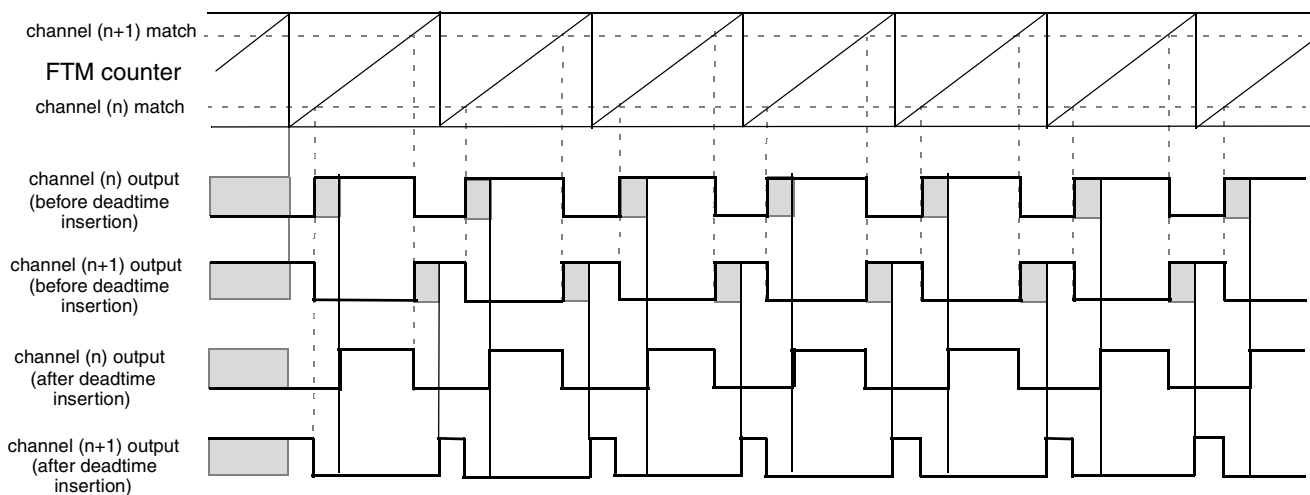


Figure 13-76. Deadtime Insertion with EL<sub>n</sub>B:EL<sub>n</sub>A = 1:0

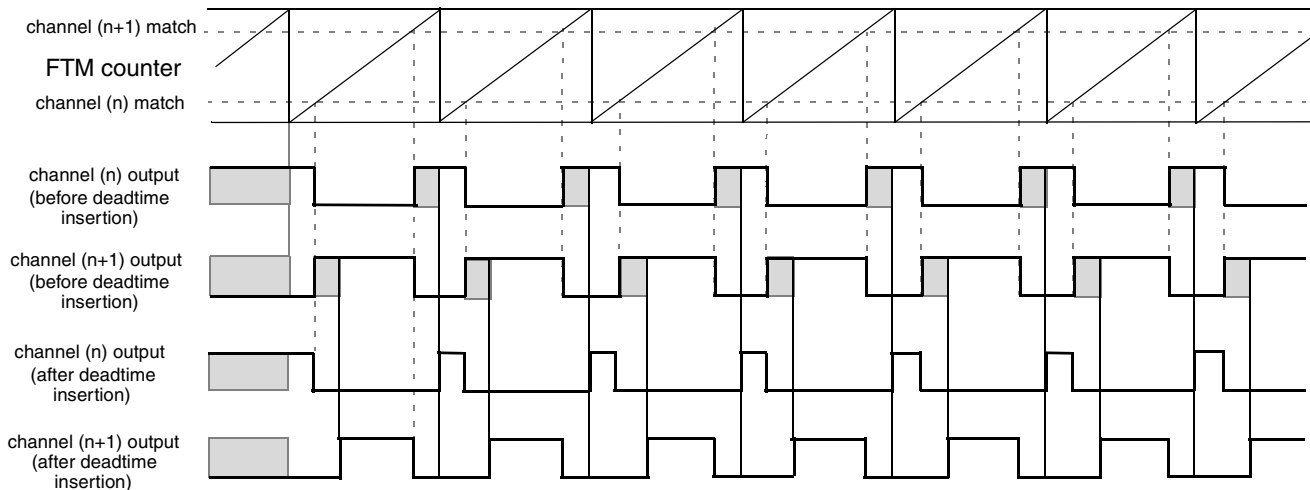


Figure 13-77. Deadtime Insertion with  $ELSnB:ELSnA = X:1$

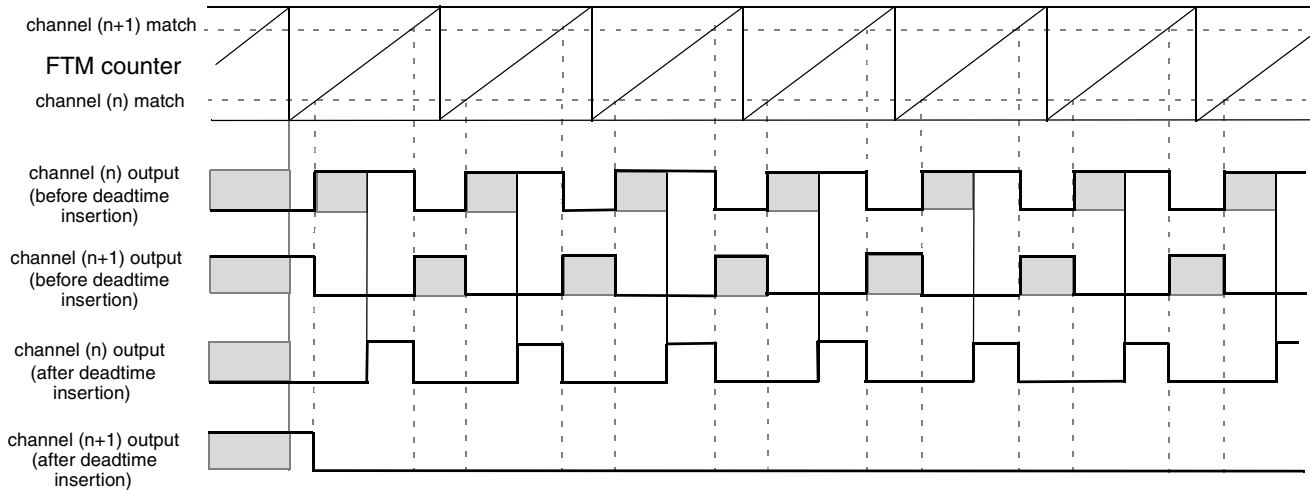
### 13.4.12.1 Deadtimer Insertion Corner Cases

In the case that (PS[2:0] bits are cleared) and (DTPS[1:0] = 0:0 or DTPS[1:0] = 0:1):

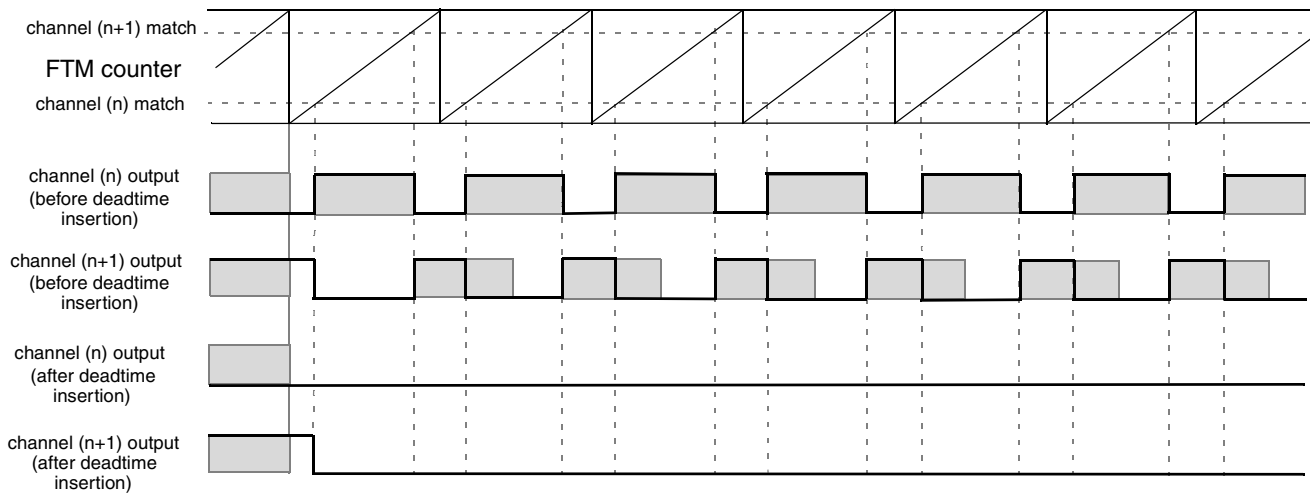
- If the deadtime delay is greater than or equal to the channel (n) duty cycle ( $(FTMxC(n+1)VH:L - FTMxC(n)VH:L) \times \text{system clock}$ ), then the channel (n) output is always 0.
- If the deadtime delay is greater than or equal to the channel (n+1) duty cycle ( $(FTMxMODH:L - FTMxCNTINH:L + 1 - (FTMxC(n+1)VH:L - FTMxC(n)VH:L)) \times \text{system clock}$ ), then the channel (n+1) output is always 0.

Although, in the most of cases the deadtime delay is not comparable to channels (n) and (n+1) duty cycle, [Figure 13-78](#) shows an example when the deadtime delay is comparable to channel (n+1) duty cycle and [Figure 13-79](#) shows an example when the deadtime delay is comparable to channels (n) and (n+1) duty cycles.





**Figure 13-78. Example of the Deadtime Insertion (ELSnB:ELSnA = 1:0) when the Deadtime Delay Is Comparable To Channel (n+1) Duty Cycle**



**Figure 13-79. Example of the Deadtime Insertion (ELSnB:ELSnA = 1:0) when the Deadtime Delay Is Comparable To Channels (n) and (n+1) Duty Cycle**

**NOTE**

- Deadtime insertion is only available when (FTMEN = 1), (COMBINE = 1), (CPWMS = 0), and (COMP = 1).
- Deadtime insertion with (FTMEN = 0), (COMBINE = 0), (CPWMS = 1), or (COMP = 0) is not recommended and its results are not guaranteed.

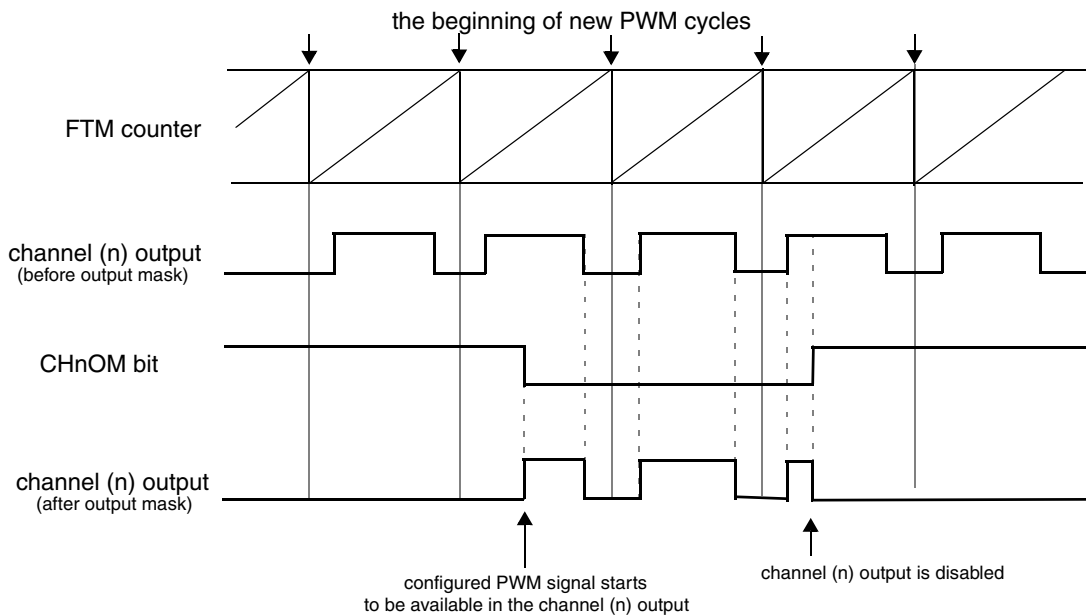
### 13.4.13 Output Mask

The output mask register FTMxOUTMASK can be used to force channel outputs to their inactive state through software (for example: to control a BLDC motor).

Any write to a CHnOM bits updates the FTMxOUTMASK write buffer. The CHnOM bit is updated with the value of its corresponding bit in the FTMxOUTMASK write buffer according to [Section 13.4.11.6, “CHnOM Synchronization.”](#)

If CHnOM = 1, then the channel (n) output is forced to its inactive state, defined by the POLn bit in register FTMxPOL. If CHnOM = 0, then the channel (n) output is unaffected by the output mask function.

When a CHnOM bit is cleared, the channel (n) output is enabled ([Figure 13-80](#)).



**Figure 13-80. Output Mask**

The [Table 13-27](#) shows the output mask result before the polarity control.

**Table 13-27. Output Mask Result for Channel (n) (Before the Polarity Control)**

CHnOM	Output Mask Input	Output Mask Result
0	inactive state	inactive state
	active state	active state
1	inactive state	inactive state
	active state	inactive state

**NOTE**

- Output mask is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).

- Output mask with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

### 13.4.14 Fault Control

The fault control is enabled if (FTMEN = 1) and (FAULTM[1:0] ≠ 0:0).

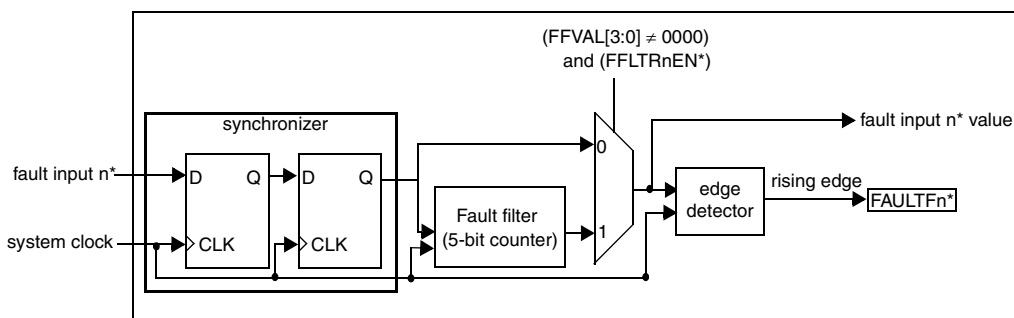
FTM can have up to four fault inputs. FAULTnEN bit (where n = 0, 1, 2, 3) enables the fault input n and FFLTRnEN bit enables the fault input n filter. FFVAL[3:0] bits select the value of the enabled filter in each enabled fault input.

First each fault input signal is synchronized by the system clock (synchronizer block in Figure 13-81). Following synchronization, the fault input n signal enters the filter block. When there is a state change in the fault input n signal, the 5-bit counter is reset and starts counting up. As long as the new state is stable on the fault input n, the counter continues to increment. If the 5-bit counter overflows (the counter exceeds the value of the FFVAL[3:0] bits), the new fault input n value is validated. It is then transmitted as a pulse edge to the edge detector.

If the opposite edge appears on the fault input n signal before validation (counter overflow), the counter is reset. At the next input transition, the counter starts counting again. Any pulse that is shorter than the minimum value selected by FFVAL[3:0] bits (× system clock) is regarded as a glitch and is not passed on to the edge detector.

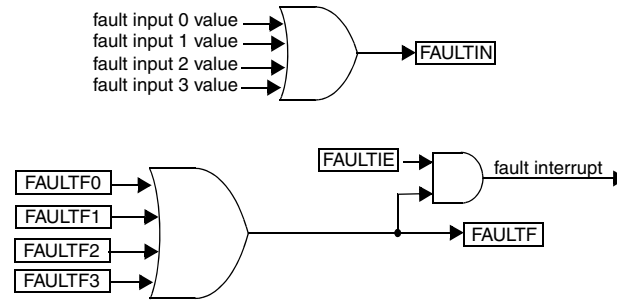
The fault input n filter is disabled when the FFVAL[3:0] bits are zero or when FAULTnEN = 0. In this case the fault input n signal is delayed 2 rising edges of the system clock and the FAULTFn bit is set on 3th rising edge of the system clock after a rising edge occurs on the fault input n.

If FFVAL[3:0] ≠ 0000 and FAULTnEN = 1, then the fault input n signal is delayed (3 + FFVAL[3:0]) rising edges of the system clock, that is, the FAULTFn bit is set (4 + FFVAL[3:0]) rising edges of the system clock after a rising edge occurs on the fault input n.



\* where n = 3, 2, 1, 0

Figure 13-81. Fault Input n Control Block Diagram



**Figure 13-82. FAULTF and FAULTIN Bits and Fault Interrupt**

If the fault control and fault input  $n$  are enabled and a rising edge at the fault input  $n$  signal is detected, then the  $FAULTFn$  bit is set (Figure 13-81). The  $FAULTF$  bit is the logic OR of  $FAULTFn[3:0]$  bits (Figure 13-82).

If the fault control is enabled ( $FAULTM[1:0] \neq 0:0$ ), a fault condition has occurred (rising edge at the logic OR of the enabled fault input) and ( $FAULTEN = 1$ ), then channel ( $n$ ) and ( $n+1$ ) outputs are forced to their safe value (that is, the channel ( $n$ ) output is forced to the value of  $POL(n)$  and the channel ( $n+1$ ) is forced to the value of  $POL(n+1)$ ).

The fault interrupt is generated when ( $FAULTF = 1$ ) and ( $FAULTIE = 1$ ) (Figure 13-82). This interrupt request remains set until:

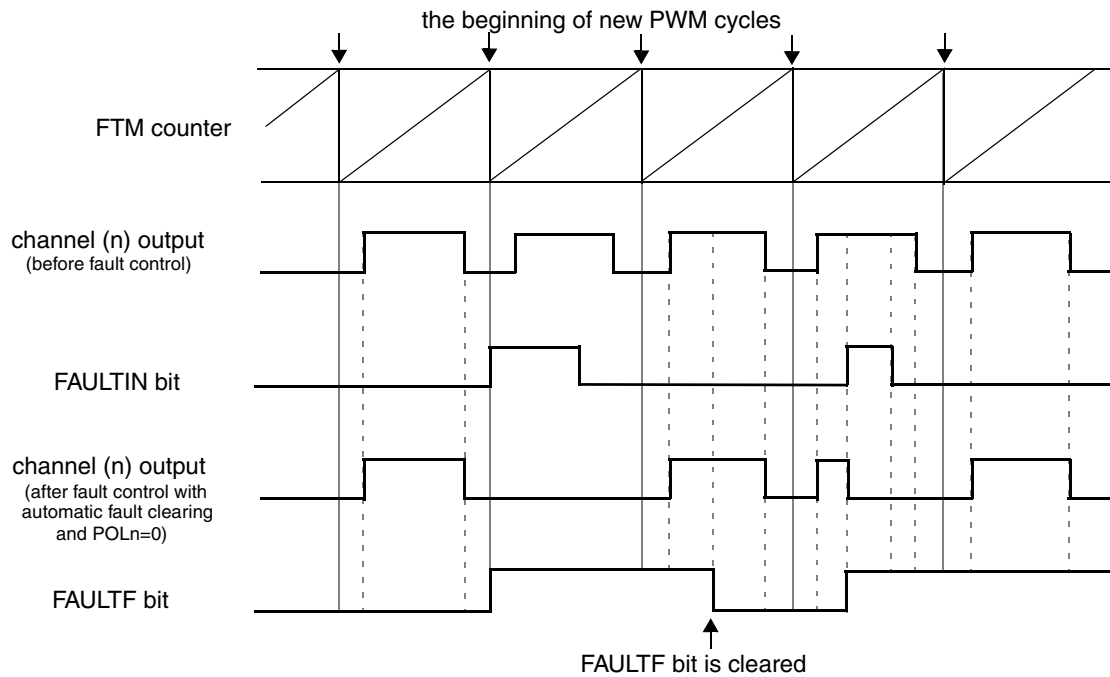
- Software clears the  $FAULTF$  bit (by reading  $FAULTF$  bit as 1 and writing 0 to it)
- Software clears the  $FAULTIE$  bit
- A reset occurs

#### NOTE

- Fault control is only available when ( $FTMEN = 1$ ), ( $COMBINE = 1$ ), and ( $CPWMS = 0$ ).
- Fault control with ( $FTMEN = 0$ ), ( $COMBINE = 0$ ), or ( $CPWMS = 1$ ) is not recommended and its results are not guaranteed.

#### 13.4.14.1 Automatic Fault Clearing

If the automatic fault clearing is selected ( $FAULTM[1:0] = 1:1$ ), then the disabled channel outputs are enabled when the fault input signal ( $FAULTIN$ ) returns to zero and a new PWM cycle begins (Figure 13-83).



**Figure 13-83. Fault Control with Automatic Fault Clearing**

### 13.4.14.2 Manual Fault Clearing

If the manual fault clearing is selected ( $FAULTM[1:0] = 0:1$  or  $1:0$ ), then disabled channel outputs are enabled when the FAULTF bit is cleared and a new PWM cycle begins (Figure 13-84).

It is possible to manually clear a fault, by clearing the FAULTF bit, and enable disabled channels regardless of the fault input signal (FAULTIN) (the filter output if the filter is enabled or the synchronizer output if the filter is disabled). However, it is recommended to verify the value of the fault input signal (value of the FAULTIN bit) before clearing the FAULTF bit to avoid unpredictable results.

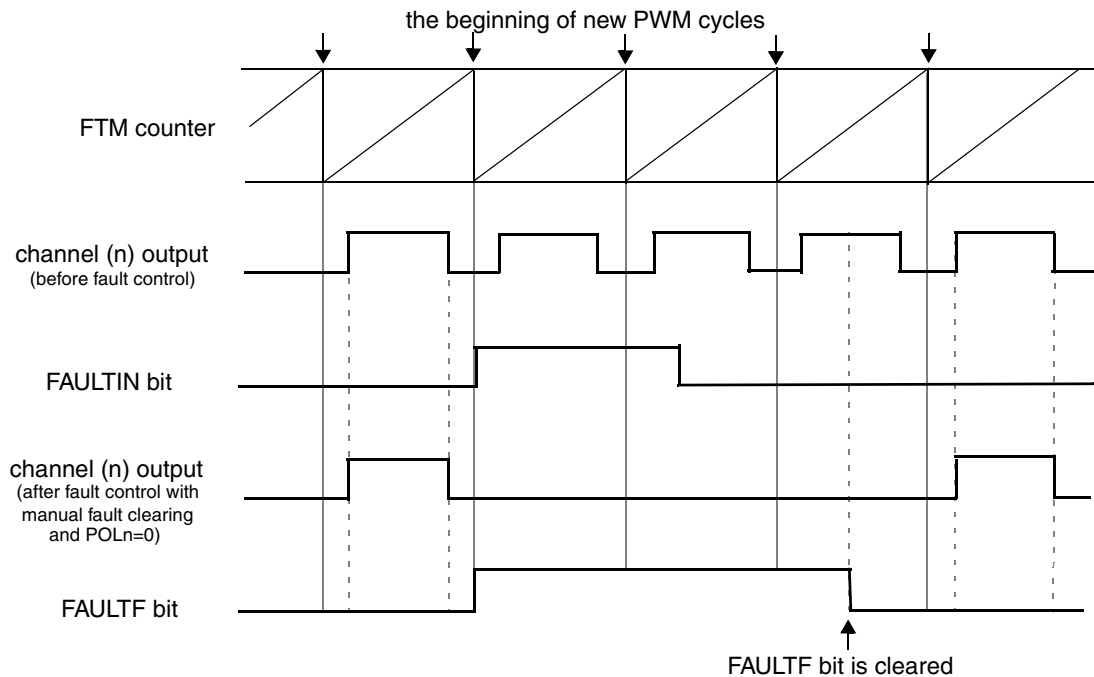


Figure 13-84. Fault Control with Manual Fault Clearing

### 13.4.15 Polarity Control

The polarity control mode is enabled if (FTMEN = 1). Each channel has its polarity control defined by its POLn bit.

- If (POLn = 0), the signal driving the polarity control is the same polarity as the channel (n) output.
- If (POLn = 1), the signal driving the polarity control is negated at the channel (n) output.

#### NOTE

- Polarity control is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- Polarity control with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

### 13.4.16 Initialization

The initialization of the output channels is enabled if (FTMEN = 1). When a logic 1 is written to the INIT bit, the value of the CHnOI bit is forced to the channel (n) output. Figure 13-85 shows the priority of the initialization feature over the other features that are combined to generate the channels (n) and (n+1) output.

It is recommended using the initialization only if the FTM counter is disabled (that is, CLKS[1:0] = 0:0), otherwise unpredictable behavior is likely.

## NOTE

- Initialization is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- Initialization with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

### 13.4.17 Features Priority

Figure 13-85 shows the priority of the features that can be combined to generate channel (n) and (n+1) outputs.

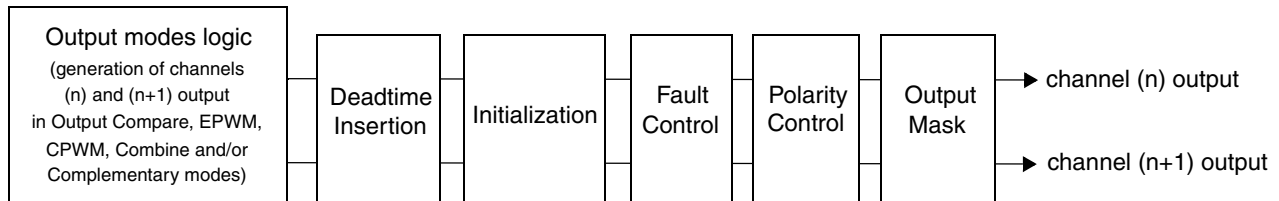


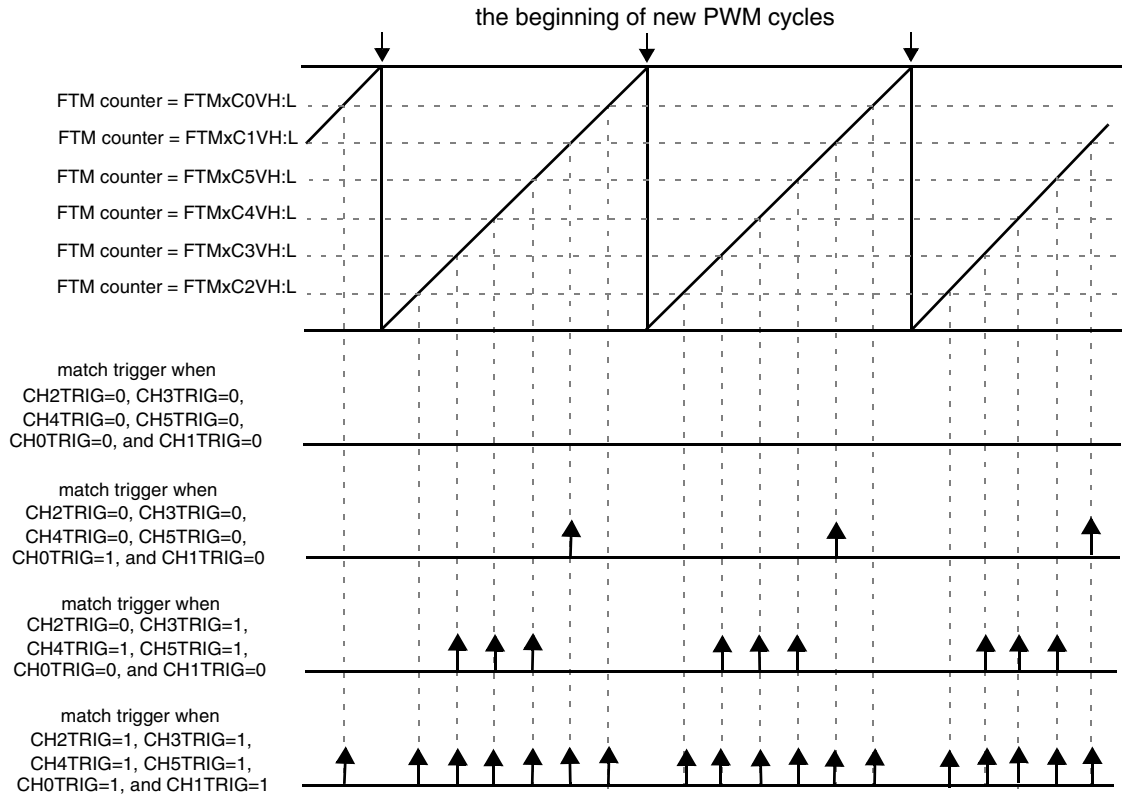
Figure 13-85. FTM Features Priority

### 13.4.18 Channel Trigger Output

The channel trigger output is generated if (FTMEN = 1) and (one or more channels were selected by the CH<sub>j</sub>TRIG bit, where j = 0, 1, 2, 3, 4, or 5). The CH<sub>j</sub>TRIG bit defines if the channel (j) match (that is, FTM counter = FTMxC(j)VH:L) generates the trigger.

The channel trigger output provides a trigger signal that is used for on-chip modules.

The FTM is able to generate multiple triggers in one PWM period. Since each trigger is generated for a specific channel several channels are required to implement this functionality. This behavior is described in Figure 13-86.



**Figure 13-86. Match Triggers**

**NOTE**

- Match trigger is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- Match trigger with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

### 13.4.19 Initialization Trigger

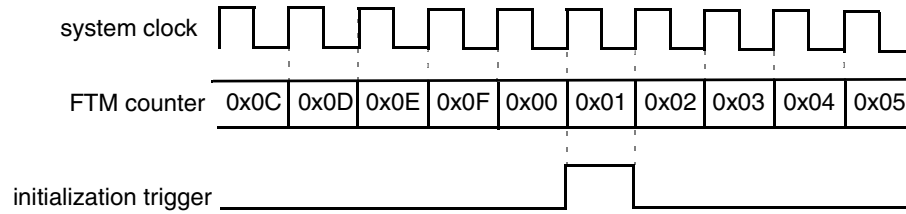
The initialization trigger is generated if (FTMEN = 1) and (INITTRIGEN = 1). The initialization trigger is generated when the value of FTMxCNTINH:L registers are loaded into FTM counter in the following cases:

- The FTMxCNTINH:L value is automatically loaded into the FTM counter after the maximum value has been reached (Figure 13-87)
- When there is a write to FTMxCNTH or FTMxCNTL register (Figure 13-88)
- When there is the FTM counter synchronization (Figure 13-89)
- If (FTMxCNTH:L = FTMxCNTINH:L), (CLKS[1:0] = 0:0), and a value different from zero is written to CLKS[1:0] bits (Figure 13-90)

The initialization trigger output provides a trigger signal that is used for on-chip modules.

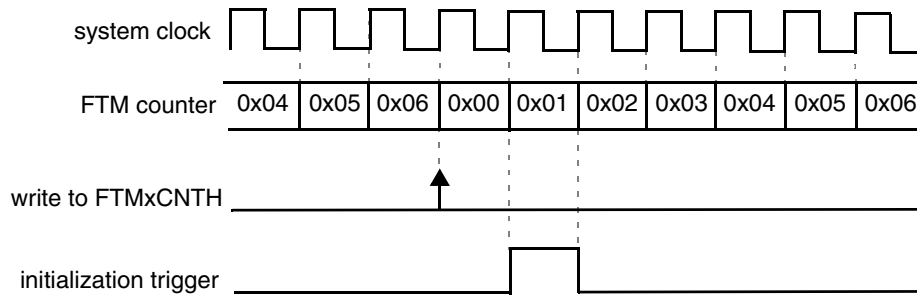


FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x000F  
 CPWMS = 0



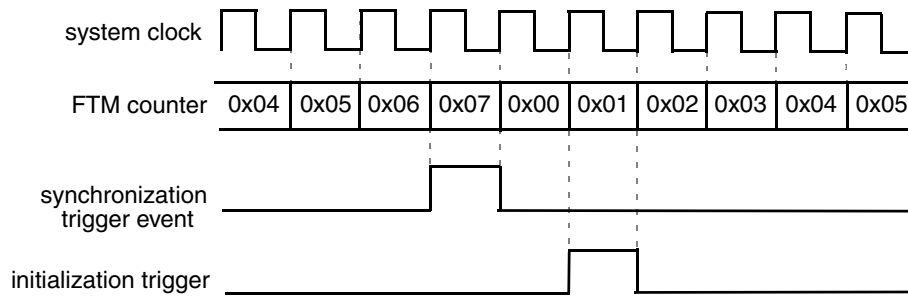
**Figure 13-87. Initialization Trigger Generated when the FTM Counting Achieved the Value of FTMxCNTINH:L**

FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x000F  
 CPWMS = 0



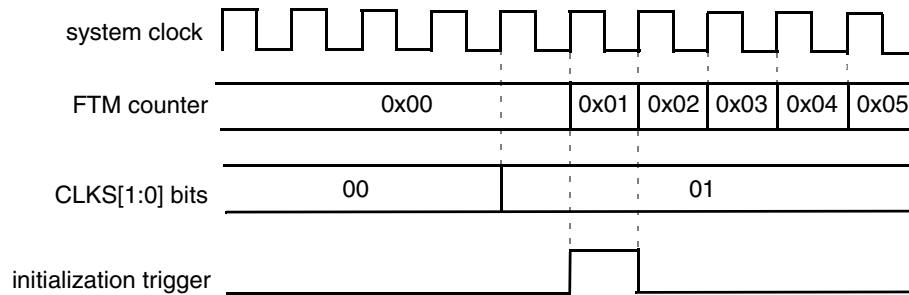
**Figure 13-88. Initialization Trigger Generated when There Is a Write to FTMxCNTH (or FTMxCNTL)**

FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x000F  
 CPWMS = 0  
 REINIT = 1



**Figure 13-89. Initialization Trigger Generated when There Is the FTM Counter Synchronization**

FTMxCNTINH:L = 0x0000  
 FTMxMODH:L = 0x000F  
 CPWMS = 0



**Figure 13-90. Initialization Trigger Generated If (FTMxCNTH:L = FTMxCNTINH:L) and (CLKS[1:0] = 0:0) and a Value Different From Zero Is Written to CLKS[1:0] Bits**

#### NOTE

- Initialization trigger is only available when (FTMEN = 1), (COMBINE = 1), and (CPWMS = 0).
- Initialization trigger with (FTMEN = 0), (COMBINE = 0), or (CPWMS = 1) is not recommended and its results are not guaranteed.

### 13.4.20 Capture Test Mode

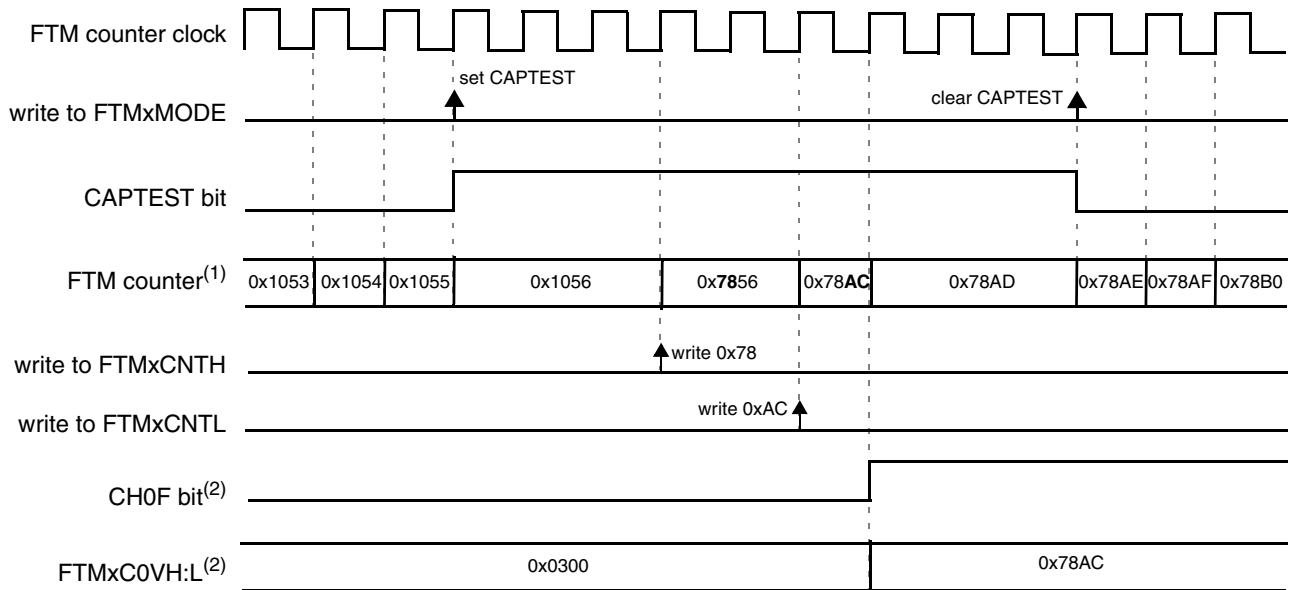
The capture test mode allows to test the FTMxCnVH:L registers, the FTM counter and the interconnection logic between the FTM counter and FTMxCnVH:L registers.

In this test mode, all channels must be configured for input capture mode (COMBINE = 0, CPWMS = 0, and MSnB:MSnA = 0:0).

When the capture test mode is enabled (CAPTEST bit is set), the FTM counter is frozen and any write to FTMxCNTH and FTMxCNTL updates directly the FTM counter (Figure 13-91). After both bytes were written (independent of the order), all FTMxCnVH:L registers are updated with the value that was written to FTMxCNTH:L registers and CHnF bits are set. Therefore, the FTM counter is updated with its next value according to its configuration (its next value depends on FTMxCNTINH:L, FTMxMODH:L, and the value that was written to FTM counter).

The next reads of FTMxCnVH:L registers return the value that was written to FTM counter and the next reads of FTMxCNTH:L register return the next value of the FTM counter.

The read coherency mechanism of FTMxCNTH:L and FTMxCnVH:L of the input capture mode remains enabled.



Notes

(1) FTM counter configuration: (CPWMS=0), (FTMxCNTINH:L=0x0000) and (FTMxMODH:L=0xFFFF)

(2) FTM channel 0 configuration: input capture mode (COMBINE=0, CPWMS=0, MSnB=0 and MSnA=0)

**Figure 13-91. Capture Test Mode**

**NOTE**

- Capture test mode is only available when (FTMEN = 1).
- Capture test mode with (FTMEN = 0) is not recommended and its results are not guaranteed.

**13.4.21 DMA**

The channel generates a DMA transfer request according to DMA and CHnIE bits ([Table 13-28](#)).

**Table 13-28. Channel DMA Transfer Request**

DMA	CHnIE	Channel DMA Transfer Request	Channel Interrupt
0	X	The channel DMA transfer request is not generated.	The channel interrupt is generated if (CHnIE = 1) and (CHnF = 1).
1	0	The channel DMA transfer request is not generated.	The channel interrupt is not generated.
1	1	The channel DMA transfer request is generated if (CHnF = 1).	The channel interrupt is not generated.

If DMA = 1, the CHnF bit is cleared either by channel DMA transfer done or reading FTMxCnSC while CHnF is set and then writing a logic 0 to CHnF bit according to CHnIE bit ([Table 13-29](#)).

**Table 13-29. Clear CHnF Bit when DMA = 1**

CHnIE	How CHnF Bit Can Be Cleared
0	CHnF bit is cleared either by the channel DMA transfer done or reading FTMxCnSC while CHnF is set and then writing a logic 0 to CHnF bit.
1	CHnF bit is cleared by the channel DMA transfer done.

## 13.4.22 TPM Emulation

This section describe the FTM features that are selected according to the FTMEN bit.

### 13.4.22.1 FTMxMODH:L and FTMxCnVH:L Synchronization

If (FTMEN = 0), then the FTMxMODH:L and FTMxCnVH:L registers are updated according to the [Section 13.4.10, “Load of the Registers With Write Buffers,”](#) and they are not updated by PWM synchronization.

If (FTMEN = 1), then the FTMxMODH:L and FTMxCnVH:L registers are updated only by PWM synchronization ([Section 13.4.11, “PWM Synchronization”](#)).

### 13.4.22.2 Free Running Counter

If (FTMEN = 0), then the FTM counter is a free running counter when (FTMxMODH:L = 0x0000) or (FTMxMODH:L = 0xFFFF) ([Section 13.4.3.3, “Free Running Counter”](#)).

If (FTMEN = 1), then the FTM counter is a free running counter when (CPWMS = 0), (FTMxCNTINH:L = 0x0000), and (FTMxMODH:L = 0xFFFF).

### 13.4.22.3 Write to FTMxSC

If (FTMEN = 0), then a write to the FTMxSC register resets the write coherency mechanism of FTMxMODH:L registers ([Section 13.3.5, “FTM Counter Modulo Registers \(FTMxMODH:FTMxMODL\)”](#)).

If (FTMEN = 1), then a write to the FTMxSC register does not reset the write coherency mechanism of FTMxMODH:L registers.

### 13.4.22.4 Write to FTMxCnSC

If (FTMEN = 0), then a write to the FTMxCnSC register resets the write coherency mechanism of FTMxCnVH:L registers ([Section 13.3.7, “FTM Channel Value Registers \(FTMxCnVH:FTMxCnVL\)”](#)).

If (FTMEN = 1), then a write to the FTMxCnSC register does not reset the write coherency mechanism of FTMxCnVH:L registers.

### 13.4.23 BDM Mode

When BDM mode is active, the FlexTimer counter and the channels output are frozen.

However, the value of FlexTimer counter or the channels output are modified in BDM mode in the following cases.

- Write any value to FTMxCNTH or FTMxCNTL registers (Section 13.4.3.4, “Counter Reset”) resets the FTM counter to the value of FTMxCNTINH:L registers and the channels output to their initial value (except for channels in output compare mode).
- The PWM synchronization with REINIT = 1 (Section 13.4.11, “PWM Synchronization”) resets the FTM counter to the value of FTMxCNTINH:L registers and the channels output to their initial value (except for channels in output compare mode).
- The initialization (Section 13.4.16, “Initialization”) forces the value of the CHnOI bit to the channel (n) output.

#### NOTE

It is not recommended to use the above cases together the fault control (Section 13.4.14, “Fault Control”). If the fault control is enabled and there is the fault condition at the enabled fault input, these cases reset the FTM counter to the value FTMxCNTINH:L and the channels output to their initial value.

## 13.5 Reset Overview

The FTM is reset whenever any MCU reset occurs.

## 13.6 FTM Interrupts

### 13.6.1 Timer Overflow Interrupt

The timer overflow interrupt is generated when (TOIE = 1) and (TOF = 1).

### 13.6.2 Channel (n) Interrupt

The channel (n) interrupt is generated when (CHnIE = 1) and (CHnF = 1).

### 13.6.3 Fault Interrupt

The fault interrupt is generated when (FAULTIE = 1) and (FAULTF = 1).

---

## Chapter 14

# Inter-Integrated Circuit (S08IICV6)

### 14.1 Introduction

The inter-integrated circuit (IIC) provides a method of communication between a number of devices. The interface is designed to operate 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  $\text{clock}/20$ , with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

#### NOTE

The SDA and SCL must not be driven above  $V_{DD}$ . These pins are pseudo open-drain containing a protection diode to  $V_{DD}$ .

## 14.1.1 Features

The IIC includes these distinctive features:

- Compatible with IIC bus standard
- Multi-master operation
- Software programmable for one of 64 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/detection
- Acknowledge bit generation/detection
- Bus busy detection
- General call recognition
- 10-bit address extension
- Support system management bus specification (SMBus), version2 programmable glitch input filter
- Address matching causes wakeup when MCU is in stop3 mode.
- DMA support

## 14.1.2 Modes of Operation

A brief description of the IIC in the various MCU modes is given here.

- Run mode — This is the basic mode of operation. To conserve power in this mode, disable the module.
- Wait mode — The module continues to operate when the MCU is in wait mode and can provide a wakeup interrupt.
- Stop mode — The IIC is inactive in stop3 mode for reduced power consumption except address matching is enabled in stop3 mode. The STOP instruction does not affect IIC register states. In stop2 mode the register contents are reset.

### 14.1.3 Block Diagram

Figure 14-1 provides a block diagram of the IIC module.

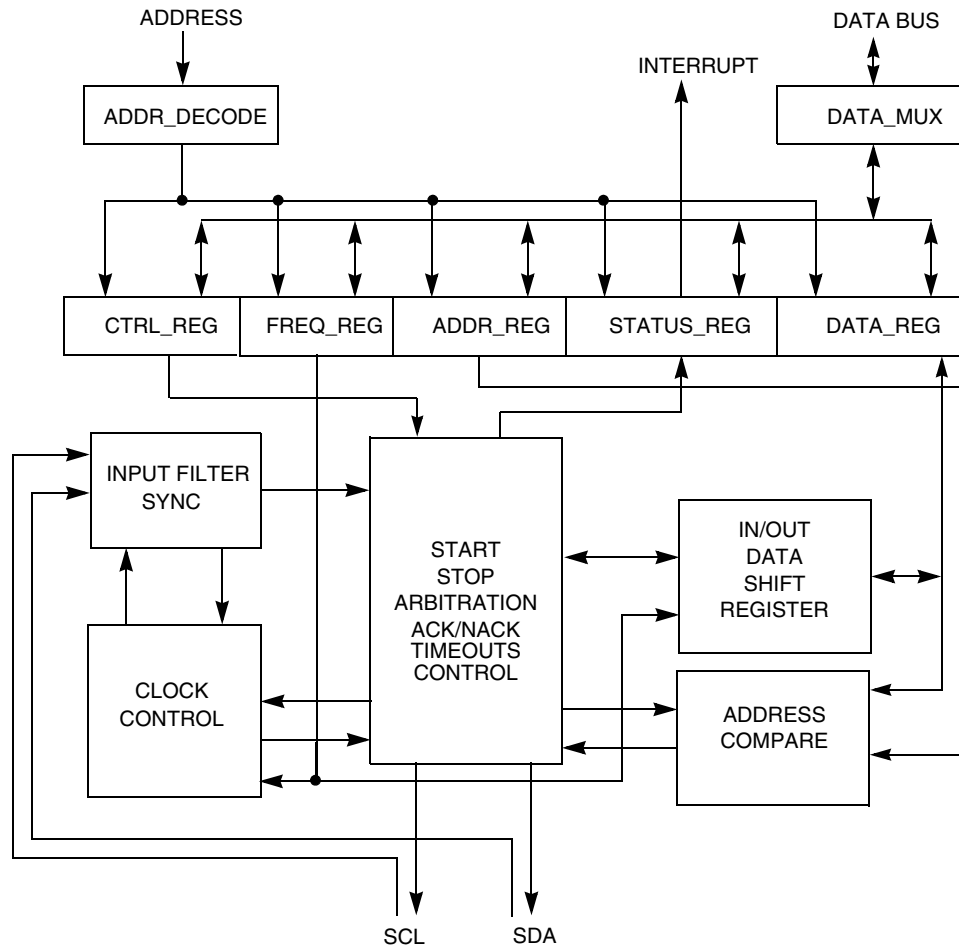


Figure 14-1. IIC Functional Block Diagram

## 14.2 External Signal Description

This section describes each user-accessible pin signal.

### 14.2.1 SCL — Serial Clock Line

The bidirectional SCL is the serial clock line of the IIC system.

### 14.2.2 SDA — Serial Data Line

The bidirectional SDA is the serial data line of the IIC system.



## 14.3 Register Definition

### 14.3.1 Module Memory Map

The IIC has ten 8-bit registers. The base address of the module is hardware programmable. The IIC register map is fixed and begins at the module's base address. [Table 14-1](#) summarizes the IIC module's address space. The following section describes the bit-level arrangement and functionality of each register.

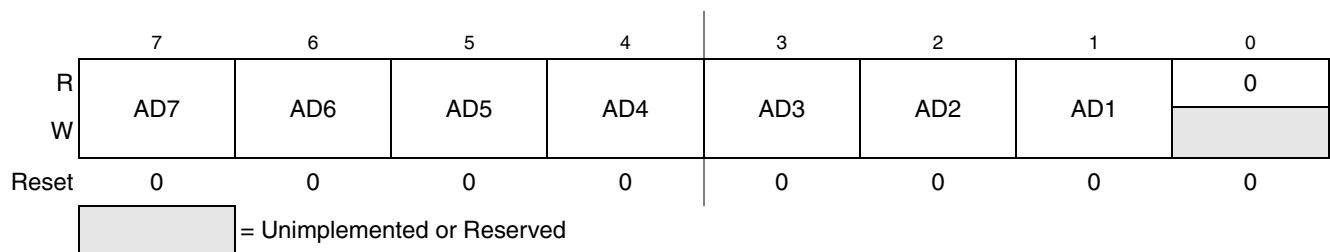
**Table 14-1. Module Memory Map**

Address	Use	Access
Base + 0x0000	IIC Address Register 1 (IICA1)	read/write
Base + 0x0001	IIC Frequency Divider Register (IICF)	read/write
Base + 0x0002	IIC Control Register 1 (IICC1)	read/write
Base + 0x0003	IIC Status Register (IICS)	read
Base + 0x0004	IIC Data IO Register (IICD)	read/write
Base + 0x0005	IIC Control Register 2 (IICC2)	read/write
Base + 0x0006	IIC input programmable filter (IICFLT)	read/write
Base + 0x0007	SMBUS IIC Control and Status Register (IICSMB)	read/write
Base + 0x0008	IIC Address Register 2 (IICA2)	read/write
Base + 0x0009	IIC SCL Low Time Out Register High (IICSLTH)	read/write
Base + 0x000A	IIC SCL Low Time Out Register Low (IICSLTL)	read/write

This section consists of the IIC register descriptions in address order.

Refer to the direct-page register summary in the [Chapter 4, "Memory,"](#) for the absolute address assignments for all IIC registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

### 14.3.2 IIC Address Register 1 (IICA1)

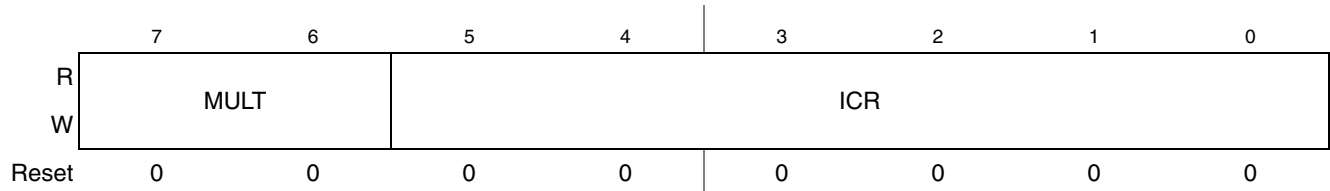


**Figure 14-2. IIC Address Register 1 (IICA1)**

**Table 14-2. IICA1 Field Descriptions**

Field	Description
7:1 AD[7:1]	<b>Slave Address 1</b> — The AD[7:1] field contains the slave address to be used by the IIC module. This field is used on the 7-bit address scheme and the lower seven bits of the 10-bit address scheme.

### 14.3.3 IIC Frequency Divider Register (IICF)



**Figure 14-3. IIC Frequency Divider Register (IICF)**

**Table 14-3. IICF Field Descriptions**

Field	Description
7:6 MULT	<b>IIC Multiplier Factor</b> — The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the IIC baud rate. The multiplier factor mul as defined by the MULT bits is provided below. 00 mul = 01 01 mul = 02 10 mul = 04 11 Reserved
5:0 ICR	<b>IIC Clock Rate</b> — The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the IIC baud rate, the SDA hold time, the SCL Start hold time and the SCL Stop hold time. <a href="#">Table 14-4</a> provides the SCL divider and hold values for corresponding values of the ICR.  The SCL divider multiplied by multiplier factor mul is used to generate IIC baud rate.  <b>IIC baud rate = bus speed (Hz)/(mul × SCL divider) <span style="float:right">Eqn. 14-1</span></b>  SDA hold time is the delay from the falling edge of SCL (IIC clock) to the changing of SDA (IIC data).  <b>SDA hold time = bus period (s) × mul × SDA hold value <span style="float:right">Eqn. 14-2</span></b>  SCL start hold time is the delay from the falling edge of SDA (IIC data) while SCL is high (Start condition) to the falling edge of SCL (IIC clock).  <b>SCL Start hold time = bus period (s) × mul × SCL Start hold value <span style="float:right">Eqn. 14-3</span></b>  SCL stop hold time is the delay from the rising edge of SCL (IIC clock) to the rising edge of SDA (IIC data) while SCL is high (Stop condition).  <b>SCL Stop hold time = bus period (s) × mul × SCL Stop hold value <span style="float:right">Eqn. 14-4</span></b>

For example, if the bus speed is 8 MHz, the table below shows the possible hold time values with different ICR and MULT selections to achieve an IIC baud rate of 100 kbps.

MULT	ICR	Hold times ( $\mu\text{s}$ )		
		SDA	SCL Start	SCL Stop
0x2	0x00	3.500	3.000	5.500
0x1	0x07	2.500	4.000	5.250
0x1	0x0B	2.250	4.000	5.250
0x0	0x14	2.125	4.250	5.125
0x0	0x18	1.125	4.750	5.125

Table 14-4. IIC Divider and Hold Values

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SCL Hold (Stop) Value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	53
17	128	21	58	65
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SCL Hold (Stop) Value
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	1022	1025
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3E	3072	513	1534	1537
3F	3840	513	1918	1921

### 14.3.4 IIC Control Register (IICC1)

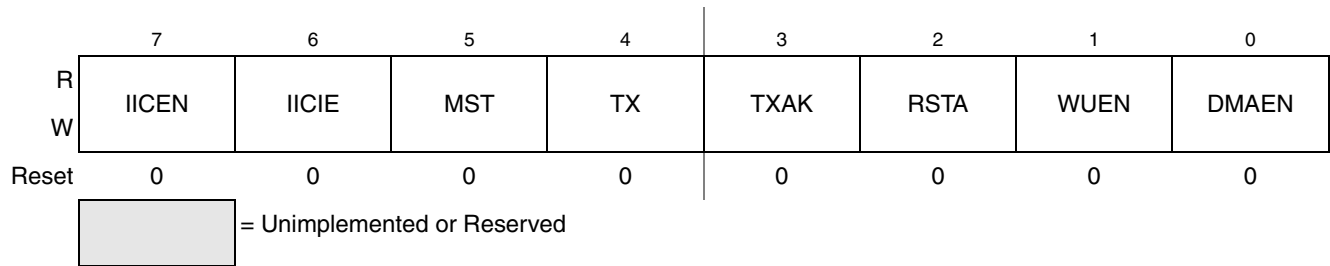


Figure 14-4. IIC Control Register (IICC1)

Table 14-5. IICC1 Field Descriptions

Field	Description
7 IICEN	<b>IIC Enable</b> — The IICEN bit determines whether the IIC module is enabled. 0 IIC is not enabled. 1 IIC is enabled.
6 IICIE	<b>IIC Interrupt Enable</b> — The IICIE bit determines whether an IIC interrupt is requested. 0 IIC interrupt request not enabled. 1 IIC interrupt request enabled.
5 MST	<b>Master Mode Select</b> — When the MST bit is changed from 0 to 1, a START signal is generated on the bus and master mode is selected. When this bit changes from 1 to 0 a STOP signal is generated and the mode of operation changes from master to slave. 0 Slave mode. 1 Master mode.
4 TX	<b>Transmit Mode Select</b> — The TX bit selects the direction of master and slave transfers. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. 0 Receive. 1 Transmit.
3 TXAK	<b>Transmit Acknowledge Enable</b> — This bit specifies the value driven onto the SDA during data acknowledge cycles for both master and slave receivers. There are two conditions that effect NACK/ACK generation. If FACK (fast NACK/ACK) is cleared, 0 An acknowledge signal is sent to the bus on the following receiving data byte. 1 No acknowledge signal response is sent to the bus on the following receiving data byte. If FACK bit is set, no ACK or NACK is sent after receiving one data byte until this TXAK bit is written 0 An acknowledge signal is sent out to the bus on the current receiving data byte 1 No acknowledge signal response is sent to the bus on the current receiving data byte <b>Note:</b> SCL is held to low until TXAK is written.
2 RSTA (Write Only read always 0)	<b>Repeat START</b> — Writing 1 to this bit generates a repeated START condition provided it is the current master. Attempting a repeat at the wrong time results in loss of arbitration. 0 No repeat start detected in bus operation. 1 Repeat start generated.

Table 14-5. IICC1 Field Descriptions (continued)

Field	Description
1 WUEN	<p><b>Wakeup Enable</b> — IIC can wake the MCU from stop3 mode when slave address or general call address matching occurs.</p> <p>0 Normal operation. No interrupt generated when address matching in stop3 mode.</p> <p>1 Enables the wakeup function in stop3 mode.</p>
0 DMAEN	<p><b>DMA Enable</b> — The DMAEN bit enables or disables the DMA function.</p> <p>0 All DMA signalling disabled.</p> <p>1 DMA transfer is enabled and the following conditions trigger the DMA request:</p> <ul style="list-style-type: none"> <li>• While FACK = 0, a data byte is received, either address or data is transmitted. (ACK/NACK automatic)</li> <li>• While FACK = 0, the first byte received matches IICA1 register or is general call address.</li> </ul> <p>If any address matching occurs then IAAS and TCIF are set. If the direction of transfer is known from master to slave then it is not required to check the SRW. With this assumption, DMA can also be utilized here. In other case, if the master reads data from slave then it is required to rewrite the IICC1 operation. With this assumption, DMA can not be used here.</p> <p>When FACK = 1, an address or a data byte is transmitted.</p>

## 14.3.5 IIC Status Register (IICS)

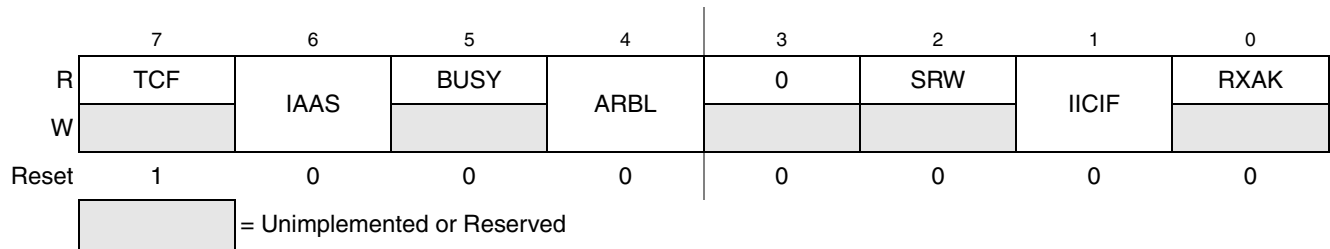


Figure 14-5. IIC Status Register (IICS)

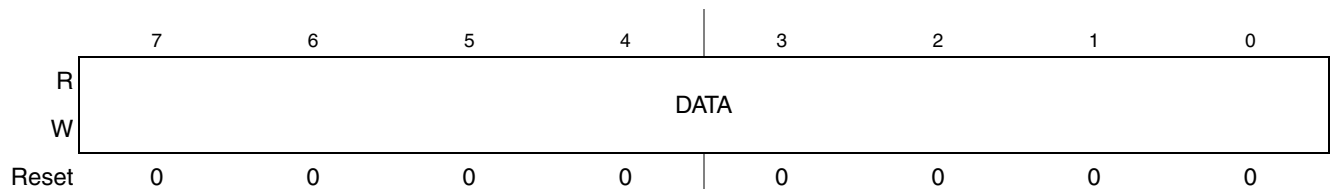
Table 14-6. IICS Field Descriptions

Field	Description
7 TCF	<b>Transfer Complete Flag</b> — This bit is set on the completion of a byte and acknowledge bit transfer. Note that this bit is only valid during or immediately following a transfer to the IIC module or from the IIC module. The TCF bit is cleared by reading the IICD register in receive mode or writing to the IICD in transmit mode. 0 Transfer in progress. 1 Transfer complete.
6 IAAS	<b>Addressed as a Slave</b> — The IAAS bit is set when one of the following conditions is met: <ul style="list-style-type: none"> <li>When the calling address matches the programmed slave address</li> <li>If the GCAEN bit is set and a general call is received</li> <li>If SIICAEN bit is set, when the calling address matches the 2nd programmed slave address</li> <li>If ALERTEN bit is set and SMBus alert response address is received</li> </ul> This bit is set before ACK bit. The CPU needs to check the SRW bit and set TX/RX bit accordingly. Writing the IICC1 register with any value clears this bit. 0 Not addressed. 1 Addressed as a slave.
5 BUSY	<b>Bus Busy</b> — The BUSY bit indicates the status of the bus regardless of slave or master mode. The BUSY bit is set when a START signal is detected and cleared when a STOP signal is detected. 0 Bus is idle. 1 Bus is busy.
4 ARBL	<b>Arbitration Lost</b> — This bit is set by hardware when the arbitration procedure is lost. The ARBL bit must be cleared by software or by writing 1 to it. 0 Standard bus operation. 1 Loss of arbitration.
2 SRW	<b>Slave Read/Write</b> — When addressed as a slave, the SRW bit indicates the value of the $R/\bar{W}$ command bit of the calling address sent to the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.

**Table 14-6. IICS Field Descriptions (continued)**

Field	Description
1 IICIF	<p><b>IIC Interrupt Flag</b> — The IICIF bit is set when an interrupt is pending. This bit must be cleared by software or by writing a 1 to it in the interrupt routine. One of the following events can set the IICIF bit:</p> <ul style="list-style-type: none"> <li>• One byte transfer including ACK/NACK bit completes if FACK = 0</li> <li>• One byte transfer excluding ACK/NCAK bit completes if FACK = 1. an ACK or NACK is sent on the bus by writing 0 or 1 to TXAK after this bit is set as receive mode.</li> <li>• Match of slave addresses to calling address including primary slave address, general call address, alert response address, and second slave address. (When address matching happens in stop3 mode, the IICIF is cleared automatically after core gets out of STOP.)</li> <li>• Arbitration lost</li> <li>• Timeouts in SMBus mode except both SCL and SDA high timeout</li> </ul> <p>0 No interrupt pending. 1 Interrupt pending.</p>
0 RXAK	<p><b>Receive Acknowledge</b> — When the RXAK bit is low, it indicates an acknowledge signal has been received after the completion of one byte of data transmission on the bus. If the RXAK bit is high it means that no acknowledge signal is detected.</p> <p>0 Acknowledge received. 1 No acknowledge received.</p>

### 14.3.6 IIC Data I/O Register (IICD)



**Figure 14-6. IIC Data I/O Register (IICD)**

**Table 14-7. IICD Field Descriptions**

Field	Description
7:0 DATA	<p><b>Data</b> — In master transmit mode, when data is written to the IICD, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates receiving of the next byte of data.</p>

#### NOTE

When transitioning out of master receive mode, the IIC mode must be switched before reading the IICD register to prevent an inadvertent initiation of a master receive data transfer.

In slave mode, the same functions are available after an address match has occurred.

TX bit in IICC must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the IIC is configured for master transmit but a master receive is desired, then reading the IICD does not initiate the receive.



Reading the IICD returns the last byte received while the IIC is configured in either master receive or slave receive modes. The IICD does not reflect every byte that is transmitted on the IIC bus, nor can software verify that a byte has been written to the IICD correctly by reading it back.

In master transmit mode, the first byte of data written to IICD following assertion of MST (start bit) or assertion of RSTA bit (repeated start) is used for the address transfer and must comprise of the calling address (in bit 7 to bit 1) concatenated with the required  $R/\overline{W}$  bit (in position bit 0).

### 14.3.7 IIC Control Register 2 (IICC2)

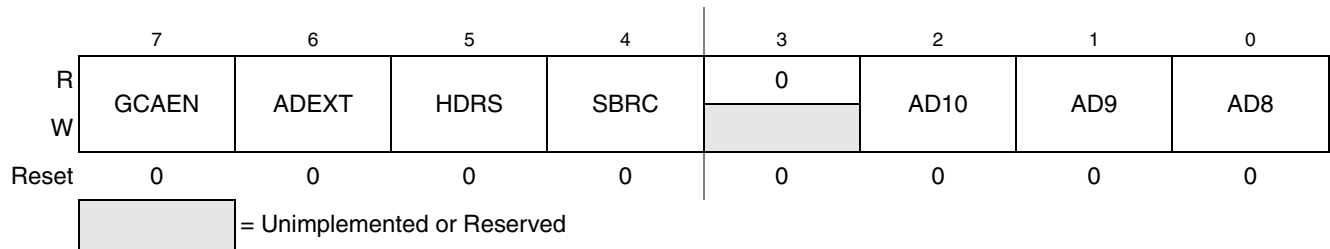


Figure 14-7. IIC Control Register 2 (IICC2)

Table 14-8. IICC2 Field Descriptions

Field	Description
7 GCAEN	<b>General Call Address Enable</b> — The GCAEN bit enables or disables general call address. 0 General call address is disabled. 1 General call address is enabled.
6 ADEXT	<b>Address Extension</b> — The ADEXT bit controls the number of bits used for the slave address. 0 7-bit address scheme. 1 10-bit address scheme.
5 HDRS	<b>High Drive Select</b> — The HDRS bit control the PAD's drive capability. 0 Normal drive mode. 1 High drive mode.
4 SBRC	<b>Slave Baud Rate Control</b> — Alternative slave mode baud rate is independent from master baud rate at max frequency. This forces clock stretching only on SCL line on very fast IIC modes. 0 No control on slave baud rate that follows master baud rate and clock stretch works. 1 Slave baud rate is independent from master baud rate.
2:0 AD[10:8]	<b>Slave Address</b> — The AD[10:8] field contains the upper three bits of the slave address in the 10-bit address scheme. This field is only valid when the ADEXT bit is set.

### 14.3.8 IIC Programmable Input Glitch Filter (IICFLT)

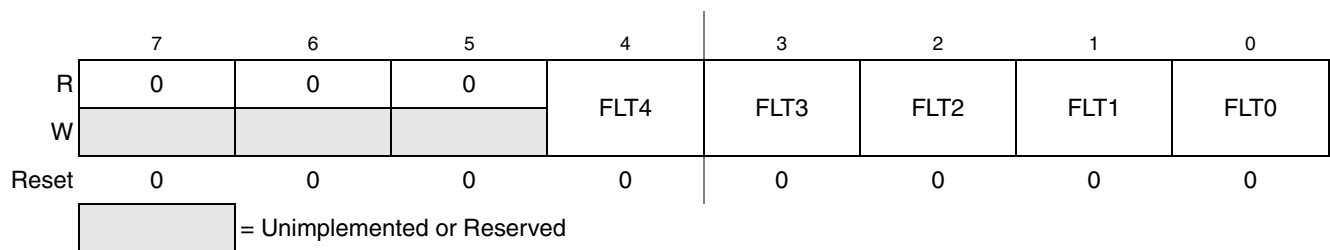


Figure 14-8. IIC Programmable Input Glitch Filter Register (IICFLT)

Table 14-9. IICFLT Field Descriptions

Field	Description
4:0 FLT	<p>IIC Programmable Filter Factor contains the programming controls for the width of glitch (in terms of bus clock cycles) the filter must absorb; in other words, the filter does not let glitches less than or equal to this width setting pass. For instance: FLT[3:0]</p> <p>0000 No Filter / Bypass  0001 Filter glitches up to width of 1 (half) IPBUS clock cycle  0010 Filter glitches up to width of 2 (half) IPBUS clock cycles  0011 Filter glitches up to width of 3 (half) IPBUS clock cycles  0100 Filter glitches up to width of 4 (half) IPBUS clock cycles  0101 Filter glitches up to width of 5 (half) IPBUS clock cycles  0110 Filter glitches up to width of 6 (half) IPBUS clock cycles  0111 Filter glitches up to width of 7 (half) IPBUS clock cycles  1000 Filter glitches up to width of 8 (half) IPBUS clock cycles  1001 Filter glitches up to width of 9 (half) IPBUS clock cycles  1010 Filter glitches up to width of 10 (half) IPBUS clock cycles  1011 Filter glitches up to width of 11 (half) IPBUS clock cycles  1100 Filter glitches up to width of 12 (half) IPBUS clock cycles  1101 Filter glitches up to width of 13 (half) IPBUS clock cycles  1110 Filter glitches up to width of 14 (half) IPBUS clock cycles  1111 Filter glitches up to width of 15 (half) IPBUS clock cycles</p> <p><b>Note:</b> The width of the FLT is an integration option which can be changed in different SoCs. Also the clock source used is an integration configurative option - It could be the 2X IPBus clock or the IPbus clock - which one needs to be identified at architectural definition. For the 4-bit definitions above, hard descriptions of "half" IPBUS clock cycles is not the case when the IPBUS clock is used for filtering logic.</p>

### 14.3.9 IIC SMBus Control and Status Register (IICSMB)

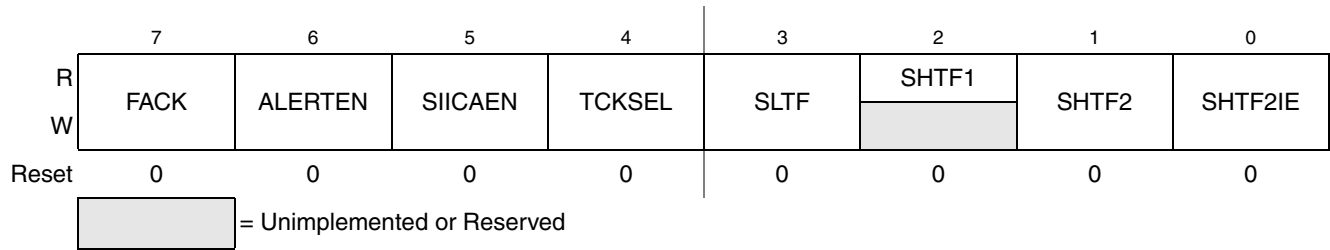


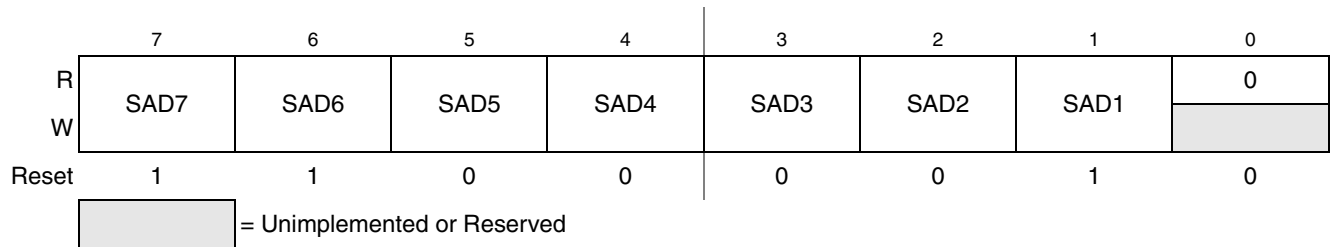
Figure 14-9. IIC SMBus Control and Status Register (IICSMB)

Table 14-10. IICSMB Field Descriptions

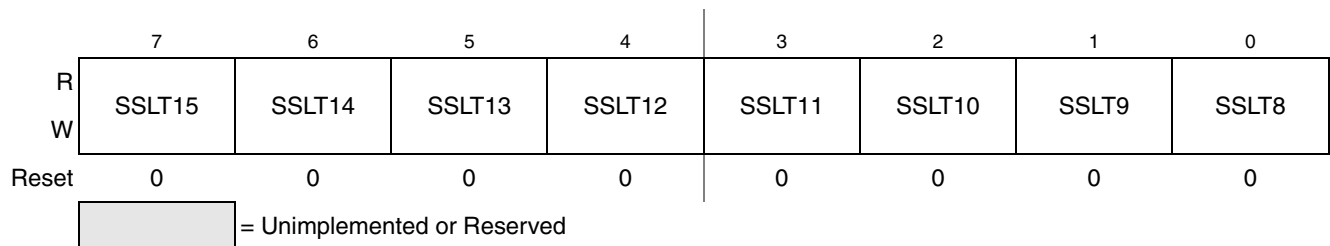
Field	Description
7 FACK	<b>Fast NACK/ACK enable</b> — For SMBus packet error checking, CPU must be able to issue an ACK or NACK according to the result of receiving data byte. 0 ACK or NACK is sent out on the following receiving data byte. 1 Writing 0 to TXAK after receiving data byte generates an ACK; writing 1 to TXAK after receiving data byte generates a NACK.
6 ALERTEN	<b>SMBus Alert Response Address Enable</b> — The ALERTEN bit enables or disable SMBus alert response address. 0 SMBus alert response address matching is disabled 1 SMBus alert response address matching is enabled.
5 SIICAEN	<b>Second IIC Address Enable</b> — The SIICAEN bit enables or disable SMBus device default address. 0 IIC address register 2 matching is disabled. 1 IIC address register 2 matching is enabled.
4 TCKSEL	<b>Time Out Counter Clock Select</b> — This bit selects the clock sources of time out counter 0 Time out counter counts at bus/64 frequency. 1 Time out counter counts at the bus frequency.
3 SLTF	<b>SCL Low Timeout Flag</b> — This bit is set to logic 1 when IICSLT loaded non zero value (LoValue) and a SCL low time out occurs. This bit is cleared by software, by writing a logic 1 to it 0 No LOW TIME OUT occurs. 1 LOW TIME OUT occurs. <b>Note:</b> LOW TIME OUT function is disabled when IIC SCL low timer out register is set to zero.
2 SHTF1	<b>SCL High Timeout Flag 1</b> — This read-only bit is set to logic 1 when SCL and SDA are held high more than clock × LoValue/512, which indicates the Bus Free. This bit is cleared automatically. 0 No SCL high and SDA high TIMEOUT occurs. 1 SCL high and SDA high TIMEOUT occurs.
1 SHTF2	<b>SCL High Timeout Flag 2</b> — This bit is set to logic 1 when SCL is held high and SDA is held low more than clock × LoValue/512. This bit is cleared by software, by writing a logic 1 to it 0 No SCL high and SDA low TIMEOUT occurs. 1 SCL high and SDA low TIMEOUT occurs.
0 SHTF2IE	<b>SHTF2 Interrupt enable</b> — This bit is Interrupt enable for SCL high and SDA low timeout. 0 SHTF2 interrupt is disabled. 1 SHTF2 interrupt is enabled.

**NOTE**

- A master assumes that the bus is free when detecting the clock and data signals being high for greater than high time out period. However, the SHTF1 rises in bus transmission process with idle bus state.
- When TCKSEL=1, there is no meaning to monitor SHTF1 since the bus speed is too high to match the protocol of SMBus.

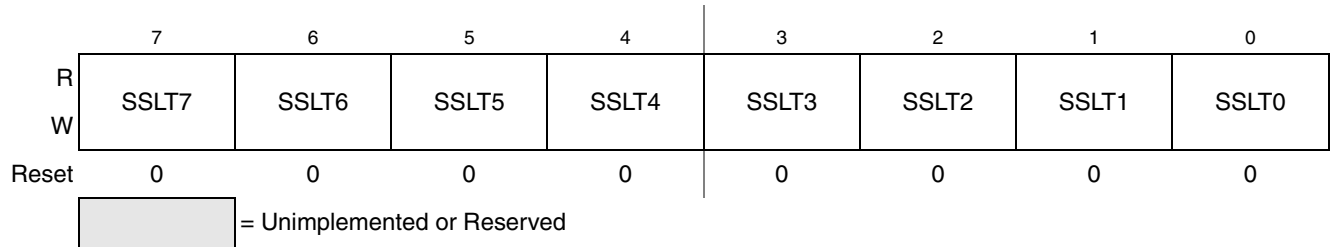
**14.3.10 IIC Address Register 2 (IICA2)**

Field	Description
7:1 SAD[7:1]	<b>SMBus Address</b> — The AD[7:1] field contains the slave address to be used by the SMBus. This field is used on the device default address or other related addresses.

**14.3.11 IIC SCL Low Time Out Register High (IICSLTH)****Figure 14-10. IIC SCL Low Time Out Register High (IICSLTH)****Table 14-11. IICSLTH Field Descriptions**

Field	Description
7:0 SSLT[15:8]	The value in this register is the most significant byte of SCL low time out value that determines the time-out period of SCL low.

### 14.3.12 IIC SCL Low Time Out register Low (IICSLTL)



**Figure 14-11. IIC SCL Low Time Out register Low (IICSLTL)**

**Table 14-12. IICSLTL Field Descriptions**

Field	Description
7:0 SSLT[7:0]	The value in this register is the least significant byte of SCL low time out value that determines the timeout period of SCL low.

## 14.4 Functional Description

This section provides a complete functional description of the IIC module.

### 14.4.1 IIC Protocol

The IIC bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A logic AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts:

- START signal
- Slave address transmission
- Data transfer
- STOP signal

The STOP signal should not be confused with the CPU STOP instruction. The IIC bus system communication is described briefly in the following sections and is illustrated in [Figure 14-12](#).

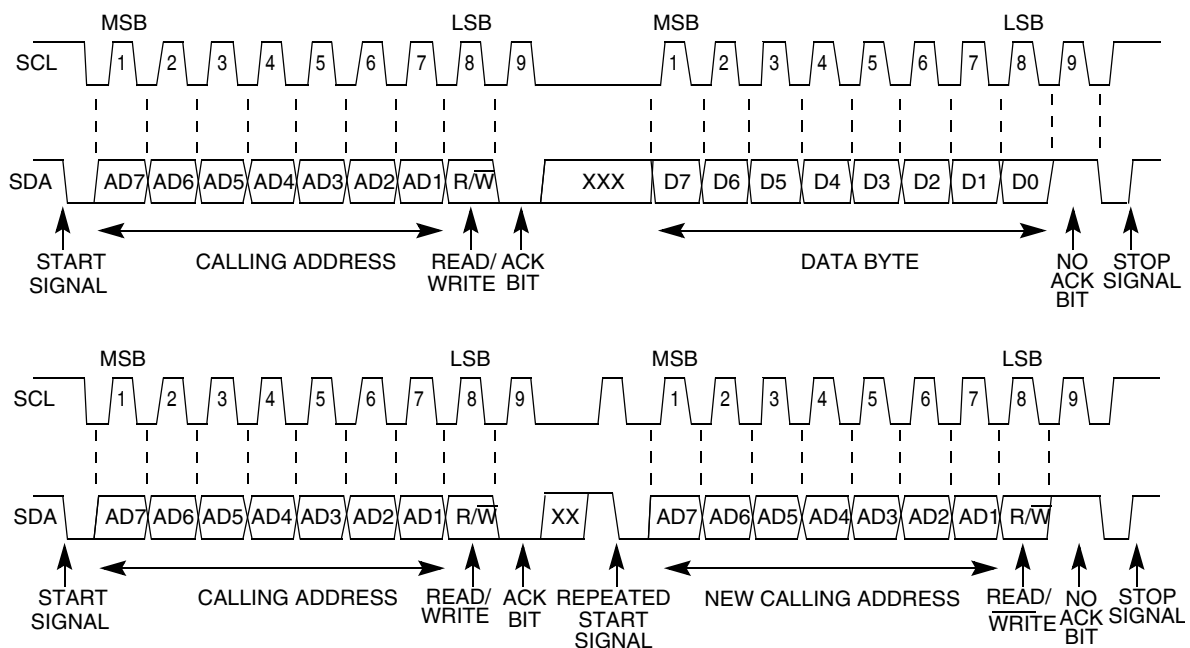


Figure 14-12. IIC Bus Transmission Signals

### 14.4.1.1 START Signal

When the bus is free that is, no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in [Figure 14-12](#), a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

### 14.4.1.2 Slave Address Transmission

The first byte of data transferred immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a  $\overline{R/W}$  bit. The  $\overline{R/W}$  bit tells the slave the desired direction of data transfer.

- 1 = Read transfer, the slave transmits data to the master.
- 0 = Write transfer, the master transmits data to the slave.

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see [Figure 14-12](#)).

No two slaves in the system may have the same address. If the IIC module is the master, it must not transmit an address that is equal to its own slave address. The IIC cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the IIC reverts to slave mode and operates correctly even if it is being addressed by another master.

### 14.4.1.3 Data Transfer

Before successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the  $\overline{R/W}$  bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in [Figure 14-12](#). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte is followed by a 9th (acknowledge) bit that is signalled from the receiving device. An acknowledge is signalled by pulling the SDA low at the ninth clock. In summary, one complete data transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master in the 9th bit time, the SDA line must be left high by the slave. The master interprets the failed acknowledge as an unsuccessful data transfer.

If the master receiver does not acknowledge the slave transmitter after a data byte transmission, the slave interprets this as an end of data transfer and releases the SDA line.

In either case, the data transfer is aborted and the master does one of two things:

- Relinquishes the bus by generating a STOP signal.
- Commences a new calling by generating a repeated START signal.



#### 14.4.1.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL at logical 1 (see [Figure 14-12](#)).

The master can generate a STOP even if the slave has generated an acknowledge at which point the slave must release the bus.

#### 14.4.1.5 Repeated START Signal

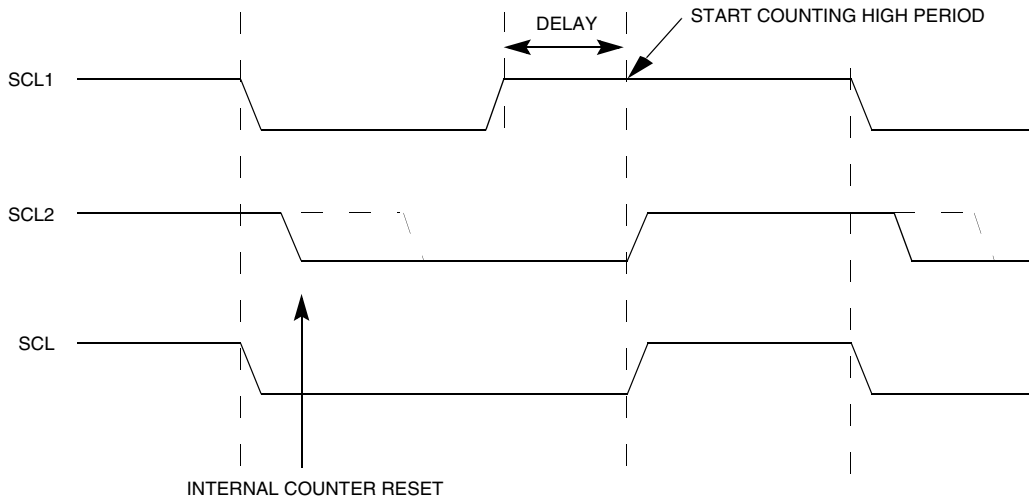
As shown in [Figure 14-12](#), a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

#### 14.4.1.6 Arbitration Procedure

The IIC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure, a bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

#### 14.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see [Figure 14-13](#)). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.



**Figure 14-13. IIC Clock Synchronization**

### 14.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

### 14.4.1.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

## 14.4.2 10-bit Address

For 10-bit addressing, 0x11110 is used for the first 5 bits of the first address byte. Various combinations of read/write formats are possible within a transfer that includes 10-bit addressing.

### 14.4.2.1 Master-Transmitter Addresses a Slave-Receiver

The transfer direction is not changed (see Table 14-13). When a 10-bit address follows a START condition, each slave compares the first seven bits of the first byte of the slave address (11110XX) with its own address and tests whether the eighth bit ( $R/\overline{W}$  direction bit) is 0. It is possible that more than one device finds a match and generates an acknowledge (A1). Each slave that finds a match compares the eight bits of the second byte of the slave address with its own address, but only one slave finds a match and generate an acknowledge (A2). The matching slave remains addressed by the master until it receives a STOP condition (P) or a repeated START condition (Sr) followed by a different slave address.

S	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 0	A1	Slave Address 2nd byte AD[8:1]	A2	Data	A	...	Data	A/A	P
---	--	----------	----	-----------------------------------	----	------	---	-----	------	-----	---

**Table 14-13. Master-Transmitter Addresses Slave-Receiver with a 10-bit Address**

After the master-transmitter has sent the first byte of the 10-bit address, the slave-receiver sees an IIC interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

### 14.4.2.2 Master-Receiver Addresses a Slave-Transmitter

The transfer direction is changed after the second  $R/\overline{W}$  bit (see Table 14-14). Up to and including acknowledge bit A2, the procedure is the same as that described for a master-transmitter addressing a slave-receiver. After the repeated START condition (Sr), a matching slave remembers that it was addressed before. This slave then checks whether the first seven bits of the first byte of the slave address following Sr are the same as they were after the START condition (S), and tests whether the eighth ( $R/\overline{W}$ ) bit is 1. If there is a match, the slave considers that it has been addressed as a transmitter and generates acknowledge A3. The slave-transmitter remains addressed until it receives a STOP condition (P) or a repeated START condition (Sr) followed by a different slave address.

After a repeated START condition (Sr), all other slave devices also compare the first seven bits of the first byte of the slave address with their own addresses and test the eighth ( $R/\overline{W}$ ) bit. However, none of them are addressed because  $R/\overline{W} = 1$  (for 10-bit devices), or the 11110XX slave address (for 7-bit devices) does not match.

S	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 0	A1	Slave Address 2nd byte AD[8:1]	A2	Sr	Slave Address 1st 7 bits 11110 + AD10 + AD9	R/W 1	A3	Data	A	...	Data	A	P
---	--	----------	----	-----------------------------------	----	----	--	----------	----	------	---	-----	------	---	---

**Table 14-14. Master-Receiver Addresses a Slave-Transmitter with a 10-bit Address**

After the master-receiver has sent the first byte of the 10-bit address, the slave-transmitter sees an IIC interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

### 14.4.3 Address Matching

All received addresses can be requested in 7-bit or 10-bit address. IIC address register 1 that contains IIC primary slave address, always participates the address matching process. If the GCAEN bit is set, general call participates the address matching process. If the ALERTEN bit is set, alert response participates the address matching process. If SIICAEN bit is set, the IIC address register 2 participates the address matching process.

When the IIC responds to one of the above mentioned address, it acts as a slave-receiver and the IAAS bit is set after the address cycle. Software need to read the IICD register after the first byte transfer to determine that the address is matched.

### 14.4.4 System Management Bus Specification

SMBus provides a control bus for system and power management related tasks. A system may use SMBus to pass messages to and from devices instead of tripping individual control lines. Removing the individual control lines reduces pin count. Accepting messages ensures future expandability. With system management bus, a device can provide manufacturer information, tell the system what its model/part number is, save its state for a suspend event, report different types of errors, accept control parameters, and return its status.

#### 14.4.4.1 Timeouts

The  $T_{\text{TIMEOUT,MIN}}$  parameter allows a master or slave to conclude that a defective device is holding the clock low indefinitely or a master is intentionally trying to drive devices off the bus. It is highly recommended that a slave device release the bus (stop driving the bus and let SCL and SDA float high) when it detects any single clock held low longer than  $T_{\text{TIMEOUT,MIN}}$ . Devices that have detected this condition must reset their communication and be able to receive a new START condition in no later than  $T_{\text{TIMEOUT,MAX}}$ .

SMBus defines a clock low time out,  $T_{\text{TIMEOUT}}$  of 35 ms and specifies  $T_{\text{LOW:SEXT}}$  as the cumulative clock low extend time for a slave device and specifies  $T_{\text{LOW:MEXT}}$  as the cumulative clock low extend time for a master device.

##### 14.4.4.1.1 SCL Low Timeout

If the SCL line is held low by a slave device on the bus, no further communication is possible. Furthermore, the master cannot force the SCL line high to correct the error condition. To solve this problem, the SMBus protocol specifies that devices participating in a transfer must detect any clock cycle held low longer than a timeout value condition. Devices that have detected the timeout condition must reset the communication. When active master, if the IIC detects that SMBCLK low has exceeded the value of  $T_{\text{TIMEOUT,MIN}}$  it must generate a stop condition within or after the current data byte in the transfer process. When slave, upon detection of the  $T_{\text{TIMEOUT,MIN}}$  condition, the IIC shall reset its communication and be able to receive a new START condition.

### 14.4.4.1.2 SCL High Timeout

The IIC shall assume that the bus is idle, when it has determined that the SMBCLK and SMBDAT signals have been high for at least  $T_{HIGH:MAX}$ . HIGH timeout can occur in two ways:

1. HIGH timeout detected after a STOP condition appears on the bus.
2. HIGH timeout detected after a START condition, but before a STOP condition appears on the bus.

Any master detecting either scenario can assume the bus is free then SHTF1 rises. HIGH timeout occurred in scenario 2 if it ever detects that both the following is true: BUSY bit is high and SHTF1 is high.

When SMBDAT signal is low and SMBCLK signal is high for a period of time, the other kind of time out occurs. The time period needs to be defined in software. SHTF2 is used as the flag when limited time reaches. This flag is also an interrupt resource, therefore it also triggers IICIF.

### 14.4.4.1.3 CSMBCLK TIMEOUT MEXT

Figure 14-14 illustrates the definition of the timeout intervals,  $T_{LOW:SEXT}$  and  $T_{LOW:MEXT}$ . When master mode, the IIC must not cumulatively extend its clock cycles for a period greater than  $T_{LOW:MEXT}$  within a byte, where each byte is defined as START-to-ACK, ACK-to-ACK, or ACK-to-STOP. When CSMBCLK TIMEOUT MEXT occurs, SMBus MEXT rises and also triggers the SLTF.

### 14.4.4.1.4 CSMBCLK TIMEOUT SEXT

A master is allowed to abort the transaction in progress to any slave that violates the  $T_{LOW:SEXT}$  or  $T_{TIMEOUT,MIN}$  specifications. This can be accomplished by the master issuing a STOP condition at the conclusion of the byte transfer in progress. When slave, the IIC must not cumulatively extend its clock cycles for a period greater than  $T_{LOW:SEXT}$  during any message from the initial START to the STOP. When CSMBCLK TIMEOUT SEXT occurs, SEXT rises and also triggers SLTF.

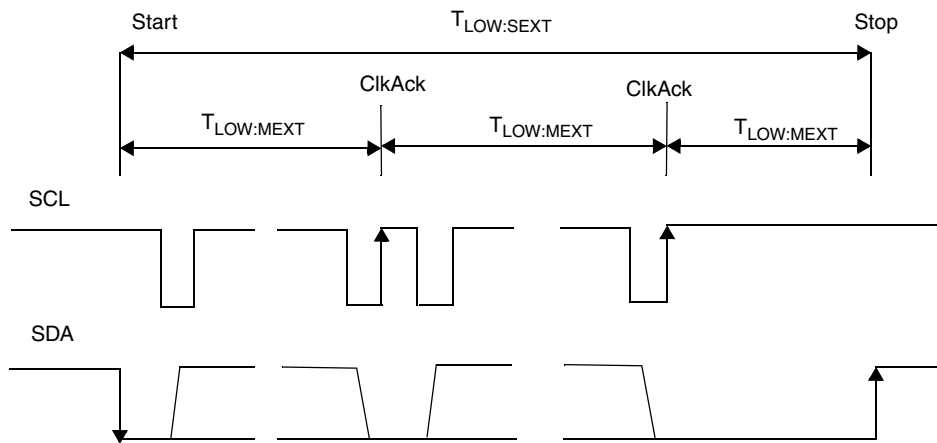


Figure 14-14. Timeout measurement intervals

#### NOTE

CSMBCLK TIMEOUT SEXT and MEXT are optional functions that are implemented in second step.

### 14.4.4.2 FAST ACK and NACK

To improve reliability and communication robustness, implementation of packet error checking (PEC) by SMBus devices is optional for SMBus devices but required for devices participating in and only during the address resolution protocol (ARP) process. The PEC is a CRC-8 error checking byte, calculated on all the message bytes. The PEC is appended to the message by the device that supplied the last data byte. If the PEC is present but not correct, a NACK is issued by receiver. Otherwise an ACK is issued. In order to calculate the CRC-8 by software, this module can hold SCL line to low after receiving eighth SCL (bit 8th) if this byte is a data byte. So software can determine whether an ACK or NACK should be sent out to the bus by setting or clearing TXAK bit if FASK (fast ACK/NACK enable bit) is enabled.

SMBus requires devices to acknowledge their own address always, as a mechanism to detect a removable devices presence on the bus (battery, docking station, etc.) Besides to indicate a slave device busy condition, SMBus is using the NACK mechanism also to indicate the reception of an invalid command or data. Since such a condition may occur on the last byte of the transfer, it is required that SMBus devices have the ability to generate the not acknowledge after the transfer of each byte and before the completion of the transaction. This is important because SMBus does not provide any other resend signaling. This difference in the use of the NACK signaling has implications on the specific implementation of the SMBus port, especially in devices that handle critical system data such as the SMBus host and the SBS components.

#### NOTE

In the last byte of master receive slave transmit mode, the master must send NACK to bus, so FACK must be switched off before the last byte transmit.

## 14.5 Resets

The IIC is disabled after reset. The IIC cannot cause an MCU reset.

## 14.6 Interrupts

The IIC generates a single interrupt.

An interrupt from the IIC is generated when any of the events in [Table 14-15](#) occur, provided the IICIE bit is set. The interrupt is driven by bit IICIF (of the IIC status register) and masked with bit IICIE (of the IIC control register). The IICIF bit must be cleared by software by writing 1 to it in the interrupt routine. You can determine the interrupt type by reading the status register. For SMBus timeouts interrupt, the interrupt is driven by SLTF and masked with bit IICIE. The SLTF bit must be cleared by software by writing 1 to it in the interrupt routine. You can determine the interrupt type by reading the status register.

**NOTE**

In master receive mode, the FACK must be set to zero before the last byte transfer.

**Table 14-15. Interrupt Summary**

Interrupt Source	Status	Flag	Local Enable
Complete 1-byte transfer	TCF	IICIF	IICIE
Match of received calling address	IAAS	IICIF	IICIE
Arbitration lost	ARBL	IICIF	IICIE
SMBus SCL low timeout interrupt flag	SLTF	IICIF	IICIE
SMBus SCL high SDA low timeout interrupt flag	SHTF2	IICIF	IICIE&SHTF2IE
Wakeup form sop3 interrupt	IAAS	IICIF	IICIE&WUEN

**14.6.1 Byte Transfer Interrupt**

The TCF (transfer complete flag) bit is set at the falling edge of the 9th clock to indicate the completion of byte and acknowledge transfer. When FACK is enabled, TCF is then set at the falling edge of 8th clock to indicate the completion of byte.

**14.6.2 Address Detect Interrupt**

When the calling address matches the programmed slave address (IIC address register) or when the GCAEN bit is set and a general call is received, the IAAS bit in the status register is set. The CPU is interrupted, provided the IICIE is set. The CPU must check the SRW bit and set its Tx mode accordingly.

**14.6.3 Exit from Low-Power/Stop Modes**

The slave receive input detect circuit and address matching feature are still active on low power modes (wait and stop). An asynchronous input matching slave address or general call address brings the CPU out of low power/stop mode if the interrupt is not masked. Therefore, TCF and IAAS both can trigger this interrupt.

**14.6.4 Arbitration Lost Interrupt**

The IIC is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, the relative priority of the contending masters is determined by a data arbitration procedure. The IIC module asserts this interrupt when it loses the data arbitration process and the ARBL bit in the status register is set.

Arbitration is lost in the following circumstances:

- SDA sampled as a low when the master drives a high during an address or data transmit cycle.
- SDA sampled as a low when the master drives a high during the acknowledge bit of a data receive cycle.
- A START cycle is attempted when the bus is busy.

- A repeated START cycle is requested in slave mode.
- A STOP condition is detected when the master did not request it.

This bit must be cleared by software by writing a 1 to it.

### 14.6.5 Timeouts Interrupt in SMBus

When IICIE is set, the IIC asserts a timeout interrupt outputs SLTF and SHTF2 upon detection of any of the mentioned timeout conditions, with one exception. The SCL high and SDA high TIMEOUT mechanism must not be used to influence the timeout interrupt output, because this timeout indicates an idle condition on the bus. SHTF1 rises when it matches the SCL high and SDA high TIMEOUT and fall automatically to just indicate the bus status. The SHTF2's timeout period is same as SHTF1 that is short as compared to SLTF, so another control bit SHTF2IE is added to enable or disable it.

### 14.6.6 Programmable Input Glitch Filter

An IIC glitch filter has been added outside the IIC legacy modules, but within the IIC package. This filter can absorb glitches on the IIC clock and data lines for IIC module. The width of the glitch to absorb can be specified in terms of number of (half) bus clock cycles. A single glitch filter control register is provided as IICFLT. Effectively, any down-up-down or up-down-up transition on the data line that occurs within the number of clock cycles programmed here is ignored by the IIC. The programmer only needs to specify the size of glitch (in terms of bus clock cycles) for the filter to absorb and not pass.

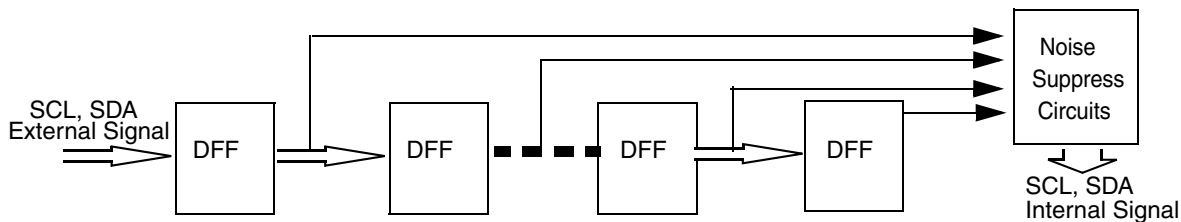


Figure 14-15. Programmable input glitch filter diagram

### 14.6.7 Address Matching Wakeup

When address matching happens as IIC works in slave receive mode, the MCU wakes from stop3 mode. After the address matching IAAS bit is set, an interrupt is sent out at the end of address matching to wake the MCU. The ACK will also be sent from slave if address is correctly matched. The IAAS bit must be cleared after the clock recovery.

#### NOTE

After the system was recovered to run mode IIC must restart if it is needed to work. The SCL line will not be hold low until the IIC resets after address matching.



## 14.6.8 DMA Support

DMA requests when DMAEN is enabled. DMA requests are generated by the transfer complete flag (TCF).

Control bit enables a DMA transfer rather than a CPU interrupt when the conditions listed in [Table 14-15](#) occur. When CPU interrupts are enabled and DMAEN is cleared, all the interrupt conditions generate CPU interrupt. If DMAEN is set, the only arbitration lost to another IIC module (error) and SCL low timeouts (error) then generates CPU interrupts. All other events initiate a DMA transfer signal.

### NOTE

- Before the last byte of master receive mode, the TXAK must be written to 1 to send NACK after the last byte's transfer. Therefore, the DMA needs to be disabled before the last byte's transfer.
- In 10-bit address mode transmission, the addresses has to be sent occupies 2-3 bytes. During this transfer period the DMA must be disabled because the IICC1 is written to send repeat start or transfer direction is changed.

## 14.7 Initialization/Application Information

### Module Initialization (Slave)

1. Write: IICC2
  - to enable or disable general call
  - to select 10-bit or 7-bit addressing mode
2. Write: IICA1
  - to set the slave address
3. Write: IICC1
  - to enable IIC and interrupts
4. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmit data
5. Initialize RAM variables used to achieve the routine shown in [Figure 14-16](#)

### Module Initialization (Master)

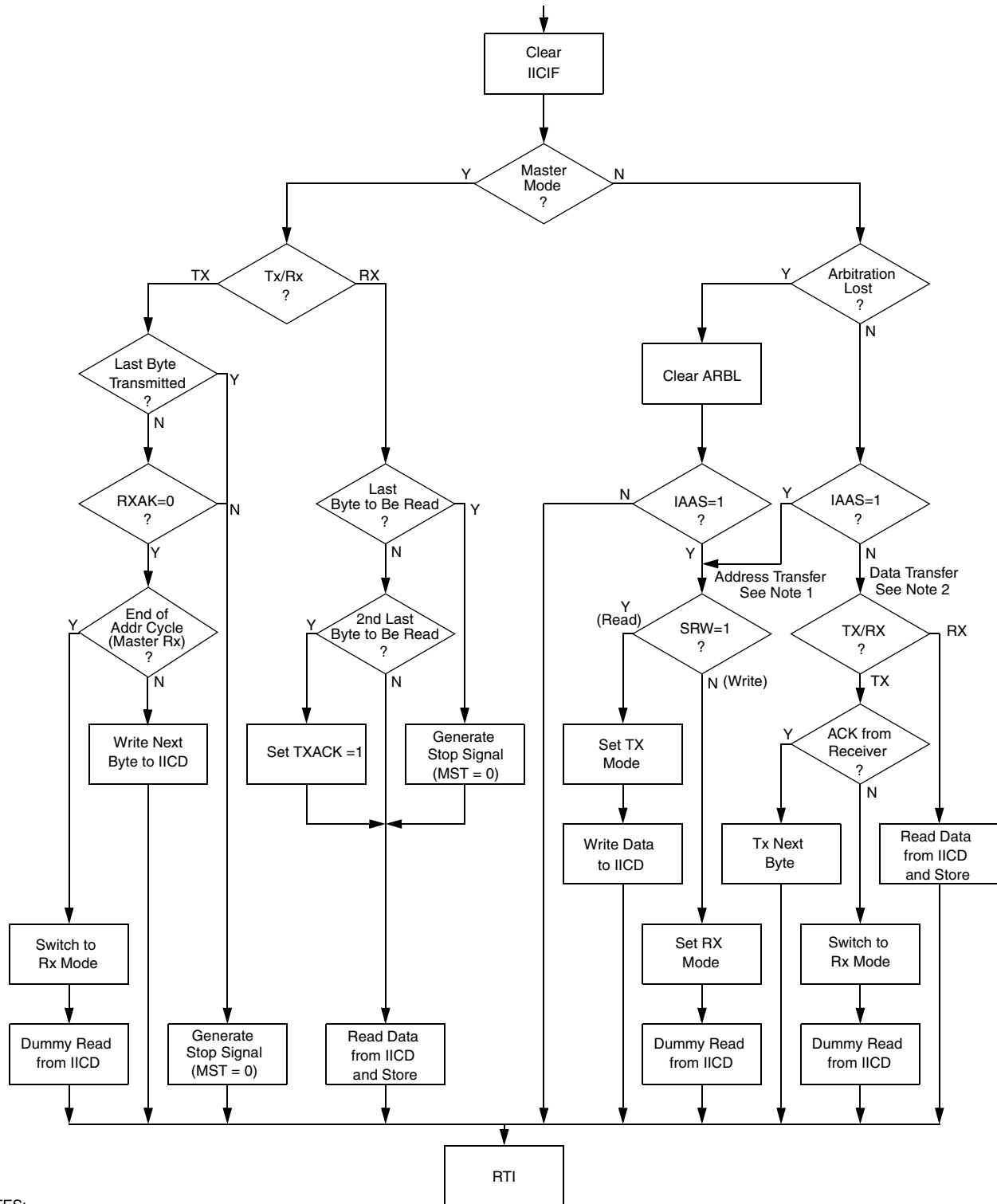
1. Write: IICF
  - to set the IIC baud rate (example provided in this chapter)
2. Write: IICC1
  - to enable IIC and interrupts
3. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmit data
4. Initialize RAM variables used to achieve the routine shown in [Figure 14-16](#)
5. Write: IICC1
  - to enable TX
6. Write: IICC1
  - to enable MST (master mode)
7. Write: IICD
  - with the address of the target slave. (The LSB of this byte determines whether the communication is master receive or transmit.)

### Module Use

The routine shown in [Figure 14-16](#) can handle both master and slave IIC operations. For slave operation, an incoming IIC message that contains the proper address begin IIC communication. For master operation, communication must be initiated by writing to the IICD register.

## Register Model

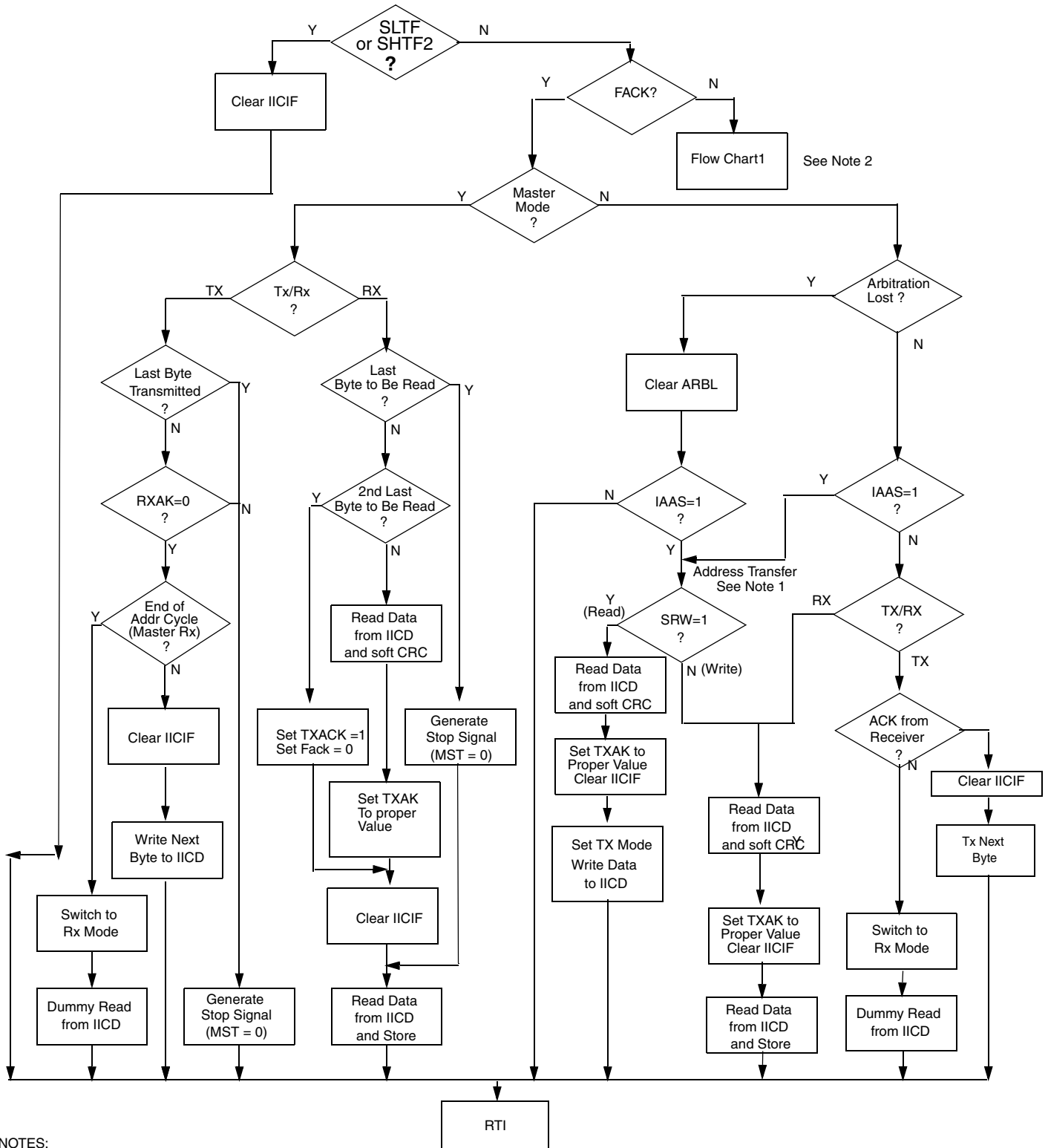
IICA1	AD[7:1]							0
	Address to which the module responds when addressed as a slave (in slave mode)							
IICF	MULT			ICR				
	Baud rate = BUSCLK / (2 x MULT x (SCL DIVIDER))							
IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	WUEN	DMAEN
	Module configuration							
IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
	Module status flags							
IICD	DATA							
	Data register; Write to transmit IIC data read to read IIC data							
IICC2	GCAEN	ADEXT	HCRS	SBRC	0	AD10	AD9	AD8
	Address configuration							
IICFLT	0	0	0	0	FLT3	FLT2	FLT1	FLT0
	IIC Programmable Input Glitch Filter							
IICSMB	FACK	ALERTEN	SIICAEN	TCKSEL	SLTF	SHTF1	SHTF2	SHTF2IE
	IIC SMBus Control and Status Register							
IICA2	SAD[7:1]							0
	IIC Address Register 2							
IICSLTH	SSLT[15:8]							
	IIC SCL Low Time Out Register High							
IICSLTL	SSLT[7:0]							
	IIC SCL Low Time Out Register Low							



NOTES:

1. If general call is enabled, a check must be done to determine whether the received address was a general call address (0x00). If the received address was a general call address, then the general call must be handled by user software.
2. When 10-bit addressing is used to address a slave, the slave sees an interrupt following the first byte of the extended address. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as a valid data transfer.

Figure 14-16. Typical IIC Interrupt Routine



NOTES:

1. If general call or SIICAEN is enabled, a check must be done to determine whether the received address was a general call address (0x00) or SMBus device default address. If the received address was one of them, then it must be handled by user software.
2. Flow chart1 means Figure 14-16. Typical IIC Interrupt Routine.
3. As receive side the second data reading should be done after 9th SCL cycle.

Figure 14-17. Typical IIC SMBus Interrupt Routine

## 14.8 SMBALERT#

Another optional signal is an interrupt line for devices that want to trade their ability to master for a pin. SMBALERT# is a wired-AND signal just as the SMBCLK and SMBDAT signals are. SMBALERT# is used in conjunction with the SMBus general call address. Messages invoked with the SMBus are 2 bytes long. (Now there is no ALERT# port in current block)

A slave-only device can signal the host through SMBALERT# that it wants to talk. The host processes the interrupt and simultaneously accesses all SMBALERT# devices through the alert response address (ARA). Only the devices that pulled SMBALERT# low acknowledge the alert response address.

The host performs a modified receive byte operation. The 7-bit device address provided by the slave transmit device is placed in the 7 most significant bits of the byte. The eighth bit can be a zero or one.

If more than one device pulls SMBALERT# low, the highest priority (lowest address) device win communication rights via standard arbitration during the slave address transfer.

After acknowledging the slave address, the device must disengage its SMBALERT# pulldown. If the host still sees SMBALERT# low when the message transfer is completed, it knows to read the ARA again. A host that does not implement the SMBALERT# signal may periodically access the ARA.

S	Alert Response Address	Rd	A	Device Address	A	P
---	---------------------------	----	---	----------------	---	---

**Table 14-16. A 7-bit-Addressable Device Responds to an ARA**

### NOTE

You must put device address on bus by software after response to the alert response address in current block.

# Chapter 15

## DMA Controller Module

### 15.1 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 programming model. 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.

#### 15.1.1 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 15-1](#), has four channels that allow byte, word, or longword data transfers. Each channel has a dedicated source address register (SAR $n$ ), destination address register (DAR $n$ ), status register (DSR $n$ ), byte count register (BCR $n$ ), and control register (DCR $n$ ). Collectively, the combined program-visible registers associated with each channel define a transfer control descriptor (TCD). All transfers are dual address, moving data from a source memory location to a destination memory location with the module operating as a 32-bit bus master connected to the system bus. The programming model is accessed through a 32-bit connection with the slave peripheral bus. DMA data transfers may be explicitly initiated by software or by peripheral hardware requests.

A simplified block diagram of the 4-channel DMA controller is shown in [Figure 15-1](#).

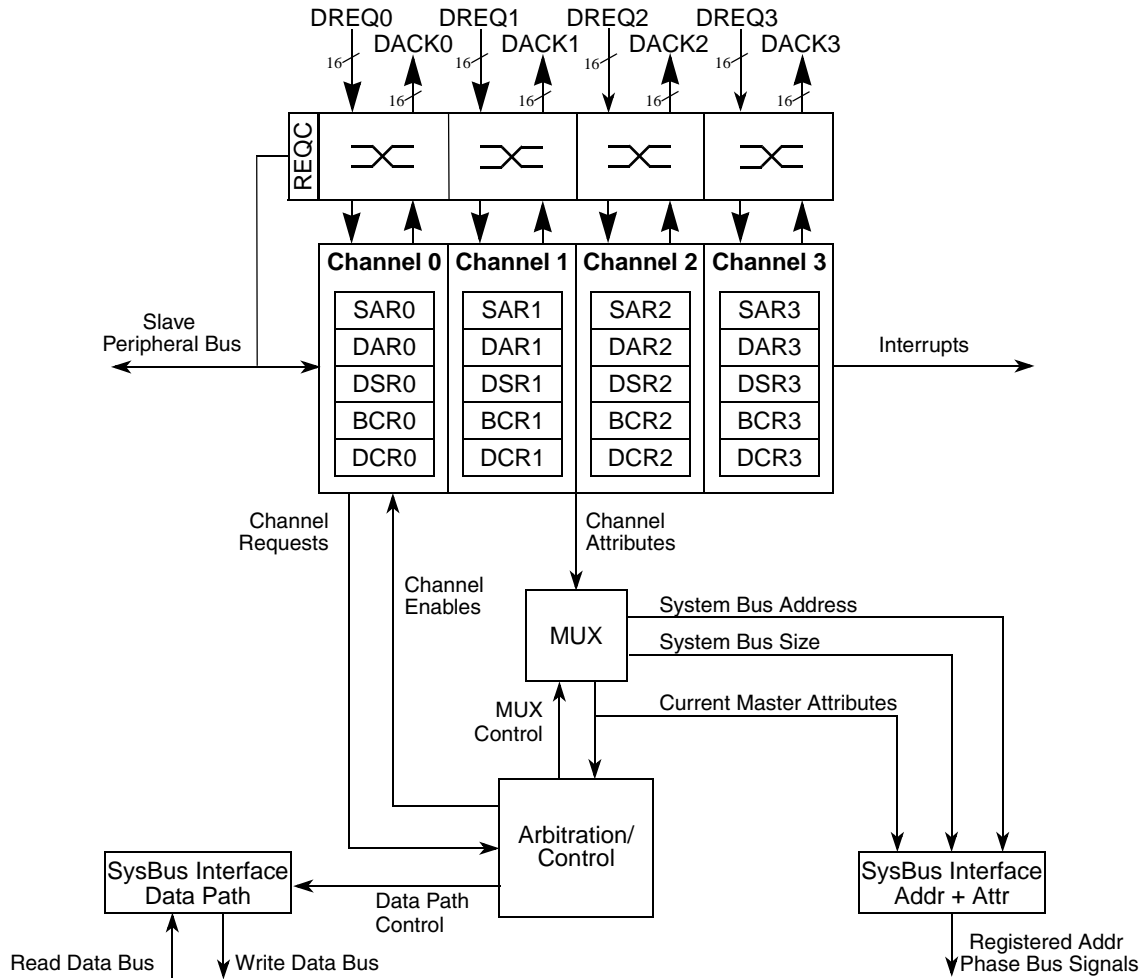


Figure 15-1. 4-Channel DMA Block Diagram

Throughout this chapter, the terms peripheral request or DREQ are used to refer to a DMA request from one of the on-chip peripherals or package pins. The DMA provides hardware handshake signals, either a DMA acknowledge (DACK) or a done indicator back to the peripheral. For details on the connections associated with DMA request inputs, see [Section 15.4.1, “DMA Request Control \(DMAREQC\).”](#)

### 15.1.2 Peripheral DMA request inputs

DMA enabled peripherals in MCF51AG128 include IIC, SCI, SPI, FTM, HSCMP, ADC, RTC, and eGPIO. These peripherals can output DMA transfer signal rather than a CPU interrupt if DMAEN bit is enabled, and the DMA transfer signals are connected to iEvent and DMA controller module. [Table 15-1](#) lists the DMA request sources assigned to each DMA channel.

DMA transfer signals from these peripherals can be operated as a source to DMA controller or as an iEvent input. Typically, make sure the single DMA transfer signal can not be configured as active DMA source and iEvent input at the same time.



Table 15-1. Peripheral DMA Request Inputs

Request ID	Assignment to DMA Channel 0	Assignment to DMA Channel 1	Assignment to DMA Channel 2	Assignment to DMA Channel 3
0	iEvent_ch0	iEvent_ch1	iEvent_ch2	iEvent_ch3
1	DMA_FTM1_ch0	DMA_FTM1_ch0	DMA_FTM2_ch0	DMA_FTM2_ch0
2	DMA_FTM1_ch1	DMA_FTM1_ch1	DMA_FTM2_ch1	DMA_FTM2_ch1
3	DMA_FTM1_ch2	DMA_FTM1_ch2	DMA_FTM2_ch2	DMA_FTM2_ch2
4	DMA_FTM1_ch3	DMA_FTM1_ch3	DMA_FTM2_ch3	DMA_FTM2_ch3
5	DMA_FTM1_ch4	DMA_FTM1_ch4	DMA_FTM2_ch4	DMA_FTM2_ch4
6	DMA_FTM1_ch5	DMA_FTM1_ch5	DMA_FTM2_ch5	DMA_FTM2_ch5
7	DMA_ADC	DMA_ADC	DMA_ADC	DMA_ADC
8	DMA_HSCMP1	DMA_SPI1_rx	DMA_HSCMP1	DMA_SPI2_rx
9	DMA_HSCMP2	DMA_SPI1_tx	DMA_HSCMP2	DMA_SPI2_tx
10	DMA_IIC	DMA_SCI1_rx	DMA_SPI2_rx	DMA_SCI2_rx
11	DMA_SPI1_rx	DMA_SCI1_tx	DMA_SPI2_tx	DMA_SCI2_tx
12	DMA_SPI1_tx	DMA_PTC	DMA_SCI2_rx	DMA_PTG
13	DMA_SCI1_rx	DMA_PTD	DMA_SCI2_tx	DMA_PTH
14	DMA_SCI1_tx	DMA_RTC	DMA_PTE	DMA_PTJ
15	DMA_PTA	DMA_PTB	DMA_PTF	DMA_IIC

## 15.2 Low Power Mode Operation

The DMA controller is capable of functioning in run and wait modes of operation, and does not differentiate between run and wait modes. For details on low-power mode operation, refer to [Table 3-1](#) in [Chapter 3, “Modes of Operation”](#).

### 15.2.1 DMA Clock Gating

The bus clock to the DMA controller can be gated on and off using the SCGC1[DMA] bits (see [Section 5.8.9, “System Clock Gating Control 1 Register \(SCGC1\)”](#)). This bit is set after any reset that enables the bus clock to this module. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

### 15.2.2 Features

The DMA controller module features:

- Four independently programmable DMA controller channels
- Dual-address transfers via 32-bit master connection to the system bus
- Data transfers in 8-, 16-, or 32-bit blocks
- Continuous-mode or cycle-steal transfers from software or peripheral initiation
- One programmable input selected from 16 possible peripheral requests per channel

- Automatic hardware acknowledge/done indicator from each channel
- Independent source and destination address registers
- Optional modulo addressing and automatic updates of source and destination addresses
- Independent transfer sizes for source and destination
- Optional auto-alignment feature for source or destination accesses
- Optional automatic single or double channel linking
- Programming model accessed via 32-bit slave peripheral bus
- Channel arbitration on transfer boundaries using fixed priority scheme

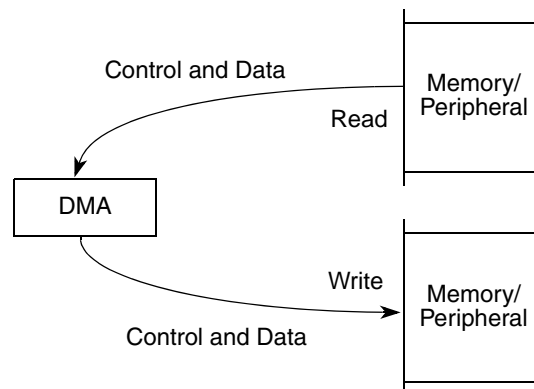
### 15.3 DMA Transfer Overview

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 address all four channels at once.

The processor generates DMA requests by setting DCR[START]. Each channel can be programmed to select one peripheral request from a set of 16 possible request inputs. The channels support cycle-steal and continuous transfer modes; see [Section 15.5.1, “Transfer Requests \(Cycle-Steal and Continuous Modes\).”](#)

The DMA controller supports dual-address transfers using its bus master connection to the system bus. The DMA channels support transfers up to 32 data bits in size and have the same memory map addressability as the processor.

- Dual-address transfers—A dual-address transfer consists of a read followed by a write and is initiated by a request using the DCR $n$ [START] bit or by a peripheral DMA request. The read data is temporarily held in the DMA channel hardware until the write operation. Two types of single transfers occur: a read from a source address followed by a write to a destination address. See [Figure 15-2](#).



**Figure 15-2. Dual-Address Transfer**

Any operation involving a DMA channel follows the same three steps:

1. Channel initialization—The transfer control descriptor, contained in the channel registers, is loaded with address pointers, a byte-transfer count, and control information using accesses from the slave peripheral bus.
2. Data transfer—The DMA accepts requests for data transfers. Upon receipt of a request, it provides address and bus control for the transfers via its master connection to the system bus and temporary storage for the read data. The channel performs one or more source read and destination write data 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 15.4.2.3, “DMA Status Registers \(DSRn\) and Byte Count Registers \(BCRn\).”](#)

## 15.4 Memory Map/Register Definition

This section describes each register and its bit assignments. Modifying DMA control registers during a transfer can result in undefined operation. [Table 15-2](#) shows the mapping of DMA controller registers. The DMA programming model is accessed via the slave peripheral bus. The concatenation of the source and destination address registers, the status and byte count register, and the control register creates a 128-bit transfer control descriptor (TCD) that defines the operation of each DMA channel. The programming model for the DMA controller is based at address 0x(FF)FF\_E400.

**Table 15-2. DMA Controller Memory Map**

Memory Address	Register	Width	Access	Reset Value	Section/Page
0x(FF)FF_E400	DMA request control register (DMAREQC)	32	R/W	0x0000_0000	<a href="#">15.4.1/15-5</a>
0x(FF)FF_E500 + $n * 0x10$	Source address register $n$ (SAR $n$ ) where $n = 0-3$	32	R/W	0x0000_0000	<a href="#">15.4.2.1/15-7</a>
0x(FF)FF_E504 + $n * 0x10$	Destination address register $n$ (DAR $n$ ) where $n = 0-3$	32	R/W	0x0000_0000	<a href="#">15.4.2.2/15-8</a>
0x(FF)FF_E508 + $n * 0x10$	DMA status (DSR $n$ ) and byte count register $n$ (BCR $n$ ) where $n = 0-3$	32	R/W	0x0000_0000	<a href="#">15.4.2.3/15-8</a>
0x(FF)FF_E50C + $n * 0x10$	DMA control register $n$ (DCR $n$ ) where $n = 0-3$	32	R/W	0x0000_0000	<a href="#">15.4.2.4/15-10</a>

### 15.4.1 DMA Request Control (DMAREQC)

The DMAREQC register provides a software-controlled connection matrix for DMA requests and acknowledges. Each channel supports 16 possible peripheral requests. The DMAREQC is programmed to select one peripheral request from the available sources for each channel of the DMA controller. Additionally, the DMAREQC routes DMA acknowledge from the channel back to the appropriate peripheral. Writing to this register determines the exact routing of the DMA requests to each of the four channels of the DMA module.

If DCR $n$ [ERQ] is set and the channel is idle, the assertion of the appropriate DREQ $n$  signal activates channel  $n$ .

The connections of the DMA request sources to the specific channels are device-specific.

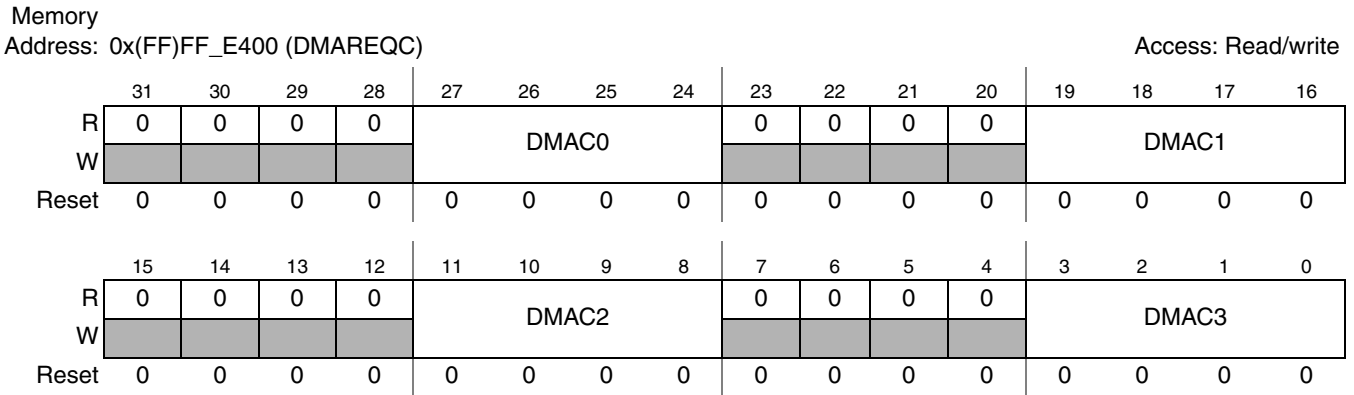


Figure 15-3. DMA Request Control Register (DMAREQC)

Table 15-3. DMAREQC Field Description

Field	Description
31–28, 23–20, 15–12, 7–4	Reserved, must be cleared.
27–24, 19–16, 11–18, 3–0 DMAC $n$	<p>DMA channel <math>n</math>. Each four bit field defines the logical connection between the DMA requesters and that DMA channel. There are sixteen possible requesters per channel and any request from the possible sources can be routed to the specified DMA channel. Effectively, the DMAREQC provides a software-controlled routing matrix of the DMA request signals to the 4 channels of the DMA module. DMAC0 controls DMA channel 0, DMAC1 controls DMA channel 1, etc.</p> <p>The DMA also uses this register to control the broadcasting of acknowledge/done signals back to the selected peripheral to complete the hardware-initiated data transfer.</p> <p><i>The definition of the 16 possible DMA request sources for each channel is device specific.</i></p> <p>0000 Select request 0 as the source                      0001 Select request 1 as the source                      0010 Select request 2 as the source                      ...                      1110 Select request 14 as the source                      1111 Select request 15 as the source</p>

### 15.4.2 Transfer Control Descriptor (TCD $n$ )

Each channel requires a 16-byte transfer control descriptor for defining the desired data movement operation. The TCD structure is implemented in program-visible registers in each channel. The channel descriptor registers are mapped in the module’s programming model in sequential order: channel 0, channel 1, channel 2, and channel 3. The definitions of the TCD are presented as four 32-bit values.

The structure of the transfer control descriptor is fundamental to the operation of the DMA module. Figure 15-4 defines it in a ‘C’ pseudo-code specification where `int` refers to a 32-bit variable.

Figure 15-4. TCD Software Structure

```

typedef struct {
    unsigned int sar; /* source address */
    unsigned int dar; /* destination address */

    unsigned int dsr_rfu_1:1; /* reserved */
    unsigned int ce:1; /* configuration error */
    unsigned int bes:1; /* bus error on source read */
    unsigned int bed:1; /* bus error on destination write */
    unsigned int dsr_rfu_2:1; /* reserved */
    unsigned int req:1; /* request pending */
    unsigned int bsy:1; /* channel busy */
    unsigned int done:1; /* transactions done */
    unsigned int bcr:24; /* byte count register */

    unsigned int eint:1; /* enable interrupt on completion */
    unsigned int erq:1; /* enable peripheral request */
    unsigned int cs:1; /* cycle steal */
    unsigned int aa:1; /* auto-align */
    unsigned int dcr_1:5; /* reserved */
    unsigned int sinc:1; /* source increment */
    unsigned int ssize:2; /* source size */
    unsigned int dinc:1; /* destination increment */
    unsigned int dsize:2; /* destination size */
    unsigned int start:1; /* explicit channel start */
    unsigned int smod:4; /* source address modulo */
    unsigned int dmod:4; /* destination address modulo */
    unsigned int d_req:1; /* disable request */
    unsigned int dcr_2:1; /* reserved */
    unsigned int linkcc:2; /* link channel control */
    unsigned int lch1:2; /* link channel 1 */
    unsigned int lch2:2; /* link channel 2 */
} tcd /* transfer_control_descriptor */

```

### 15.4.2.1 Source Address Registers (SAR<sub>n</sub>)

SAR<sub>n</sub>, shown in [Figure 15-5](#), contains the byte address used by the DMA controller to read data. The SAR<sub>n</sub> is typically aligned on a 0-modulo-ssize boundary, that is, on the natural alignment of the source data. Recall the system only supports 24-bit addresses, so SAR<sub>n</sub>[31:24] is ignored.

Memory 0x(FF)FF\_E500 (SAR0) Access: Read/write  
 Address: 0x(FF)FF\_E510 (SAR1)  
 0x(FF)FF\_E520 (SAR2)  
 0x(FF)FF\_E530 (SAR3)

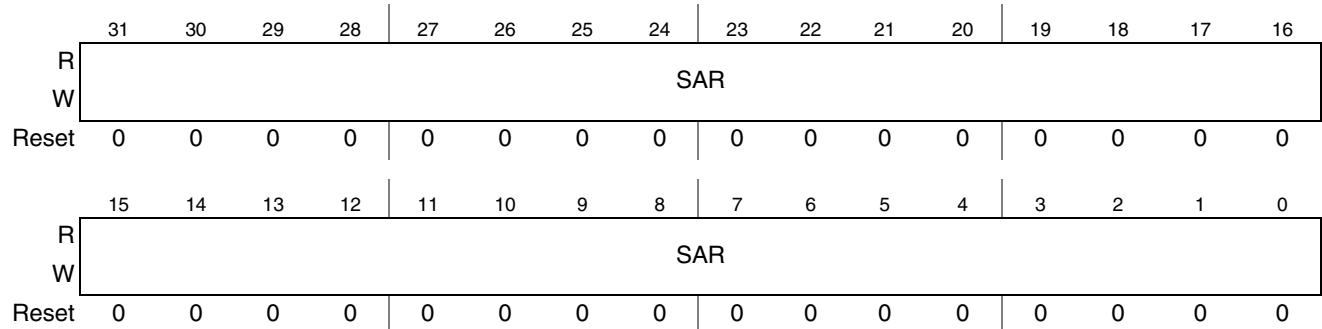


Figure 15-5. Source Address Registers (SAR $n$ )

### 15.4.2.2 Destination Address Registers (DAR $n$ )

As shown in Figure 15-6, DAR $n$  contains the byte address used by the DMA controller to write data. The DAR $n$  is typically aligned on a 0-modulo-dsize boundary, that is, on the natural alignment of the destination data. Recall the system only supports 24-bit addresses, so DAR $n$ [31:24] is ignored

Memory 0x(FF)FF\_E504 (DAR0) Access: Read/write  
 Address: 0x(FF)FF\_E514 (DAR1)  
 0x(FF)FF\_E524 (DAR2)  
 0x(FF)FF\_E534 (DAR3)

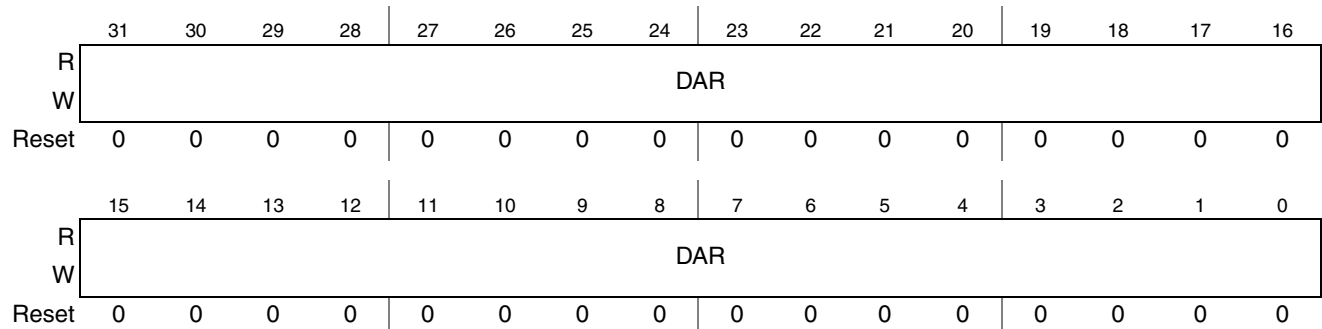


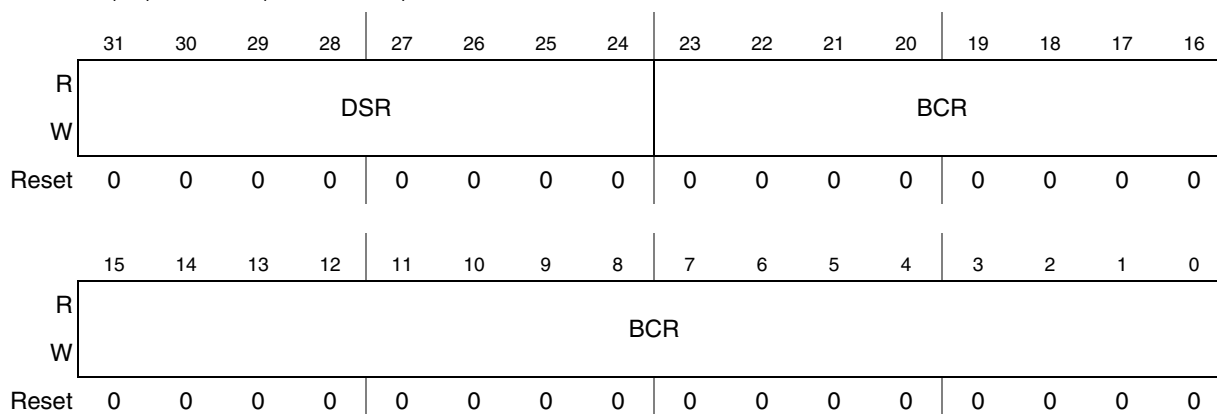
Figure 15-6. Destination Address Registers (DAR $n$ )

### 15.4.2.3 DMA Status Registers (DSR $n$ ) and Byte Count Registers (BCR $n$ )

The DSR $n$  and BCR $n$  registers are two logical registers that occupy one 32-bit address as shown in Figure 15-7. The longword address used to access both registers is the same; DSR $n$  occupies bits 31–24, and BCR $n$  occupies bits 23–0. The DSR $n$  contains flags indicating the channel status and the BCR $n$  contains the number of bytes yet to be transferred for a given block.

BCR $n$  decrements by 1, 2, 4 for byte, word, or longword accesses, respectively, on the successful completion of the write transfer. The BCR $n$  is cleared if a 1 is written to DSR[DONE].

Memory 0x(FF)FF\_E508 (DSR0/BCR0) Access: Read/write  
 Address 0x(FF)FF\_E518 (DSR1/BCR1)  
 : 0x(FF)FF\_E528 (DSR2/BCR2)  
 0x(FF)FF\_E538 (DSR3/BCR3)

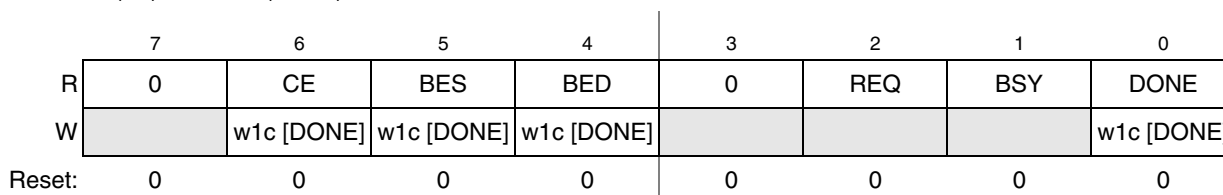


**Figure 15-7. Byte Count Registers (BCR<sub>n</sub>) and DMA Status Registers (DSR<sub>n</sub>)**

The fields of the DSR<sub>n</sub> register (bits 31–24 in Figure 15-7) are shown in Figure 15-8. In response to an event, the DMA controller writes to the appropriate DSR<sub>n</sub> bit. Only a write to DSR<sub>n</sub>[DONE] results in action. DSR<sub>n</sub>[DONE] is set when the block transfer is complete.

When a transfer sequence is initiated and BCR<sub>n</sub>[BCR] is not a multiple of 4 or 2 when the DMA is configured for longword or word transfers, respectively, DSR<sub>n</sub>[CE] is set and no transfer occurs.

Memory 0x(FF)FF\_E508 (DSR0) Access: Read/write  
 Address: 0x(FF)FF\_E518 (DSR1)  
 0x(FF)FF\_E528 (DSR2)  
 0x(FF)FF\_E538 (DSR3)



**Figure 15-8. DMA Status Registers (DSR<sub>n</sub>)**

**Table 15-4. DSR<sub>n</sub> Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6 CE	Configuration error. Occurs when BCR, SAR, or DAR does not match the requested transfer size, or if SSIZE, DSIZE is set to an unsupported value, 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.

**Table 15-4. DSR<sub>n</sub> Field Descriptions (continued)**

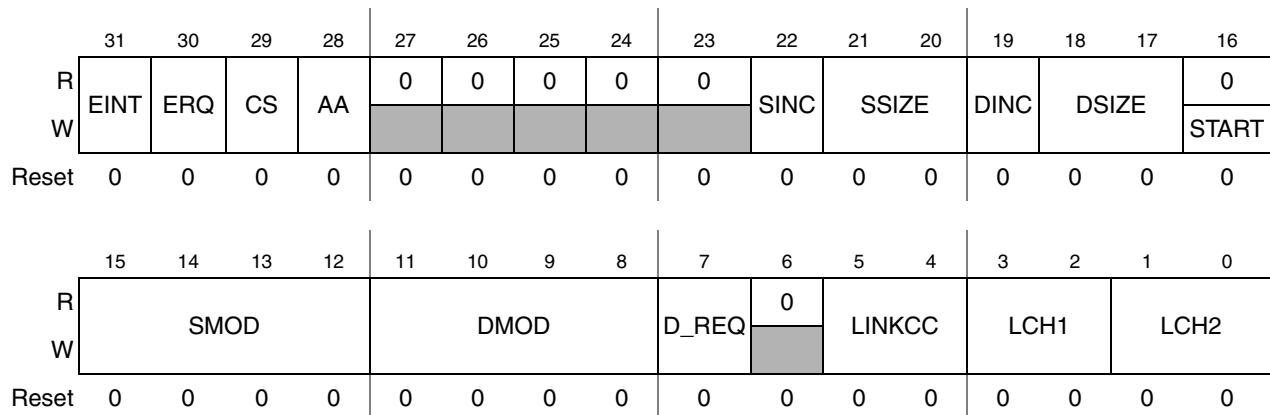
Field	Description
4 BED	Bus error on destination 0 No bus error occurred. 1 The DMA channel terminated with a bus error during the write portion of a transfer.
3	Reserved, must be cleared.
2 REQ	Request 0 No request is pending or the channel is currently active. Cleared when the channel is selected. 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 based on 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 DMA transfer is not yet complete. Writing a 0 has no effect. 1 DMA transfer completed. Writing a 1 to this bit clears all DMA status bits and should be used in an interrupt service routine to clear the DMA interrupt and error bits.

### 15.4.2.4 DMA Control Registers (DCR<sub>n</sub>)

The DMA control registers (DCR<sub>n</sub>) are described in [Figure 15-9](#) and [Table 15-5](#).

Memory 0x(FF)FF\_E50C (DCR0)  
Address 0x(FF)FF\_E51C (DCR1)  
: 0x(FF)FF\_E52C (DCR2)  
0x(FF)FF\_E53C (DCR3)

Access: Read/write



**Figure 15-9. DMA Control Registers (DCR<sub>n</sub>)**



Table 15-5. DCR<sub>n</sub> Field Descriptions

Field	Description
31 EINT	Enable 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 Interrupt signal is enabled.
30 ERQ	Enable peripheral request. Care should be taken because a collision can occur between the START bit and DREQ <sub>n</sub> when ERQ equals 1. 0 Peripheral request is ignored. 1 Enables peripheral request, defined by the appropriate DMQREQC[DMAC <sub>n</sub> ] field, to initiate transfer. A software-initiated request (setting DMACR <sub>n</sub> [START]) 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 bits determine whether the source or destination is auto-aligned, that is, transfers are optimized based on the address and size. See <a href="#">Section 15.5.4, “Advanced Data Transfer Controls: Auto-Alignment.”</a> 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.
27–25	Reserved, must be cleared.
24–23	Reserved, must be written as zero otherwise undefined behavior results.
22 SINC	Source increment. Controls whether the source address increments after each successful transfer. 0 No change to SAR after a successful transfer. 1 The SAR increments by 1, 2, 4 as determined by the transfer size.
21–20 SSIZE	Source size. Determines the data size of the source bus cycle for the DMA controller. 00 Longword 01 Byte 10 Word 11 Reserved (generates a configuration error (DSR <sub>n</sub> [CE]) if incorrectly specified at time of channel activation)
19 DINC	Destination increment. Controls whether the 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 depending upon the size of the transfer.
18–17 DSIZE	Destination size. Determines the data size of the destination bus cycle for the DMA controller. 00 Longword 01 Byte 10 Word 11 Reserved (generates a configuration error (DSR <sub>n</sub> [CE]) if incorrectly specified at time of channel activation)
16 START	Start transfer. 0 DMA inactive 1 The DMA begins the transfer in accordance to the values in the TCD <sub>n</sub> . START is cleared automatically after one module clock and always reads as logic 0.

Table 15-5. DCR<sub>n</sub> Field Descriptions (continued)

Field	Description												
15–12 SMOD	<p>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.</p> <table border="1"> <thead> <tr> <th>SMOD</th> <th>Circular Buffer Size</th> </tr> </thead> <tbody> <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> </tbody> </table>	SMOD	Circular Buffer Size	0000	Buffer Disabled	0001	16 Bytes	0010	32 Bytes	...	...	1111	256 Kbytes
SMOD	Circular Buffer Size												
0000	Buffer Disabled												
0001	16 Bytes												
0010	32 Bytes												
...	...												
1111	256 Kbytes												
11–8 DMOD	<p>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.</p> <table border="1"> <thead> <tr> <th>DMOD</th> <th>Circular Buffer Size</th> </tr> </thead> <tbody> <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> </tbody> </table>	DMOD	Circular Buffer Size	0000	Buffer Disabled	0001	16 Bytes	0010	32 Bytes	...	...	1111	256 Kbytes
DMOD	Circular Buffer Size												
0000	Buffer Disabled												
0001	16 Bytes												
0010	32 Bytes												
...	...												
1111	256 Kbytes												
7 D_REQ	<p>Disable request. DMA hardware automatically clears the corresponding DCR<sub>n</sub>[ERQ] bit when the byte count register reaches zero.</p> <p>0 ERQ bit is not affected. 1 ERQ bit is cleared when the BCR is exhausted.</p>												
6	Reserved; must be cleared.												

Table 15-5. DCR<sub>n</sub> Field Descriptions (continued)

Field	Description
5-4 LINKCC	<p>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.</p> <p>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</p> <p>If not in cycle steal mode (DCR<sub>n</sub>[CS]=0) and LINKCC equals 01 or 10, no link to LCH1 occurs.</p> <p>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.</p>
3-2 LCH1	<p>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<sub>n</sub>[CE] is set).</p> <p>00 DMA Channel 0  01 DMA Channel 1  10 DMA Channel 2  11 DMA Channel 3</p>
1-0 LCH2	<p>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<sub>n</sub>[CE] is set).</p> <p>00 DMA Channel 0  01 DMA Channel 1  10 DMA Channel 2  11 DMA Channel 3</p>

## 15.5 Functional Description

In the following discussion, the term DMA request implies that DCR<sub>n</sub>[START] is set, or DCR<sub>n</sub>[ERQ] is set and then followed by assertion of the properly selected DMA peripheral request. The START bit is cleared when the channel is activated.

Before initiating a dual-address access, the DMA module verifies that DCR<sub>n</sub>[SSIZE] and DCR<sub>n</sub>[DSIZE] are consistent with the source and destination addresses. If they are not consistent, the configuration error bit, DSR<sub>n</sub>[CE], is set. If misalignment is detected, no transfer occurs, DSR<sub>n</sub>[CE] is set, and, depending on the DCR configuration, an interrupt event may be issued. If the auto-align bit, DCR<sub>n</sub>[AA], is set, error checking is performed on the appropriate registers.

A read/write transfer sequence reads data from the source address and writes it to the destination address. The number of bytes transferred is the largest of the sizes specified by DCR<sub>n</sub>[SSIZE] and DCR<sub>n</sub>[DSIZE]. See 15.4.2.4, “DMA Control Registers (DCR<sub>n</sub>).”

Source and destination address registers (SAR<sub>n</sub> and DAR<sub>n</sub>) can be programmed in the DCR<sub>n</sub> to increment at the completion of a successful transfer.

## 15.5.1 Transfer Requests (Cycle-Steal and Continuous Modes)

The DMA channel supports software-initiated or peripheral-initiated requests. A request is issued by setting  $DCRn[START]$  or when the selected peripheral request asserts and  $DCRn[ERQ]$  is set. Setting  $DCRn[ERQ]$  enables recognition of the peripheral DMA requests. Selecting between cycle-steal and continuous modes minimizes bus usage for either type of request.

- Cycle-steal mode ( $DCRn[CS] = 1$ )—Only one complete transfer from source to destination occurs for each request. If  $DCRn[ERQ]$  is set, the request is peripheral initiated. A software-initiated request is enabled by setting  $DCRn[START]$ .
- Continuous mode ( $DCRn[CS] = 0$ )—After a software-initiated or peripheral request, the DMA continuously transfers data until  $BCRn$  reaches zero. The DMA performs the specified number of transfers, then retires the channel.

In either mode, the platform's crossbar switch performs independent arbitration on each slave port after each transaction.

## 15.5.2 Channel Initialization and Startup

Before a data transfer starts, the channel's transfer control descriptor must be initialized with information describing configuration, request-generation method, and pointers to the data to be moved.

### 15.5.2.1 Channel Prioritization

The four DMA channels are prioritized based on number, with channel 0 having highest priority and channel 3 having the lowest, that is, channel 0 > channel 1 > channel 2 > channel 3.

Simultaneous peripheral requests activate the channels based on this priority order. Once activated, a channel runs to completion as defined by  $DCRn[CS]$  and  $BCRn$ .

### 15.5.2.2 Programming the DMA Controller Module

There is no mechanism within the DMA module itself to prevent writes to programming model registers during DMA accesses and corrupting the data transfer.

General guidelines for programming the DMA are:

- The  $DMAREQC$  register is configured to select the peripheral DMA requests and assign them to the individual DMA channels.
- Next, the  $TCDn$  is initialized.
- The  $SARn$  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 appropriately-aligned address.
- The  $DARn$  is initialized with the destination (write) address. If the transfer is from a peripheral device to memory, or from memory to memory, the  $DARn$  is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device,  $DARn$  is loaded

with the address of the peripheral data register. This address can be any appropriately-aligned address.

- $SAR_n$  and  $DAR_n$  change after each data transfer depending on  $DCR_n[SSIZE, DSIZE, SINC, DINC, SMOD, DMOD]$  and the starting addresses. Increment values can be 1, 2, 4 for byte, word, or longword transfers, respectively. If the address register is programmed to remain unchanged, the register is not incremented after the data transfer.
- $BCR_n[BCR]$  must be loaded with the total number of bytes to be transferred. It is decremented by 1, 2, or 4 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 may be started by setting  $DCR_n[START]$ , or a properly-selected peripheral DMA request, depending on the status of  $DCR_n[ERQ]$ . For a software-initiated transfer, the channel can be started by setting  $DCR_n[START]$  as part of a single 32-bit write to the last longword of the  $TCD_n$ , that is, it is not required to write the  $DCR_n$  with  $START$  cleared and then perform a second write to explicitly set  $START$ .
- Programming the channel for a software-initiated request causes the channel to request the system bus and start transferring data immediately. If the channel is programmed for peripheral-initiated request, a properly-selected peripheral DMA request must be asserted before the channel begins the system bus transfers.
- The hardware can automatically clear  $DCR_n[ERQ]$ , disabling the peripheral request, when  $BCR_n$  reaches zero by setting  $DCR_n[D\_REQ]$ .
- 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.

### 15.5.3 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 after 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 system 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 system address bus. When the appropriate number of write cycles complete (multiple writes if the source size is larger than the destination),  $DAR_n$  increments by the appropriate number of bytes if  $DCR_n[DINC]$  is set.  $BCR_n$  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 continuous mode is enabled ( $DCR_n[CS] = 0$ ).  
If a termination error occurs,  $DSR_n[BED, DONE]$  are set and DMA transactions stop.

## 15.5.4 Advanced Data Transfer Controls: Auto-Alignment

This section describes auto-alignment for DMA transfers. Typically, this DMA feature is applicable for transfers of large blocks of data, and therefore is not applicable for peripheral-initiated cycle-steal transfers.

Auto-alignment allows block transfers to occur at the optimal size based on the address, byte count, and programmed size. To use this feature,  $DCRn[AA]$  must be set. The source is auto-aligned if  $DCRn[SSIZE]$  indicates a transfer size larger than  $DCRn[DSIZE]$ . 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  $BCRn$  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  $BCRn$  is less than 16 at the start of a transfer, the number of bytes remaining dictates transfer size. For example,  $AA$  equals 1,  $SARn$  equals  $0x(00)80\_0001$ ,  $BCRn$  equals  $0x00\_00F0$ ,  $SSIZE$  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  $0x(00)80\_0001$ , increment  $SARn$ , write 1 byte (using  $DARn$ ).
2. Read word from  $0x(00)80\_0002$ , increment  $SARn$ , write 2 bytes.
3. Read longword from  $0x(00)80\_0004$ , increment  $SARn$ , write 4 bytes.
4. Repeat longwords until  $SARn = 0x(00)80\_00F0$ .
5. Read byte from  $0x(00)80\_00F0$ , increment  $SARn$ , write byte.

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.

## 15.5.5 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,  $DSRn[BES]$  is set for a read and  $DSRn[BED]$  is set for a write before the transfer is halted. If the error occurred in a write cycle, data in the internal holding registers is lost.
- Interrupts—If  $DCRn[EINT]$  is set, the DMA drives the appropriate interrupt request signal. The processor can read  $DSRn$  to determine whether the transfer terminated successfully or with an error.  $DSRn[DONE]$  is then written with a one to clear the interrupt, the DONE, and error status bits.

# Chapter 16

## Intelligent Event Controller (iEvent)

### 16.1 Introduction

The Intelligent Event Controller (iEvent) supports the generation of highly programmable combinational event signaling where the outputs can be used as interrupt requests, DMA requests, or hardware triggers.

The iEvent controller is a slave peripheral module connecting event input indicators from a variety of device modules and generating event output signals that can be routed to the interrupt controller, DMA module, or other peripheral modules. Its programming model is accessed through the standard IPS slave interface. The module is designed to be very configurable in terms of the functionality supported by the programming model and the design description for creating the minimum sized module based on the device requirements. It features a basic single-channel event logic block that can be replicated to create the required number of event outputs for a given device.

The initial application of the iEvent module is targeted for devices based on the Version 1 ColdFire core with DMA capabilities, but the module's functionality is applicable virtually to any embedded microcontroller or microprocessor requiring the ability to allow you to create programmable combinational event triggers.

#### 16.1.1 Overview

The iEvent module in MCF51AG128 supports 4 event outputs, where each event output represents a user-programmed combinational boolean function based on 4 event input indicators selected from 16 possible sources. The key features of this module include:

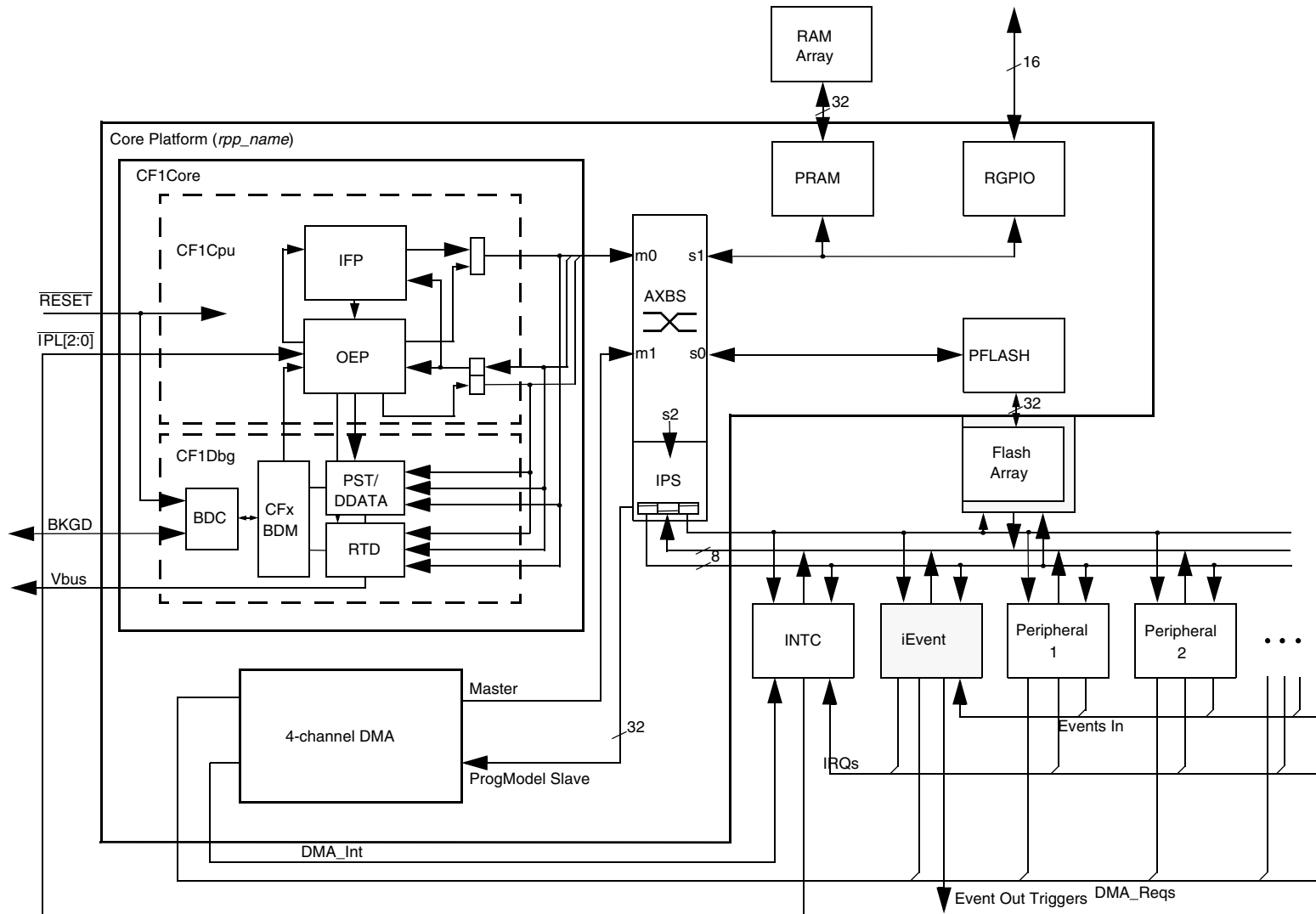
- Selection of 4 event inputs from 16 possible sources for each event output
- User-programmable combinational boolean function evaluation using the selected 4 event inputs
- Programmable configuration for use as an interrupt request (IRQ), a DMA request, or a hardware trigger
- Programmable support of event output as one-shot or reoccurring modes of operation
- Programmable support for automatic broadcast of done indicator to the event input modules involved in the generation of the request
- Memory-mapped device connected to an 8-bit slave peripheral (IPS) bus
  - Support for all access sizes: byte, word, and longword
  - All accesses complete with a one wait-state response to allow extensive explicit clock gating to minimize power dissipation
- Configurable design description supports 4 event output channels

### NOTE

A single event output can be programmed to operate as an interrupt request, a DMA request, or a hardware trigger. In many devices, a single iEvent output is wired to multiple destination modules. Since the iEvent output can only be configured for one function at a time, software must disable all other functions. For example, consider a single iEvent output connected to the interrupt controller, the DMA controller and as a hardware trigger to another module. If the output is being used as a hardware trigger, then software must configure the appropriate inputs to the interrupt and DMA controllers to be logically disabled for correct system operation.



The relative location of the iEvent module (highlighted in the diagram) within a typical reduced product platform with Version 1 ColdFire core and DMA is shown in Figure 16-1. The crossbar switch (AXBS) for this platform supports 2 bus masters (core, DMA) and 3 slaves (flash, RAM/RGPIO, IPS), while the DMA includes a bus master port and an IPS slave connection for its programming model. The iEvent's connections with the IPS bus, the event inputs, and the output IRQs, DMA requests and triggers are clearly shown.



**Figure 16-1. V1 ColdFire Core Platform with DMA + iEvent Block Diagram**

A simplified block diagram of the iEvent module is shown in Figure 16-2.

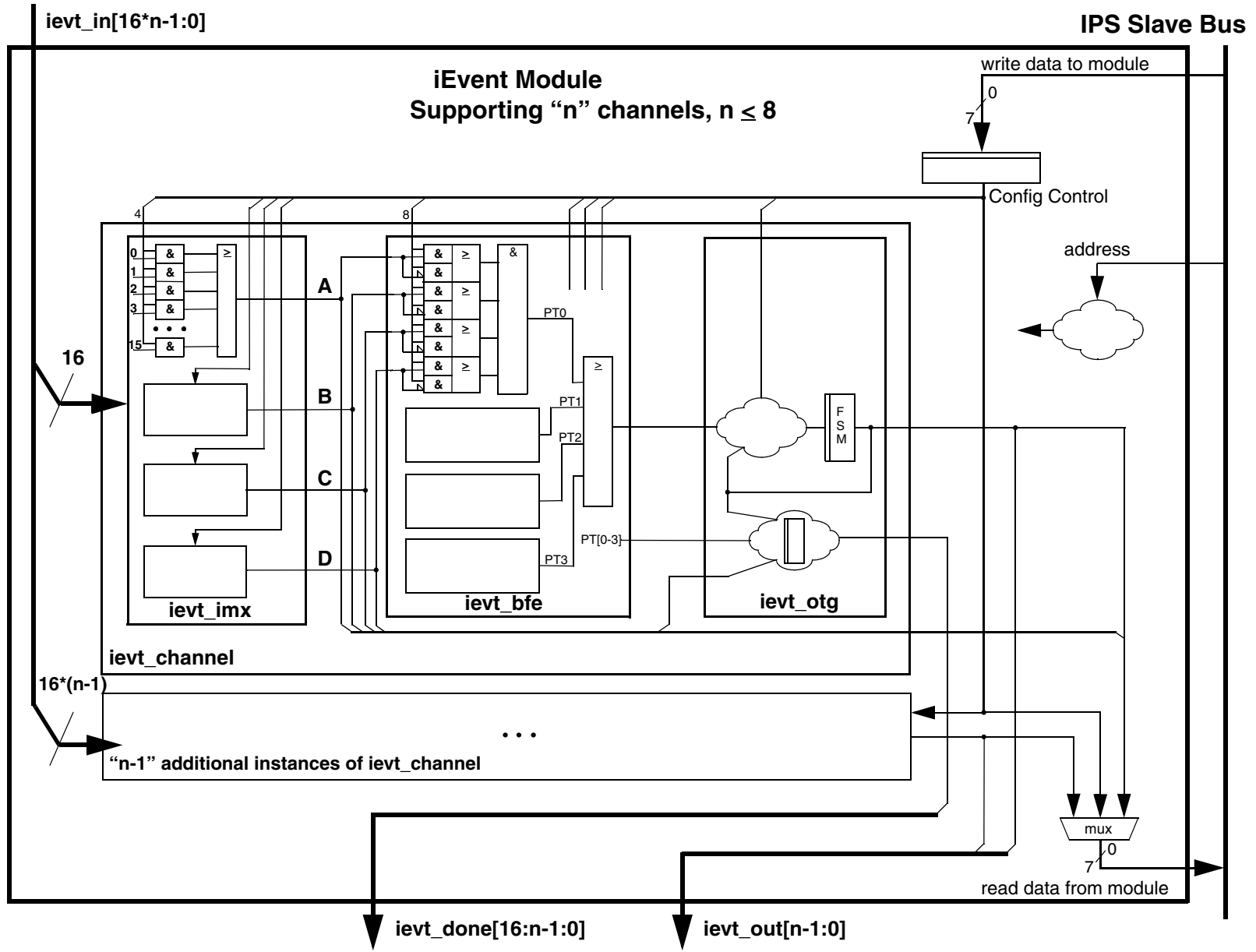


Figure 16-2. Simplified iEvent Block Diagram

## 16.1.2 iEvent Input Resources Distribution

Table shows how the DMA request sources from DMA enabled peripherals are connected to the iEvent channels (0–3).

**Table 16-1. iEvent Input Resources Distribution**

Request ID	Assignment to iEvent Channel 0	Assignment to iEvent Channel 1	Assignment to iEvent Channel 2	Assignment to iEvent Channel 3
0	DMA_PTA	DMA_PTA	DMA_PTE	DMA_PTE
1	DMA_PT B	DMA_PT B	DMA_PTF	DMA_PTF
2	DMA_PTC	DMA_PTC	DMA_PT G	DMA_PT G
3	DMA_PTD	DMA_PTD	DMA_HSCMP2	DMA_RTC
4	DMA_FTM1_ch0	DMA_FTM1_ch0	DMA_FTM2_ch0	DMA_FTM2_ch0
5	DMA_FTM1_ch1	DMA_FTM1_ch1	DMA_FTM2_ch1	DMA_FTM2_ch1
6	DMA_FTM1_ch2	DMA_FTM1_ch2	DMA_FTM2_ch2	DMA_FTM2_ch2
7	DMA_FTM1_ch3	DMA_FTM1_ch3	DMA_FTM2_ch3	DMA_FTM2_ch3
8	DMA_FTM1_ch4	DMA_FTM1_ch4	DMA_FTM2_ch4	DMA_FTM2_ch4
9	DMA_FTM1_ch5	DMA_FTM1_ch5	DMA_FTM2_ch5	DMA_FTM2_ch5
10	DMA_ADC	DMA_ADC	DMA_ADC	DMA_ADC
11	DMA_IIC	DMA_IIC	DMA_IIC	DMA_IIC
12	DMA_HSCMP1	DMA_SPI1_rx	DMA_SPI1_rx	DMA_SPI2_rx
13	DMA_RTC	DMA_SPI1_tx	DMA_SPI1_tx	DMA_SPI2_tx
14	DMA_SCI1_rx	DMA_SCI1_rx	DMA_SCI2_rx	DMA_SCI2_rx
15	DMA_SCI1_tx	DMA_SCI1_tx	DMA_SCI2_tx	DMA_SCI2_tx

## 16.1.3 Low Power Mode Operation

The iEvent module is capable of functioning in run and wait modes of operation. For details on low-power mode operation, refer to [Table 3-1](#) in [Chapter 3](#), “Modes of Operation.”

## 16.1.4 Clock Gating

The bus clock to the iEvent module can be gated on and off using the SCGC2[iEVT] bit. This bit is set after any reset, which enables the bus clock to this module. See [Section 5.7](#), “Peripheral Clock Gating,” for details.

## 16.1.5 Features

The features of the iEvent module include:

- Highly programmable module for creating combinational boolean events for use as interrupt requests, DMA requests, or hardware triggers

- Each output channel selects 4 event inputs (A, B, C, D) from 16 possible sources
- Evaluates a combinational boolean expression as the sum of four products, where each product term includes all 4 selected input sources available as true or complement values
- Event output is registered and programmed to operate as an interrupt request, a DMA request, or a hardware trigger
- Programmable control for automatic generation of event done signals associated with the appropriate event inputs to properly complete triggering events
- Programmable support of event output as either one-shot or reoccurring modes of operation
- Memory-mapped device connected to the slave peripheral (IPS) bus
  - Programming model organized per channel for simplified software
  - Optional lock mechanism to protect programmed event output configuration
  - Per-channel registers provide state of event output along with selected event input values and control state machine
  - Software maintenance support for channel resets, capture of selected event inputs, and optional forced done broadcast

### 16.1.6 Modes of Operation

The iEvent module does not support any special modes of operation. As shown in the [Figure 16-1](#) and [Figure 16-2](#) block diagrams, its operation is primarily controlled by the selected event inputs and outputs. Additionally, as a memory-mapped device located on the platform's slave peripheral bus, it responds based strictly on memory address for accesses to its programming model and does not consider the operating mode (supervisor, user) of its references.

Normally, the iEvent module resides in the slave peripheral bus clock domain. However, as the module may have event outputs defined to interface with a DMA controller, the iEvent also supports connections to this controller in the platform clock domain, which typically operates at a higher frequency. The iEvent module expects the DMA controller to deal with the hardware issues associated with any clock domain crossings.

## 16.2 External Signal Description

The iEvent module does not directly support any external interfaces. There may be package input signals (indirectly) connected to the module as event inputs, but since the iEvent does not include any input synchronization hardware, this function must be handled before the event input signals are routed into the module.

## 16.3 Memory Map and Register Definition

The iEvent module provides a 512-byte software programming model organized by event output channel number. In this document, the memory map is shown supporting 8 event output channels, but the actual number of implemented registers is determined by hardware configuration variables. Additionally, the programming model is organized to support larger numbers of output channels.

The programming model is split into three sections: a byte-wide data register per channel, a byte-wide control register per channel, and 2 longword configuration registers per channel.

The programming model can be referenced using any operand size (byte, word, longword) and attempted accesses of undefined/unpopulated spaces within the 512-byte space are terminated with an error response.

### 16.3.1 Memory Map

The iEvent programming model map is shown in [Table 16-2](#). In this table, “n” is the event output channel number.

**Table 16-2. iEvent Memory Map with n = 0, 1, ..., 15**

Offset Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/Page
0x000 + n	IEVENT_DRn	iEvent Data Register “n”	8	R/W	0x00	<a href="#">16.3.2.1/16-7</a>
0x080 + n	IEVENT_CRn	iEvent Control Register “n”	8	R/W	0x00	<a href="#">16.3.2.2/16-9</a>
0x100 + 8n	IEVENT_IMXCRn	iEvent Input Mux Configuration Register “n”	32	R/W	0x0000_0000	<a href="#">16.3.2.3/16-11</a>
0x104 + 8n	IEVENT_BFECRn	iEvent Boolean Function Evaluation Configuration Register “n”	32	R/W	0x0000_0000	<a href="#">16.3.2.4/16-12</a>

For this implementation of the iEvent module supporting 4 event output channels, there are certain spaces within the 512-byte memory map that always generate an error response on any attempted reference. These spaces include addresses 0x008-0x07F, 0x088 - 0x0FF, and 0x140 - 0x1FF. Additionally, depending on the number of implemented output channels, there may be additional spaces mapped to unimplemented registers that also generate an error termination on an attempted reference.

A bit in the control register (IEVENT\_CRn) provides a read-only capability to lock out any configuration writes. This capability disables subsequent writes to the configuration registers until the next hardware reset.

### 16.3.2 Register Descriptions

The iEvent module supports access to its programming model via an 8-bit slave peripheral bus connection. Performance is typically maximized by referencing the registers using their natural size, byte for the data or control registers, and longword for the configuration registers.

#### 16.3.2.1 iEvent Data Register “n” (IEVENT\_DRn)

The IEVENT\_DRn register contains the state of the event output and the state of the 4 selected event inputs at the time when the output was asserted. The register is read/write, although the state of the selected event inputs and the event output state machine are read-only. At reset, the contents of the data register is cleared.

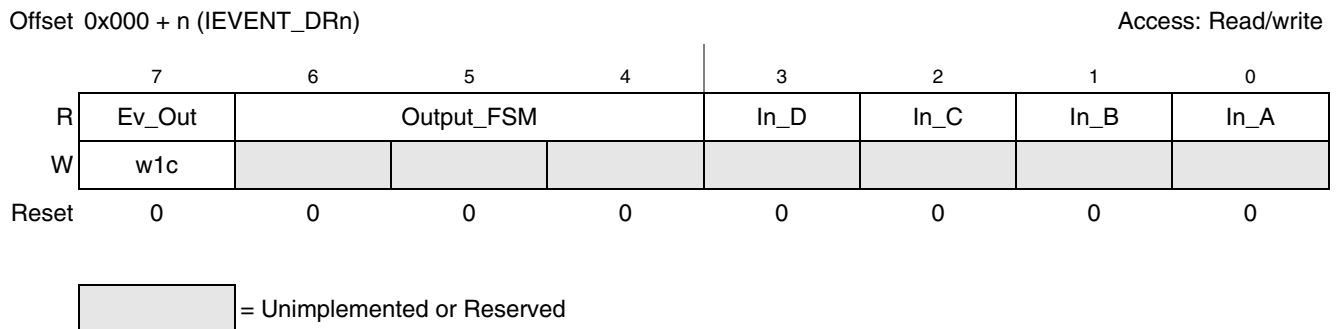
If the event output is programmed as an interrupt request, the event output must be cleared by writing a 1 to this bit to negate the request. This negation operation is typically performed in an interrupt service routine after the appropriate event inputs have been negated (if required).

If the event output is programmed as a DMA request or a hardware trigger, the event output state is automatically cleared by the hardware and attempted writes are ignored.

The register also includes the 3-bit finite state machine to control the event output. While the contents of the state machine are read-only and are provided for software monitoring and debug purposes, special iEvent functions are performed based on writes to the register specifying 2 unused FSM encodings. If the channel is disabled (IEVENT\_CRn[Type] = 00), a write to this register with the 3-bit field set to 0b010 immediately captures the event inputs and optionally forces a broadcast done (based on the state of IEVENT\_CRn[DDB]) to the selected event inputs that are asserted at that time. Additionally, any write to this register with the 3-bit field set to 0b110 generates a soft reset to the event output logic. Any soft reset clears the event output control machine and forces it to the idle state. Both capabilities are provided for software maintenance of the event channel.

**NOTE**

The operation of this register is not affected by the state of IEVENT\_CRn[RO].



**Figure 16-3. iEvent Data Register n (IEVENT\_DRn)**

Table 16-3. IEVENT\_DRn Field Descriptions

Field	Description
7 Ev_Out	<p>Event Output. This bit represents the state of the configured event output.</p> <p>0 The event output is negated. 1 The event output is asserted.</p> <p>If the event output is configured as an interrupt request, this flag must be cleared by writing a 1 to this bit. If the event output is configured as a DMA request or a hardware trigger, this bit is read-only and cleared automatically by the hardware.</p>
6–4 Output_FSM	<p>Output Finite State Machine. This 3-bit field represents the contents of the finite state machine (FSM) controlling the event output. It is provided for software monitoring and debug purposes. The state encodings are:</p> <p>000 The channel is disabled; <i>FSM = idle</i>. 001 The channel is enabled and waiting for the programmed event to occur; <i>FSM = wfevt</i>. 011 The programmed event has occurred and the channel is asserting its output; <i>FSM = assert</i>. 111 The event output has been acknowledged; done signals are broadcast during 1-cycle state; <i>FSM = done</i>. 101 The channel is waiting for the selected input signals to be negated before rearming; <i>FSM = wfclr</i>. 100 The channel is programmed for one-shot operation and the event has completed; <i>FSM = os_hold</i>.</p> <p>Although this field is read-only, writes to the register with the two unused state encodings force special channel actions. These capabilities are provided for software maintenance of the channel. If the channel is disabled (<i>IEVENT_CRn[Type] = 00</i>), a write with this 3-bit field set to 0b010 immediately captures the inputs and optionally forces a broadcast done, based on the state of <i>IEVENT_CRn[DDB]</i>, to the selected event inputs that are asserted at that time. If the channel is not disabled, this write is ignored. A write using this encoding, but with <i>IEVENT_CRn[DDB] = 1</i>, can be performed to simply capture the state of the selected event inputs that are captured in the <i>IEVENT_DRn[IN_*]</i> register bits, without the broadcast done. Additionally, any write with the 3-bit field set to 0b110 generates a soft reset to the event output logic. Both capabilities are provided for software maintenance of the event channel.</p>
3–0 In_*	<p>Event Inputs {D, C, B, A}. These bits represent the selected event inputs at different times, depending on the specific state of the event output.</p> <p>While <i>FSM = idle</i> or <i>wfevt</i>, these bits are undefined. While <i>FSM = assert</i>, these bits reflect the state of the selected event inputs when the event output was initially asserted. During the <i>FSM = done</i> state, these bits reflect the state of the selected event inputs that must receive the broadcast done, based on their state when the event output was initially asserted. While <i>FSM = wfclr</i>, these bits reflect the accumulated state of the negated event inputs. In this state, the individual bits are cleared once the selected event input is negated and remain in that state regardless of the subsequent value of the event input. If an input capture and optional forced broadcast done to a disabled channel occurs, these bits reflect the state of the select event inputs at that time.</p> <p>0 The state of the event input was negated. 1 The state of the event input was asserted.</p> <p>These bits are read-only.</p>

### 16.3.2.2 iEvent Control Register “n” (IEVENT\_CRn)

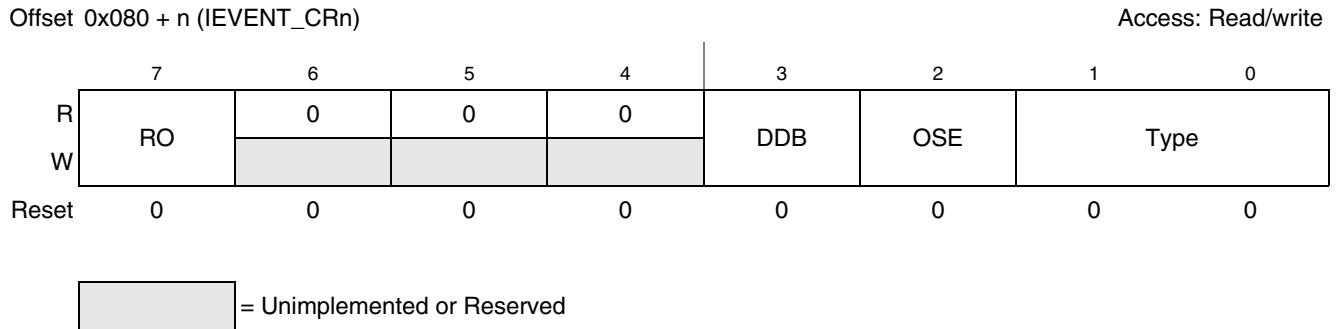
The *IEVENT\_CRn* register provides basic control functions for the event output.

The register contains the output type, a read-only flag for disabling subsequent writes to the event output configuration, control to determine if the output operates once (a one-shot) or can automatically rearm to occur multiple times, and control to determine if done conditions are automatically broadcast to the appropriate event inputs. The register is read/write. At reset, the contents of the control register disable the channel.

Software-generated channel resets (soft resets) may be performed in response to certain writes to the register. Specifically, if the write operand indicates the channel type is to become disabled, or the low-order 4 bits of the write operand are different than the current contents of the control register, a soft reset of the channel is performed. Any soft reset clears the event output control machine, forcing it to the idle state as reflected in the IEVENT\_DRn register.

**NOTE**

The operation of this register is not affected by the state of IEVENT\_CRn[RO].



**Figure 16-4. iEvent Control Register n (IEVENT\_CRn)**

**Table 16-4. IEVENT\_CRn Field Descriptions**

Field	Description
7 RO	<p>Read-Only. This bit determines if the configuration registers can be written or are read-only. This is provided to support a mechanism where the configuration (IEVENT_IMXCRn, IEVENT_BFECRn) can be protected from accidental writes. Once set, this bit remains set until the next hardware reset event. When set, an attempted write to the IEVENT_IMXCRn or IEVENT_BFECRn register is terminated with an error and does not change the register's contents.</p> <p>0 The iEvent configuration registers for this channel can be read and written.                      1 The iEvent configuration registers for this channel can only be read.</p>
3 DDB	<p>Disable Done Broadcast. This bit determines if the event done signal is broadcast to the appropriate selected event inputs. This bit would also be typically programmed as a 0, enabling the done broadcast, regardless of the channel type. This bit also enables the forced broadcast done in response to a specific write to the IEVENT_DRn register.</p> <p>The event done signal is defined based on the channel type. For a DMA request channel, the done signal is based on the assertion of dma_done. For a hardware trigger channel, the done corresponds to the first cycle after the event output state is asserted. For an interrupt request channel, the done corresponds to the cycle after the write of the data register with the event output bit (IEVENT_DRn[Ev_Out]) asserted.</p> <p>0 The event done indicators are broadcast to the appropriate selected event inputs.                      1 The event done indicators are not broadcast to the appropriate selected event inputs.</p>



Table 16-4. IEVENT\_CRn Field Descriptions (continued)

Field	Description
2 OSE	One-Shot Enable. This bit determines if the event output is intended to operate one time (a “one-shot”) or is enabled to automatically rearm itself and operate multiple times. 0 The channel event output is enabled to occur multiple times. 1 The channel event output is enabled to occur one time.
1–0 Type	Output Channel Type. This 2 bit field defines the type of event output for this channel. 00 The channel is disabled. 01 The channel is defined as a DMA request output. 10 The channel is defined as a hardware trigger output. 11 The channel is defined as an interrupt request output.

### 16.3.2.3 iEvent Input Mux Configuration Register “n” (IEVENT\_IMXCRn)

The IEVENT\_IMXCRn register defines the configuration for the selection of the four event inputs (A, B, C, D). Each byte of this register provides the bit number of the event input to be selected as A, B, C, or D.

To prevent any spurious triggering of event outputs and unintended system behavior, it is recommended that the channel be disabled (IEVENT\_CRn[Type] = 00) when writing to the IEVENT\_IMXCRn register.

If IEVENT\_CRn[RO] = 1, this register can only be read and attempted writes are terminated with an error.

Offset 0x100 + n (IEVENT\_IMXCRn)

Access: Read/write

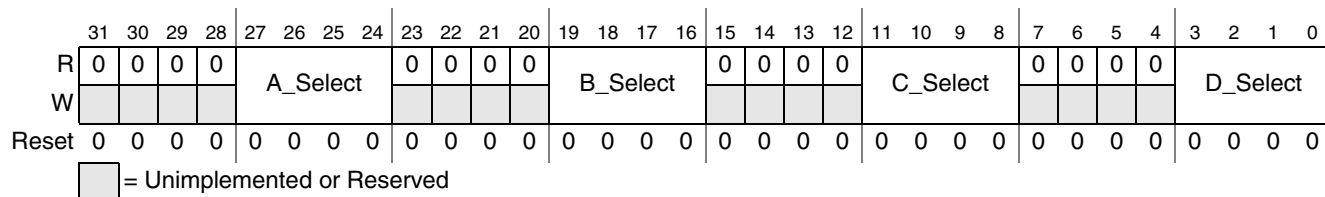


Figure 16-5. iEvent Input Mux Configuration Register n (IEVENT\_IMXCRn)

Table 16-5. IEVENT\_IMXCRn Field Descriptions

Field	Description
27–24 A_Select	<p>A_Select. This 4-bit field defines the bit number of the 16-bit event input vector to be selected as input A.</p> <p>0x0 Select input[0] as A 0x1 Select input[1] as A 0x2 Select input[2] as A ... 0xE Select input[14] as A 0xF Select input[15] as A</p> <p>If the state of the IEVENT_BFECDn register signals that this input is unused in all product terms or the IEVENT_CRn signals the channel is disabled, the selection logic is disabled to minimize unnecessary switching and power dissipation.</p>
19–16 B_Select	<p>B_Select. This 4-bit field defines the bit number of the 16-bit event input vector to be selected as input B.</p> <p>0x0 Select input[0] as B 0x1 Select input[1] as B 0x2 Select input[2] as B ... 0xE Select input[14] as B 0xF Select input[15] as B</p> <p>If the state of the IEVENT_BFECDn register signals that this input is unused in all product terms or the IEVENT_CRn signals the channel is disabled, the selection logic is disabled to minimize unnecessary switching and power dissipation.</p>
11–8 C_Select	<p>C_Select. This 4-bit field defines the bit number of the 16-bit event input vector to be selected as input C.</p> <p>0x0 Select input[0] as C 0x1 Select input[1] as C 0x2 Select input[2] as C ... 0xE Select input[14] as C 0xF Select input[15] as C</p> <p>If the state of the IEVENT_BFECDn register signals that this input is unused in all product terms or in the IEVENT_CRn signals the channel is disabled, the selection logic is disabled to minimize unnecessary switching and power dissipation.</p>
3–0 D_Select	<p>D_Select. This 4-bit field defines the bit number of the 16-bit event input vector to be selected as input D.</p> <p>0x0 Select input[0] as D 0x1 Select input[1] as D 0x2 Select input[2] as D ... 0xE Select input[14] as D 0xF Select input[15] as D</p> <p>If the state of the IEVENT_BFECDn register signals that this input is unused in all product terms or the IEVENT_CRn signals the channel is disabled, the selection logic is disabled to minimize unnecessary switching and power dissipation.</p>

#### 16.3.2.4 iEvent Boolean Function Evaluation Configuration Register “n” (IEVENT\_BFECDn)

The IEVENT\_BFECDn register defines the configuration for the evaluation of the boolean function defining the event output.

The iEvent module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). Specifically, the raw event output is defined by the following  $4 \times 4$  boolean expression:

```
raw_event_output
= (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 0
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 1
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 2
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 3
```

where each selected input of each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. Each product term uses 8 bits of configuration information and 2 bits for each of the four selected event inputs. The resulting logic provides a simple yet powerful boolean function evaluation for defining an event output.

To prevent any spurious triggering of event outputs and unintended system behavior, it is recommended that the channel be disabled (IEVENT\_CRn[Type] = 00) when writing to the IEVENT\_BFECRn register.

If IEVENT\_CRn[RO] is set, this register can only be read and attempted writes are terminated with an error.

Offset 0x104 + n (IEVENT\_BFECRn)

Access: Read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PT0_	PT0_	PT0_	PT0_	PT1_	PT1_	PT1_	PT1_	PT2_	PT2_	PT2_	PT2_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_	PT3_
W	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC	AC	BC	CC	DC
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 16-6. iEvent Boolean Function Evaluation Configuration Register n (IEVENT\_BFECRn)**

**Table 16-6. IEVENT\_BFECRn Field Descriptions**

Field	Description
31–30 PT0_AC	Product Term 0, A input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input A in product term 0. 00 Force the A input in this product term to a logical zero. 01 Pass the A input in this product term. 10 Complement the A input in this product term. 11 Force the A input in this product term to a logical one.
29–28 PT0_BC	Product Term 0, B input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input B in product term 0. 00 Force the B input in this product term to a logical zero. 01 Pass the B input in this product term. 10 Complement the B input in this product term. 11 Force the B input in this product term to a logical one.
27–26 PT0_CC	Product Term 0, C input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input C in product term 0. 00 Force the C input in this product term to a logical zero. 01 Pass the C input in this product term. 10 Complement the C input in this product term. 11 Force the C input in this product term to a logical one.

**Table 16-6. IEVENT\_BFECDn Field Descriptions (continued)**

Field	Description
25–24 PT0_DC	Product Term 0, D input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input D in product term 0. 00 Force the D input in this product term to a logical zero. 01 Pass the D input in this product term. 10 Complement the D input in this product term. 11 Force the D input in this product term to a logical one.
23–22 PT1_AC	Product Term 1, A input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input A in product term 1. It uses the same encoding as PT0_AC.
21–20 PT1_BC	Product Term 1, B input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input B in product term 1. It uses the same encoding as PT0_BC.
19–18 PT1_CC	Product Term 1, C input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input C in product term 1. It uses the same encoding as PT0_CC.
17–16 PT1_DC	Product Term 1, D input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input D in product term 1. It uses the same encoding as PT0_DC.
15–14 PT2_AC	Product Term 2, A input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input A in product term 2. It uses the same encoding as PT0_AC.
13–12 PT2_BC	Product Term 2, B input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input B in product term 2. It uses the same encoding as PT0_BC.
11–10 PT2_CC	Product Term 2, C input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input C in product term 2. It uses the same encoding as PT0_CC.
9–8 PT2_DC	Product Term 2, D input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input D in product term 2. It uses the same encoding as PT0_DC.
7–6 PT3_AC	Product Term 3, A input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input A in product term 3. It uses the same encoding as PT0_AC.
5–4 PT3_BC	Product Term 3, B input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input B in product term 3. It uses the same encoding as PT0_BC.
3–2 PT3_CC	Product Term 3, C input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input C in product term 3. It uses the same encoding as PT0_CC.
1–0 PT3_DC	Product Term 3, D input Configuration. This 2-bit field defines the boolean evaluation associated with the selected input D in product term 3. It uses the same encoding as PT0_DC.

## 16.4 Functional Description

The iEvent is a highly programmable module for creating combinational boolean outputs for use as interrupt requests, DMA requests, or hardware triggers. Each iEvent output channel, as shown in [Figure 16-2](#), is partitioned into three logic functions:

- Selection of 4 event inputs (A, B, C, D) from 16 possible sources
- Evaluation of a combinational boolean expression as a sum of four products where each product term includes all 4 selected input sources available as true or complement values
- The boolean expression event output is registered and programmed to operate as an interrupt request, a DMA request or a hardware trigger

This section presents examples of the programming model configuration for simple boolean expressions, timing diagrams showing the interaction between the event inputs and outputs, and additional information on the finite state machine to control the event output.

### 16.4.1 Configuration Examples for the Boolean Function Evaluation

The iEvent module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). Specifically, the raw event output is defined by the following  $4 \times 4$  boolean expression:

```
raw_event_output
= (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 0
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 1
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 2
| (0,A,~A,1) & (0,B,~B,1) & (0,C,~C,1) & (0,D,~D,1)// product term 3
```

where each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. Each product term uses 8 bits of configuration information, 2 bits for each of the four selected event inputs.

The actual boolean expression implemented in each channel is:

```
raw_event_output
= (PT0_AC[0] & A | PT0_AC[1] & ~A)// product term 0
  & (PT0_BC[0] & B | PT0_BC[1] & ~B)
  & (PT0_CC[0] & C | PT0_CC[1] & ~C)
  & (PT0_DC[0] & D | PT0_DC[1] & ~D)

| (PT1_AC[0] & A | PT1_AC[1] & ~A)// product term 1
  & (PT1_BC[0] & B | PT1_BC[1] & ~B)
  & (PT1_CC[0] & C | PT1_CC[1] & ~C)
  & (PT1_DC[0] & D | PT1_DC[1] & ~D)

| (PT2_AC[0] & A | PT2_AC[1] & ~A)// product term 2
  & (PT2_BC[0] & B | PT2_BC[1] & ~B)
  & (PT2_CC[0] & C | PT2_CC[1] & ~C)
  & (PT2_DC[0] & D | PT2_DC[1] & ~D)

| (PT3_AC[0] & A | PT3_AC[1] & ~A)// product term 3
  & (PT3_BC[0] & B | PT3_BC[1] & ~B)
  & (PT3_CC[0] & C | PT3_CC[1] & ~C)
  & (PT3_DC[0] & D | PT3_DC[1] & ~D)
```

where the bits of the IEVENT\_BFECDn register correspond to the  $PT\{0-3\}_{\{A,B,C,D\}C[1:0]}$  terms in the equation.

Consider the settings of the 32-bit IEVENT\_BFECDn register for several simple boolean expressions as shown in [Table 16-7](#).

**Table 16-7. IEVENT\_BFECDn Values for Simple Boolean Expressions**

iEvent Output Expression	PT0	PT1	PT2	PT3	IEVENT_BFECDn
A & B	A & B	0	0	0	32'b01011111_00000000_00000000_00000000
A & B & C	A & B & C	0	0	0	32'b01010111_00000000_00000000_00000000

**Table 16-7. IEVENT\_BFECRn Values for Simple Boolean Expressions**

iEvent Output Expression	PT0	PT1	PT2	PT3	IEVENT_BFECRn
(A & B & C) + D	A & B & C	D	0	0	32'b01010111_11111101_00000000_00000000
A + B + C + D	A	B	C	D	32'b01111111_11011111_11110111_11111101
(A & ~B) + (~A & B)	A & ~B	~A & B	0	0	32'b01101111_10011111_00000000_00000000

As you see in these examples, the resulting logic provides a simple yet powerful boolean function evaluation for defining an event output.

## 16.4.2 Event Output Finite State Machine

As the boolean function is evaluated, the event output of each channel is controlled by a simple finite state machine that handles the behavior of the module's primary output signals based on information contained the control register. Consider the basic states of this FSM and their descriptions shown in [Table 16-8](#).

**Table 16-8. Event Output Finite State Machine**

State Name	Description	Transitions From	Transitions To
idle FSM = 000	Idle (reset) state; represents a disabled event output.  Entry into this state is forced on any type of hardware reset event, or the generation of a soft reset (see <a href="#">16.4.4.1/16-18</a> for more information).	reset, done, wfclr	wfevt
wfevt FSM = 001	Wait for event state; output channel is enabled, waiting for the boolean expression to be evaluated as true.	idle, wfclr	assert
assert FSM = 011	Assert the event output; the boolean expression was evaluated as true, assert the output.	wfevt	done
done FSM = 111	1-cycle state immediately following the completion of the assert state; the definition of the completion of the assert state varies by channel type.  If enabled (IEVENT_CRn[DDB] is cleared), the appropriate iEvent done indicators are sent back to the appropriate event input modules during this state.	assert	wfclr, os_hold
wfclr FSM = 101	Wait for event inputs to clear; this state is retained until the appropriate event inputs are negated.	done	wfevt
os_hold FSM = 100	Channel is programmed as one-shot and the event has completed.  The one_shot hold state is retained until a reset event (either soft or hardware) is performed.	done	–

The value of the finite state machine is visible in the IEVENT\_DRn register in bits [6:4]. If the IEVENT\_DRn register is written and the [6:4] field contains one of the unused state encodings (0b010, 0b110), a special channel function is performed. For 0b010, a broadcast done is immediately generated and for 0b110, a soft reset event is initiated.

### 16.4.3 Broadcast Done Details

When the event output transitions to the done state (FSM = 111), the iEvent done output signals are conditionally enabled to broadcast a done indicator to clear the appropriate event inputs. This operation is exactly analogous to a DMA channel's acknowledgement clearing the request from a peripheral.

In all cases, the channel's broadcast done is controlled by the state of IEVENT\_CRn[DDB]. If this Disable Done Broadcast flag is cleared, the hardware enables the iEvent done outputs, else these signals remain negated.

Next, consider the details on the done determination for the selected event inputs. Consider two of the simple boolean expressions shown in [Table 16-7](#).

First, consider the case where the boolean expression is a simple AND function like:  $A \& B \& C$ . For this case, all three selected input signals must be asserted for the event output to occur. Accordingly, when the FSM reaches the done state, the iEvent done signals corresponding to the selected inputs: A, B, and C are asserted for a single cycle to perform the broadcast done.

Second, consider a boolean expression logically summing (OR'ing) multiple product terms, like:  $A + B + C + D$ . Further, let the A input signal be responsible for the assertion of the event output. For this case, the broadcast done signals only the A input, regardless of the state of the other inputs (B, C, D), since this input was responsible for the event output assertion. If there are multiple selected inputs that are asserted during the same cycle, for example, both A and B are asserted, then the broadcast done is signaled to both inputs at the appropriate time. For this specific case, multiple product terms are simultaneously asserted, so the broadcast done is sent to multiple input signals.

The determination of the recipients of the broadcast done is made during the last cycle of the FSM assert state based on the values of the selected input signals and the resulting product terms at the time of the entry into this state. The appropriate done values are then loaded into the IEVENT\_DRn[3:0] register for use in the done state.

Logically, the determination of the iEvent done signals uses the following equations:

```
next_state_IEVENT_DRn[0]           // input signal A
= IEVENT_DRn[0]
&(( IEVENT_BFECDn[31:30] == 0b01) & registered_product_term[0]
| ( IEVENT_BFECDn[23:22] == 0b01) & registered_product_term[1]
| ( IEVENT_BFECDn[15:14] == 0b01) & registered_product_term[2]
| ( IEVENT_BFECDn[7:6]   == 0b01) & registered_product_term[3])

next_state_IEVENT_DRn[1]           // input signal B
= IEVENT_DRn[1]
&(( IEVENT_BFECDn[29:28] == 0b01) & registered_product_term[0]
| ( IEVENT_BFECDn[21:20] == 0b01) & registered_product_term[1]
| ( IEVENT_BFECDn[13:12] == 0b01) & registered_product_term[2]
| ( IEVENT_BFECDn[5:4]   == 0b01) & registered_product_term[3])

...
```

Thus, the assertion of a broadcast done is dependent on the combination of the selected input being asserted and used in its logical true form ( $IEVENT\_BFECDn[*] = 0b01$ ) and the appropriate product term being asserted.

## 16.4.4 Software Channel Maintenance

The iEvent module supports several additional features intended for software maintenance operations. In particular, the module supports the notion of a soft reset to force the channel into a known state similar to that generated by a hardware reset. The module also supports a mechanism where the state of the selected event input signals can be captured with an optional broadcast done generated in a disabled channel to clear any event input indicators.

### 16.4.4.1 Channel Soft Resets

A soft reset can be generated at any time for an iEvent channel by performing a write to either IEVENT\_DRn or IEVENT\_CRn with the following conditions:

1. A write to IEVENT\_CRn where the operand signals the new channel type is disabled.
2. A write to IEVENT\_CRn where any of the low-order 4 bits of the operand are different than the current value contained in the register.
3. A write to the IEVENT\_DRn where the 3-bit operand corresponding to the FSM specifies the unused state value of 0b110, that is,  $\text{ips\_wdata}[6:4] = 0b110$ .

### 16.4.4.2 Input Capture and Optional Forced Broadcast Done

The second maintenance function is the ability to capture the selected event inputs and optionally force a broadcast done (based on the state of IEVENT\_CRn[DDB]) to a disabled channel. This input capture and possible broadcast done is initiated based on a write to the IEVENT\_DRn register with an operand containing the other unused FSM encoding in bits [6:4], specifically  $\text{ips\_wdata}[6:4] = 0b010$ .



# Chapter 17

## Watchdog Timer

### 17.1 Introduction

The watchdog timer keeps a watch on the system functioning and resets it in case of its failure. Some reasons for such failures are: run-away software code and the stoppage of the system clock that in a safety critical system can lead to serious consequences. In such cases, the watchdog brings the system into a safe state of operation. The watchdog monitors the operation of the system by expecting periodic communication from the software, generally known as servicing or refreshing the watchdog. In case, this periodic refreshing does not occur, the watchdog resets the system.

### 17.2 Features

The features of Watchdog include:

- Independent clock source input (independent from CPU/bus clock). Choice between two clock sources:
  - LPO Oscillator
  - External system clock
- Unlock sequence<sup>1</sup> for allowing updates to write-once WDOG control/configuration bits.
- All WDOG control/configuration bits are writable once only within 256 bus clock cycles of being unlocked.
  - You need to always update these bits after unlocking within 256 bus clock cycles. Failure to update these bits, resets the system.
- Programmable time-out period specified in terms of number of WDOG clock cycles.
- Ability to test WDOG timer and reset with a flag indicating watchdog test.
  - Quick test—Small time-out value programmed for quick test.
  - Byte test—Individual bytes of timer tested one at a time.
  - Read only access to WDOG timer<sup>2</sup>—Allows dynamic check that WDOG timer is operational.
- Windowed refresh option
  - Provides robust check that program flow is faster than expected.
  - Programmable window.
  - Refresh outside window leads to reset.
- Robust refresh mechanism

---

1. Unlock sequence is write values of 0xC520 and 0xD928 to WDOG Unlock Register within 20 bus clock cycles.  
2. Reading the watchdog timer counter while running the watchdog on bus clock might not give the accurate counter value.

- Write values of 0xA602 and 0xB480 to WDOG Refresh Register within 20 bus clock cycles.
- Count of WDOG resets as they occur.
- Configurable interrupt on time-out to provide debug breadcrumbs. This is followed by a reset after 256 bus clock cycles.

### 17.3 Functional Overview

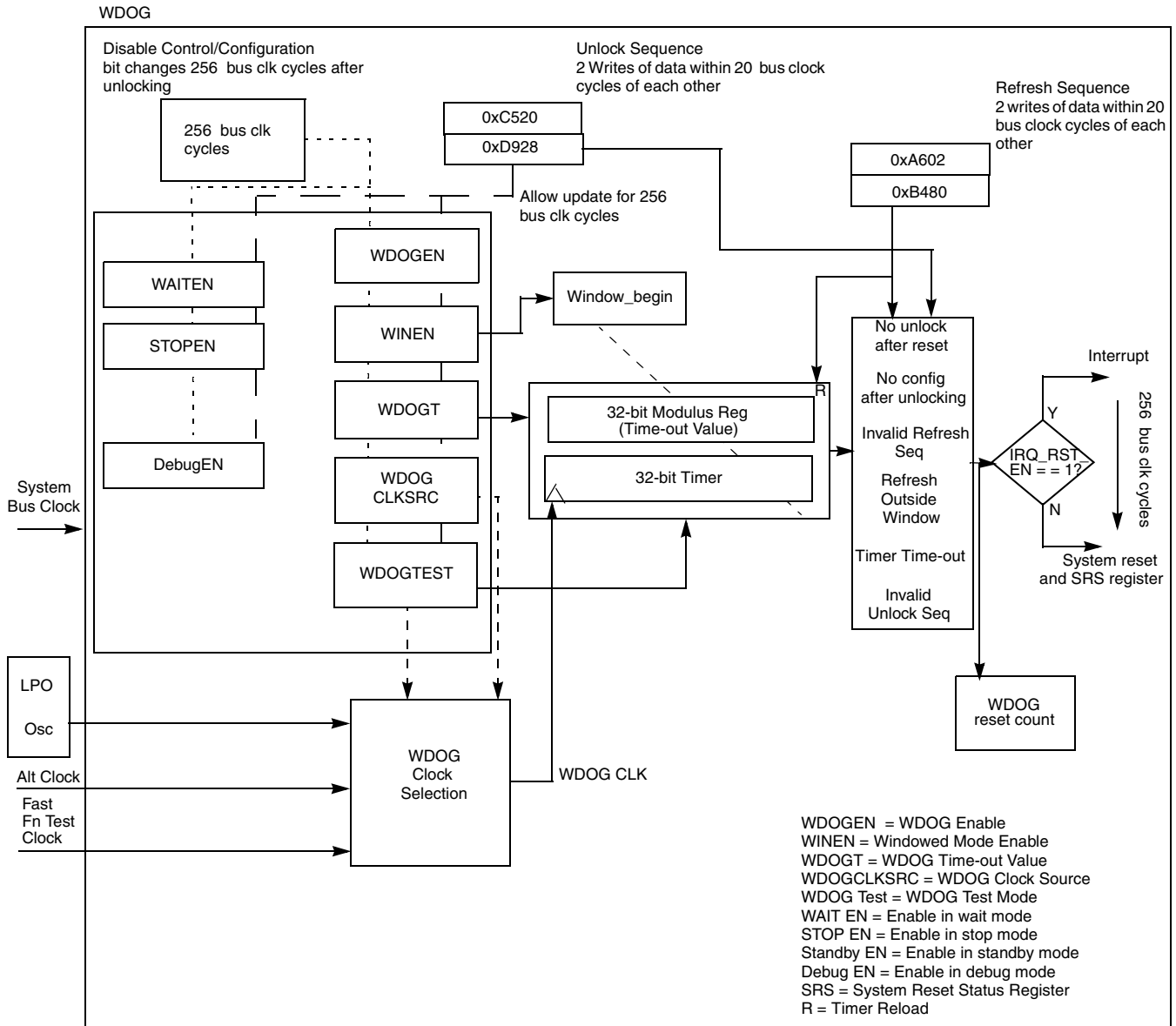


Figure 17-1. WDOG Operation

Figure 17-1 shows operation of the watchdog. The watchdog is a fail safe mechanism that brings the system into a known initial state in case of its failure due to CPU clock stopping or a run away condition in code execution. In its simplest form, the watchdog timer runs continuously off a clock source and expects to be serviced periodically, failing which it resets the system. This ensures that the software is

executing correctly and has not run away in an unintended direction. The period of servicing, or the time-out value for the watchdog timer is user programmable allowing the watchdog to adjust as per the application. You can select a windowed mode of operation that expects the servicing to be done only in a particular window of the time-out period. An attempted servicing of the watchdog outside this window results in a reset. By operating in this mode, you can get an indication of whether the code is running faster than expected. The window length is also user programmable.

If a system fails to update/refresh the watchdog due to an unknown and persistent cause, it will be caught in an endless cycle of resets from the watchdog. To analyze the cause of such conditions, you can program the watchdog to first issue an interrupt, followed a little later by a reset. In the interrupt service routine, the software can analyze the system stack to aid debugging.

To enhance the independence of watchdog from the system, it runs off an independent LPO oscillator clock. You can also switch over to an alternate clock source if required, through a control register bit.

### 17.3.1 Unlocking and Updating the Watchdog

You can unlock the write-once-only control and configuration registers for updating them. As a pre-condition, the `ALLOW_UPDATE` bit in the watchdog control register must be set. The actual unlock is accomplished by writing `0xC520` followed by `0xD928` within 20 bus clock cycles to a specific unlock register (`WDOG_UNLOCK`). This opens up an update window equal in length to the watchdog configuration time (`WCT`) within which you can update the configuration and control register bits. You can not update registers on the bus clock cycle immediately following the write of the unlock sequence, but one cycle later. These register bits can be modified only once after unlocking.

If at least one of the configuration and control registers is not updated within the update window, the watchdog issues a reset (or interrupt-then-reset) to the system. Trying to unlock the watchdog within the `WCT` time after an initial unlock, has no effect. During the update operation, the watchdog timer is not paused and keeps running in the background. After the update window closes, the watchdog timer restarts and the watchdog functions as per the new configuration.

The update feature is useful for applications that have an initial, non-safety critical part, where the watchdog is kept disabled or with a conveniently long time-out period. This means the application coder does not have to bother with frequently servicing the watchdog. After the critical part of the application begins, the watchdog can be reconfigured as per need.

If any value other than `0xC520` or `0xD928` is written to the unlock register, or if you allow a gap of more than 20 bus clock cycles in between the writing of the unlock sequence values, the watchdog issues a reset (interrupt-then-reset if enabled) to the system. This is true even if `ALLOW_UPDATE` is cleared. Also, an attempted refresh operation between the two writes of the unlock sequence and in the `WCT` time following a successful unlock, goes undetected. Also, see [17.7, “Watchdog Operation with 8-bit access,”](#) for guidelines related to 8-bit accesses to the unlock register.

#### NOTE

A context switch during unlocking and refreshing may lead to a watchdog reset.

### 17.3.2 The Watchdog Configuration Time (WCT)

To prevent unintended modification of the watchdog's control and configuration register bits, you are allowed to update them only within a period of 256 bus clock cycles after unlocking. This window period is known as the watchdog configuration time (WCT). In addition, these register bits can be modified only once after unlocking them for editing (even after reset).

You must unlock the registers within WCT time after system reset, failing which the WDOG issues a reset to the system. To be more precise, you must write at least the first word of the unlocking sequence within the WCT time after reset. Once this is done, you get a further 20 bus clock cycles (the maximum allowed gap between the words of the unlock sequence) to complete the unlocking operation. Thereafter, to make sure that you do not forget to configure the watchdog, the watchdog issues a reset if at least one of the WDOG control and configuration registers is not updated in the WCT time after unlock. After the close of this window or after the first write, these register bits are locked out from any further changes.

The watchdog timer keeps running as per its default configuration through unlocking and update operations that can extend up to a maximum total of  $2 \times \text{WCT time} + 20$  bus clock cycles. Therefore, it must be ensured that the time-out value for the watchdog is always greater than  $2 \times \text{WCT time} + 20$  bus clock cycles. The updates in the write-once registers take effect only after the WCT window closes. The exceptions are the stop, wait, debug, and standby mode enable bits, and, the IRQ\_RST\_EN bit, changes in which reflect immediately. The operations of refreshing the watchdog goes undetected during the WCT.

### 17.3.3 Refreshing the Watchdog

A robust refreshing mechanism has been chosen for the watchdog. A valid refresh is a write of 0xA602 followed by 0xB480 within 20 bus clock cycles to watchdog refresh register. If these two values are written more than 20 bus cycles apart or if something other than these two values is written to the register, a watchdog reset (or interrupt-then-reset if enabled) is issued to the system. A valid refresh makes the watchdog timer restart on the next bus clock. Also, an attempted unlock operation, in between the two writes of the refresh sequence goes undetected. See [Section 17.7, "Watchdog Operation with 8-bit access,"](#) for guidelines related to 8-bit accesses to the refresh register.

### 17.3.4 Windowed Mode of Operation

In this mode of operation a restriction is placed, on the point in time, within the time-out period at which the watchdog can be refreshed. The refresh is considered valid only when the watchdog timer increments beyond a certain count as specified by the watchdog window register. This is known as refreshing the watchdog within a window of the total time-out period. If a refresh is attempted before the timer reaches the window value, the watchdog generates a reset (or interrupt-then-reset if enabled). Of course, if there is no refresh at all, the watchdog times out and generates a reset or interrupt-then-reset if enabled.

### 17.3.5 Watchdog Disabled Mode of Operation

When the watchdog is disabled through the WDOG\_EN bit in the watchdog status and control register, the watchdog timer is reset to zero and is disabled from counting until you enable it or it is again enabled by the system reset. In this mode the watchdog timer cannot be refreshed (there is no requirement to do so while the timer is disabled). However, the watchdog still generates a reset (or interrupt-then-reset if

enabled) on a non-time-out exception (see [Section 17.5.1, “Generated Resets and Interrupts”](#)). You need to unlock the watchdog before enabling it. A system reset brings the watchdog out of the disabled mode.

### 17.3.6 Low Power Modes of Operation

- In wait mode, if the WDOG is enabled (`WAIT_EN = 1`), it can run on bus clock or low power oscillator clock (`CLK_SRC = x`) to generate interrupt (`IRQ_RST_EN=1`) followed by a reset on time-out. After reset the WDOG reset counter increments by one.
- In stop modes where the bus clock is gated (stop4/stop3), the WDOG can run only on low power oscillator clock (`CLK_SRC=0`) if it is enabled in stop (`STOP_EN=1`). In this case, the WDOG runs to time-out twice, and then generate a reset from its backup circuitry. Therefore, if you program the watchdog to time-out after 100 ms and enters such a stop mode, the reset will occur after 200 ms. Also, in this case no interrupt will be generated irrespective of the value of `IRQ_RST_EN` bit. After WDOG reset, the WDOG reset counter will also not increment.
- In stop2 mode, the watchdog is powered off.

### 17.3.7 Debug Modes of Operation

You can program the watchdog to disable in debug modes (through `DBG_EN` bit in the watchdog control register). This results in the watchdog timer pausing for the duration of the mode. Register read/writes are still allowed, which means that operations like: refresh, unlock etc. are allowed. On exit from the mode, the timer resumes its operation from the point of pausing.

The entry of the system into the debug mode does not excuse it from compulsorily configuring the watchdog in the WCT time after unlock (unless the system bus clock is gated off, in which case the internal state machine pauses too). Failing to do so still results in a reset (or interrupt-then-reset, if enabled) to the system. Also, all the exception conditions that result in a reset to the system (refer [Section 17.5.1, “Generated Resets and Interrupts”](#)) are still valid in this mode. So, if an exception condition occurs and the system bus clock is on, a reset occur (or interrupt-then-reset, if enabled).

The entry into Debug modes within WCT time after reset is treated differently. The WDOG timer is kept reset to zero and there is no need to unlock and configure it within WCT time. You must not try to refresh or unlock the WDOG in this state or unknown behavior may result. Upon exit from this mode, the WDOG timer restarts and the WDOG has to be unlocked and configured within WCT time.

## 17.4 Testing the Watchdog

For IEC 60730 and other safety standards, the expectation is that anything that monitors a safety function must be tested and this test is required to be fault tolerant. To test the watchdog, its main timer and its associated compare and reset logic must be tested. Towards this end, two tests are implemented for the watchdog that are described in [Section 17.4.1, “Quick Test”](#) and [Section 17.4.2, “Byte Test.”](#) While there is a control bit provided to put the watchdog into the test mode (functional), there is an overriding test-disable control bit which once set, disables the test mode permanently until reset.

For running a particular test, first select that test. Thereafter, set a certain test mode bit to put the watchdog in the functional test mode. Setting this bit automatically switches the watchdog timer to a fast clock source

. The switching of the clock source is done to achieve a faster time-out and hence a faster test. In a successful test, the timer times out after reaching the programmed time-out value and generates a system reset. On emerging out of the reset, the software checks the test mode bit to know that the last reset was due to a watchdog test. This check completes the test.

**NOTE**

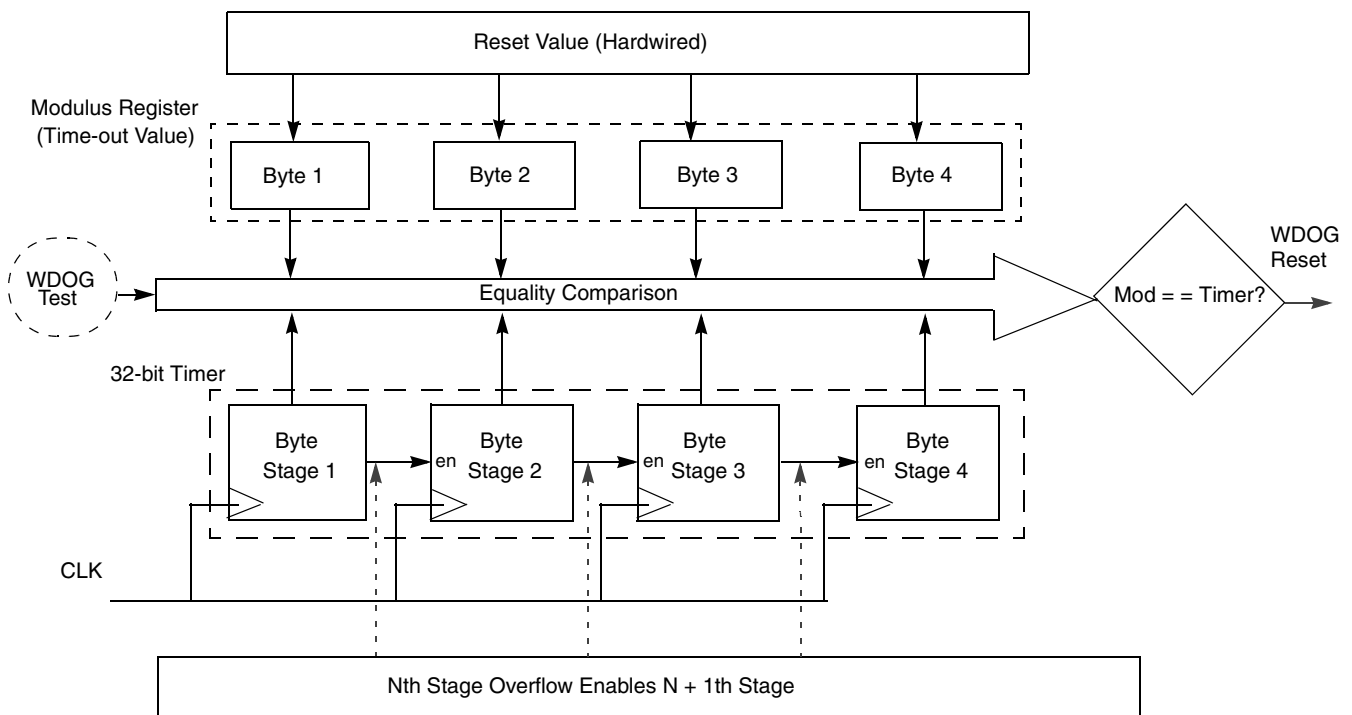
After emerging from a reset due to a watchdog test, you must follow the mandatory steps of unlocking and configuring the watchdog. The refresh and unlock operations and interrupt are not automatically disabled in the test mode.

**17.4.1 Quick Test**

In this test the time-out value of watchdog timer is programmed to a very low value to achieve quick time-out. The only difference between the quick test and the normal mode of functioning of the watchdog is that the test mode bit is set for the quick test.

**17.4.2 Byte Test**

The byte test implements more thorough a test of the watchdog timer. In this test, the timer is split up into its constituent byte-wide stages that are run independently and tested for time-out against the corresponding byte of the time-out value register. [Figure 17-2](#), explains the splitting concept:



**Figure 17-2. Watchdog Timer Byte Splitting**

Each stage is an 8-bit synchronous counter followed by combinational logic that generates an overflow signal. The overflow signal acts as an enable to the  $N + 1$ th stage.

In the test mode, when an individual byte,  $N$ , is tested, byte  $N - 1$  is loaded forcefully with  $0xFF$ , and both these bytes are allowed to run off the clock source. By doing so the overflow signal from stage  $N - 1$  is generated immediately, enabling counter stage  $N$ . The  $N$ th stage runs and compares with the  $N$ th byte of the time-out value register. In this way, the byte  $N$  is also tested along with the link between it and the preceding stage. No other stages,  $N - 2$ ,  $N - 3$ ...and  $N + 1$ ,  $N + 2$ ...are enabled for the test on byte  $N$ . These disabled stages (except the most significant stage of the counter) are loaded with a value of  $0xFF$ .

These two testing schemes achieve the overall aim of testing the counter functioning and the compare and reset logic.

### NOTE

Do not enable the watchdog interrupt during these tests. If required, you must ensure that the effective time-out value is greater than WCT time. See [Section 17.5.1, “Generated Resets and Interrupts,”](#) for more details.

## 17.5 Backup Reset Generator

The backup reset generator generates the final reset which goes out to the system. It has a backup mechanism which takes care that in case the bus clock stops and prevents the main state machine from generating a reset exception/interrupt, the watchdog timer's time-out is separately routed out as a reset to the system. Two successive timer time-outs without an intervening system reset result in the backup reset generator routing out the time-out signal as a reset to the system.

### 17.5.1 Generated Resets and Interrupts

The watchdog generates a reset on the following events (referred to as exceptions at some places in this document):

- A watchdog time-out.
- Failure to unlock the watchdog within WCT time after system reset de-assertion.
- No update of the control and configuration registers within the WCT window after unlocking. At least one of the registers, `WDOG_ST_CTRL_H`, `WDOG_ST_CTRL_L`, `WDOG_TO_VAL_H`, `WDOG_TO_VAL_L`, `WDOG_WIN_H`, and `WDOG_WIN_L`(where implemented), have to be written to within the WCT window to avoid reset.
- A value other than the unlock sequence or the refresh sequence is written to the unlock and/or refresh registers, respectively.
- A gap of more than 20 bus cycles exists between the writes of two values of the unlock sequence.
- A gap of more than 20 bus cycles exists between the writes of two values of the refresh sequence.

The watchdog can also generate an interrupt. If `IRQ_RST_EN` is set, then on the above mentioned events `WDOG_ST_CTRL_L[INT_FLG]` is set, generating an interrupt. A watchdog reset is also generated WCT time later to ensure the watchdog is fault tolerant. The interrupt can be cleared by writing 1 to `INT_FLG`.

The gap of WCT time between interrupt and reset means that the WDOG time-out value must be greater than WCT. Otherwise, if the interrupt was generated due to a time-out, a second consecutive time-out will occur in that WCT gap. This will trigger the backup reset generator to generate a reset to the system, prematurely ending the interrupt service routine execution. Also, the jobs like counting the number of watchdog resets would not be done.

## 17.6 Register Space

### 17.6.1 Memory Map

**Table 17-1. Watchdog Module Register Map**

Offset	Register	Width (bits)	Access	Reset Value
0x0000	Watchdog Status and Control Register High (WDOG_ST_CTRL_H)	16	Write Once	0x00D3
0x0002	Watchdog Status and Control Register Low (WDOG_ST_CTRL_L)	16	Write Once and write 1 to clear	0x0001
0x0004	Watchdog Time-out Value Register High (WDOG_TO_VAL_H)	16	Write Once	0x004C
0x0006	Watchdog Time-out Value Register Low (WDOG_TO_VAL_L)	16	Write Once	0x4B40
0x0008	Watchdog Window Register High (WDOG_WIN_H)	16	Write Once	0x0000
0x000A	Watchdog Window Register Low (WDOG_WIN_L)	16	Write Once	0x0010
0x000C	Watchdog Refresh Register (WDOG_REFRESH)	16	R/W	0xB480
0x000E	Watchdog Unlock Register (WDOG_UNLOCK)	16	R/W	0xD928
0x0010	Watchdog Timer Output Register High (WDOG_TIMER_OUT_H)	16	Read Only	0x0000
0x0012	Watchdog Timer Output Register Low (WDOG_TIMER_OUT_L)	16	Read Only	0x0000
0x0014	Watchdog Reset and Count Register (WDOG_RST_CNT)	16	Write 1 to clear	0x0000

As shown in [Table 17-1](#), the watchdog module has a Big Endian organization.

### 17.6.2 Register Description

Reading reserved bits in a register returns zero, while writing to the same has no effect.



### 17.6.2.1 Watchdog Status and Control Register High (WDOG\_ST\_CTRL\_H)

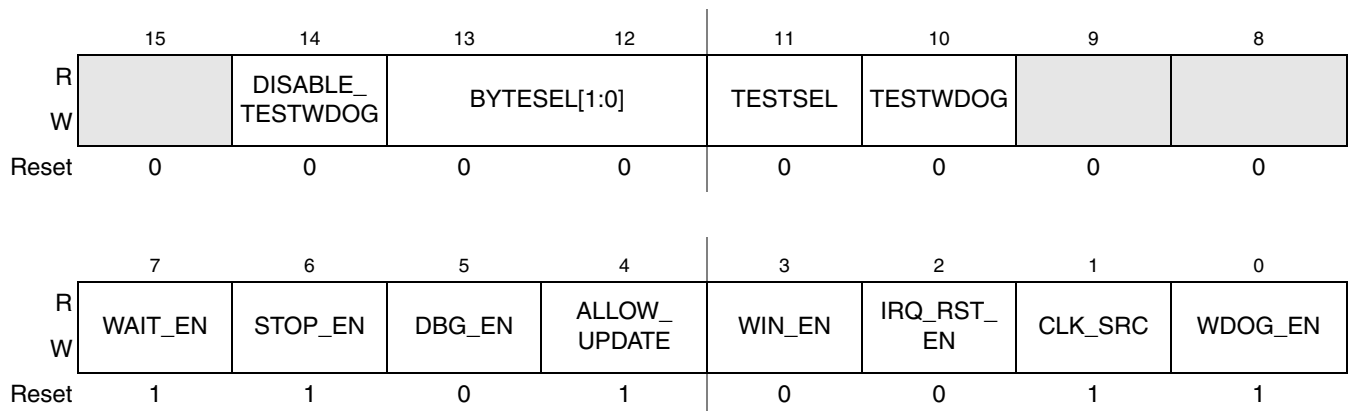


Figure 17-3. WDOG\_ST\_CTRL\_H Register

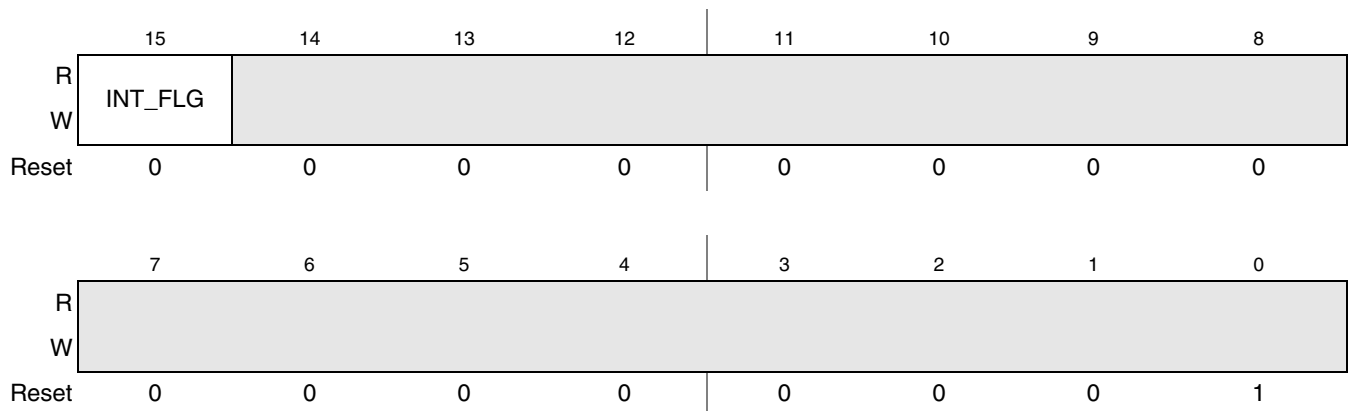
Table 17-2. WDOG\_ST\_CTRL\_H Register Field Descriptions

Field	Description
15	Reserved.
14 DISABLE_ TESTWDOG	Allows the WDOG's functional test mode to be disabled permanently. Once set, it can only be cleared by a reset. It cannot be unlocked for editing once it is set. 0 WDOG functional test mode is not disabled. 1 WDOG functional test mode is disabled permanently until reset.
[13:12] BYTESEL[1:0]	This 2-bit field select the byte to be tested when the watchdog is in the byte test mode. 00 = Byte 0 selected 01 = Byte 1 selected 10 = Byte 2 selected 11 = Byte 3 selected For more details on the byte test, see <a href="#">Section 17.4, "Testing the Watchdog."</a>
11 TESTSEL	Selects the test to be run on the watchdog timer. Effective only if TESTWDOG is set. 0 Quick test. The timer runs in normal operation. You can load a small time-out value to do a quick test. 1 Byte test. Puts the timer in the byte test mode where individual bytes of the timer are enabled for operation and are compared for time-out against the corresponding byte of the programmed time-out value. Select the byte through BYTESEL[1:0] for testing. For more details on these tests, see <a href="#">Section 17.4, "Testing the Watchdog."</a>
10 TESTWDOG	Puts the watchdog in the functional test mode. In this mode the watchdog timer and the associated compare and reset generation logic is tested for correct operation. The clock for the timer is switched from the main watchdog clock to the fast clock input for watchdog functional test. The TESTSEL bit selects the test to be run. The watchdog generated reset does not clear this bit.
[9:8]	Reserved.
7 WAIT_EN	Enables or disables WDOG in wait mode. 0 WDOG is disabled in CPU wait mode. 1 WDOG is enabled in CPU wait mode.
6 STOP_EN	Enables or disables WDOG in stop mode. 0 WDOG is disabled in CPU stop mode. 1 WDOG is enabled in CPU stop mode.

**Table 17-2. WDOG\_ST\_CTRL\_H Register Field Descriptions (continued)**

Field	Description
5 DBG_EN	Enables or disables WDOG in Debug mode. 0 WDOG is disabled in CPU Debug mode. 1 WDOG is enabled in CPU Debug mode.
4 ALLOW_UPDATE	Enables updates to watchdog write once registers, after initial configuration window (WCT) closes, through unlock sequence. 0 No further updates allowed to WDOG write once registers. 1 WDOG write once registers can be unlocked for updating.
3 WIN_EN	Enable windowing mode. 0 Windowing mode is disabled. 1 Windowing mode is enabled.
2 IRQ_RST_EN	Used to enable the debug breadcrumbs feature. A change in this bit is updated immediately, as opposed to updating after WCT. 0 WDOG time-out generates reset only. 1 WDOG time-out initially generates an interrupt. After WCT time, it generates a reset.
1 CLK_SRC	Selects clock source for the WDOG timer and other internal timing operations. 0 Dedicated clock source selected as WDOG clock (LPO Oscillator). 1 WDOG clock sourced from alternate clock source.
0 WDOG_EN	Enables or disables the WDOG's operation. In the disabled state, the watchdog timer is kept in the reset state, but the other exception conditions can still trigger a reset/interrupt. A change in the value of this bit must be held for more than one WDOG_CLK cycle for the WDOG to be enabled or disabled. 0 WDOG is disabled. 1 WDOG is enabled.

### 17.6.2.2 Watchdog Control and Status Register Low (WDOG\_ST\_CTRL\_L)

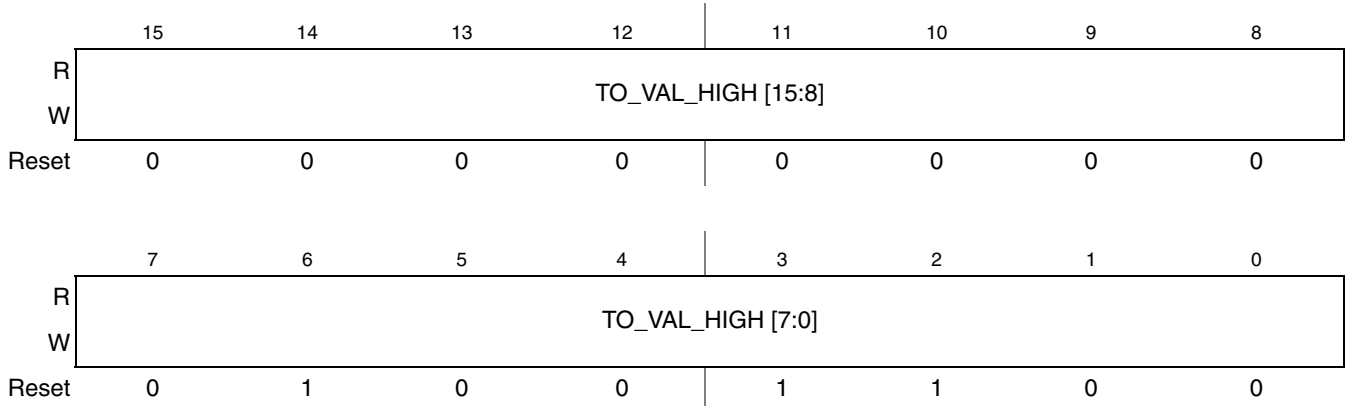


**Figure 17-4. WDOG\_ST\_CTRL\_L Register**

**Table 17-3. WDOG\_ST\_CTRL\_L Register Field Descriptions**

Field	Description
15 INT_FLG	Interrupt flag. It is set when an exception as listed in <a href="#">Section 17.5.1, “Generated Resets and Interrupts”</a> occurs. IRQ_RST_EN = 1 is a precondition to set this flag. INT_FLG = 1 results in an interrupt being issued followed by a reset, WCT time later. The interrupt can be cleared by writing 1 to this bit. It also gets cleared on a system reset.
[14:0]	Reserved. Writes to these bits are not recommended.

### 17.6.2.3 Watchdog Time-out Value Register High (WDOG\_TO\_VAL\_H)

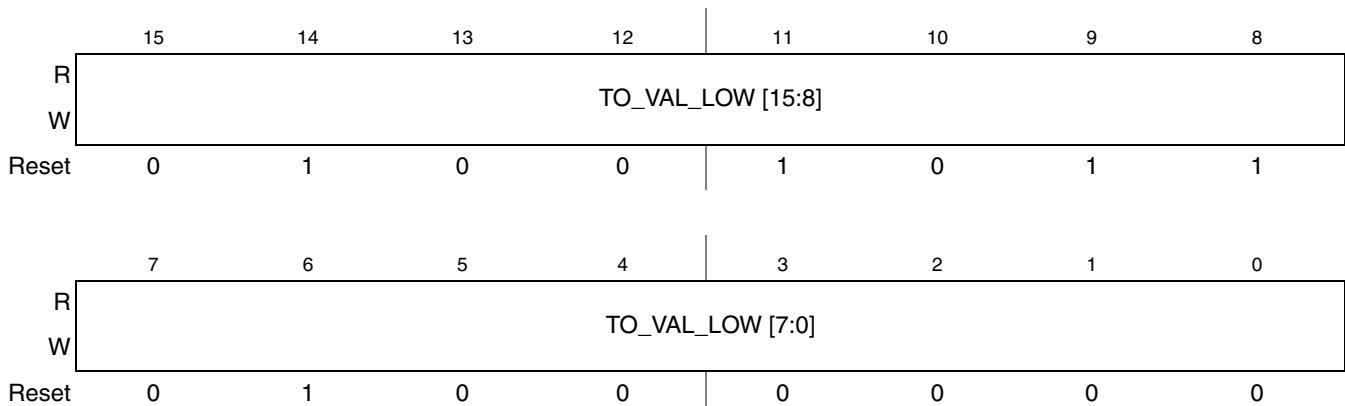


**Figure 17-5. WDOG\_TO\_VAL\_H Register**

**Table 17-4. WDOG\_TO\_VAL\_H Register Field Descriptions**

Field	Description
[15:0] TO_VAL_HIGH [15:0]	Defines the upper 16 bits of the 32-bit time-out value for the watchdog timer. It is defined in terms of cycles of the watchdog clock.

### 17.6.2.4 Watchdog Time-out Value Register Low (WDOG\_TO\_VAL\_L)



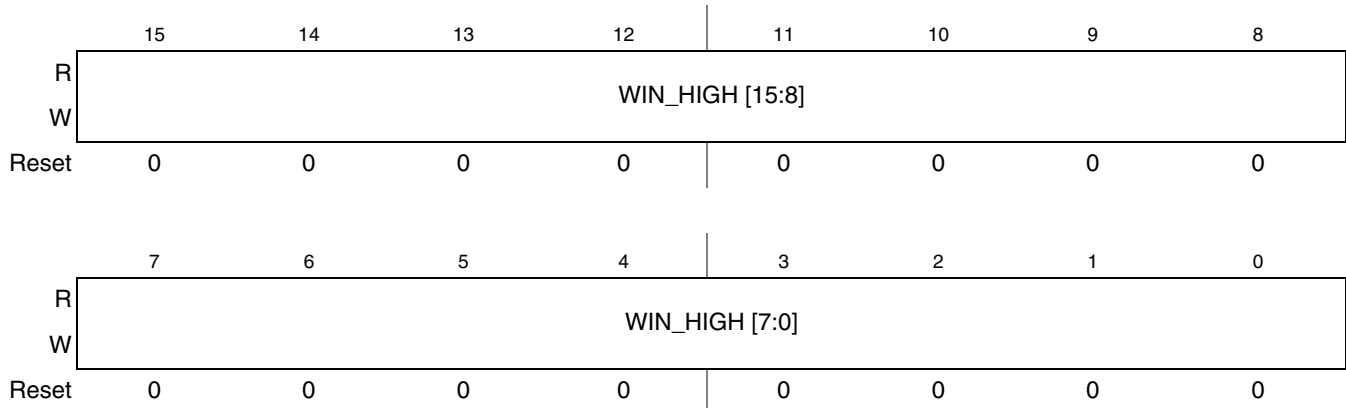
**Figure 17-6. WDOG\_TO\_VAL\_L Register**

**Table 17-5. WDOG\_TO\_VAL\_L Register Field Descriptions**

Field	Description
[15:0] TO_VAL_LOW [15:0]	Defines the lower 16 bits of the 32-bit time-out value for the watchdog timer. It is defined in terms of cycles of the watchdog clock.

The time-out value of the watchdog must be set to a minimum of four watchdog clock cycles. This is to take into account the delay in new settings taking effect in the watchdog clock domain.

### 17.6.2.5 Watchdog Window Register High (WDOG\_WIN\_H)<sup>1</sup>



**Figure 17-7. WDOG\_WIN\_H Register**

**Table 17-6. WDOG\_WIN\_H Register Field Descriptions**

Field	Description
[15:0] WIN_HIGH [15:0]	Defines the upper 16 bits of the 32-bit window for the windowed mode of operation of the watchdog. It is defined in terms of cycles of the watchdog clock. In this mode the watchdog can be refreshed only when the timer has reached a value greater than or equal to this window length. A refresh outside this window resets the system or if IRQ_RST_EN is set, it interrupts and then resets the system.

1. You must set the Window Register value lower than the Time-out Value Register.

### 17.6.2.6 Watchdog Window Register Low (WDOG\_WIN\_L)<sup>1</sup>

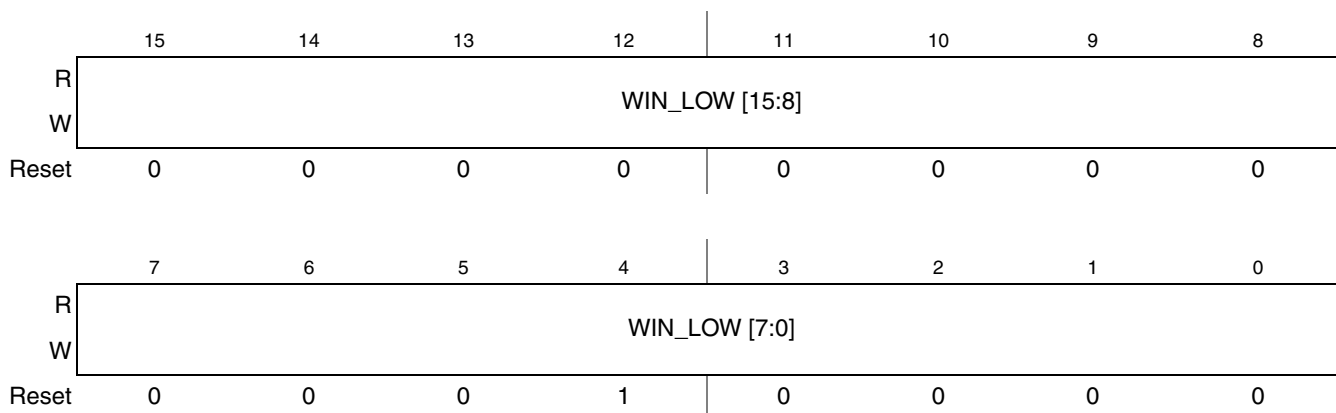


Figure 17-8. WDOG\_WIN\_L Register

Table 17-7. WDOG\_WIN\_L Register Field Descriptions

Field	Description
[15:0] WDOG_WIN_L [15:0]	Defines the lower 16 bits of the 32-bit window for the windowed mode of operation of the watchdog. It is defined in terms of cycles of the pre-scaled watchdog clock. In this mode, the watchdog can be refreshed only when the timer reaches a value greater than or equal to this window length value. A refresh outside this window resets the system or if IRQ_RST_EN is set, it interrupts and then resets the system.

### 17.6.2.7 Watchdog Refresh Register (WDOG\_REFRESH)

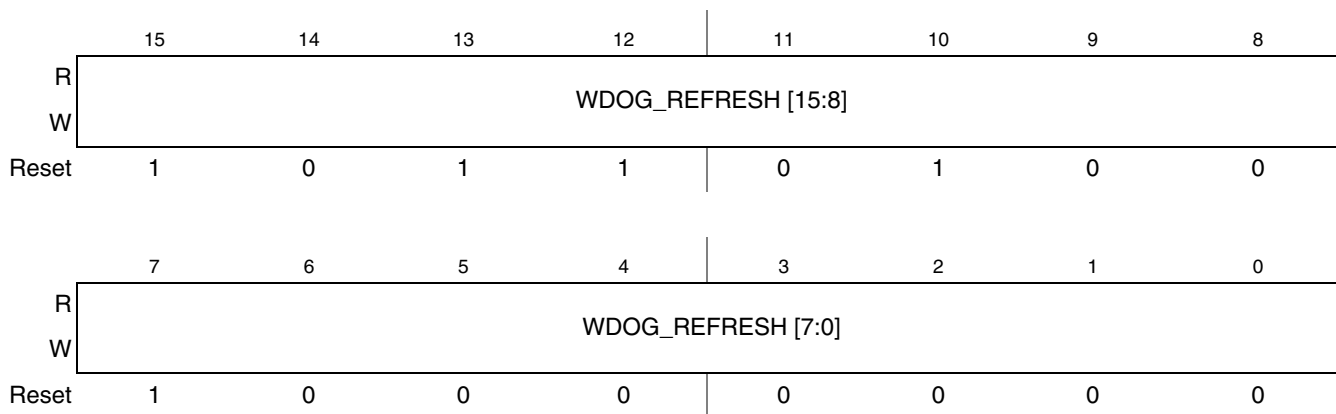


Figure 17-9. WDOG\_REFRESH Register

Table 17-8. WDOG\_REFRESH Register Field Descriptions

Field	Description
[15:0] WDOG_REFRESH [15:0]	Watchdog refresh register. A sequence of 0xA602 followed by 0xB480 within 20 bus clock cycles when written to this register, refreshes the WDOG and prevents it from resetting the system. Writing a value other than the above mentioned sequence or if the sequence is longer than 20 bus cycles, resets the system or if IRQ_RST_EN is set, it interrupts and then resets the system).

1. You must set the Window Register value lower than the Time-out Value Register.

### 17.6.2.8 Watchdog Unlock Register (WDOG\_UNLOCK)

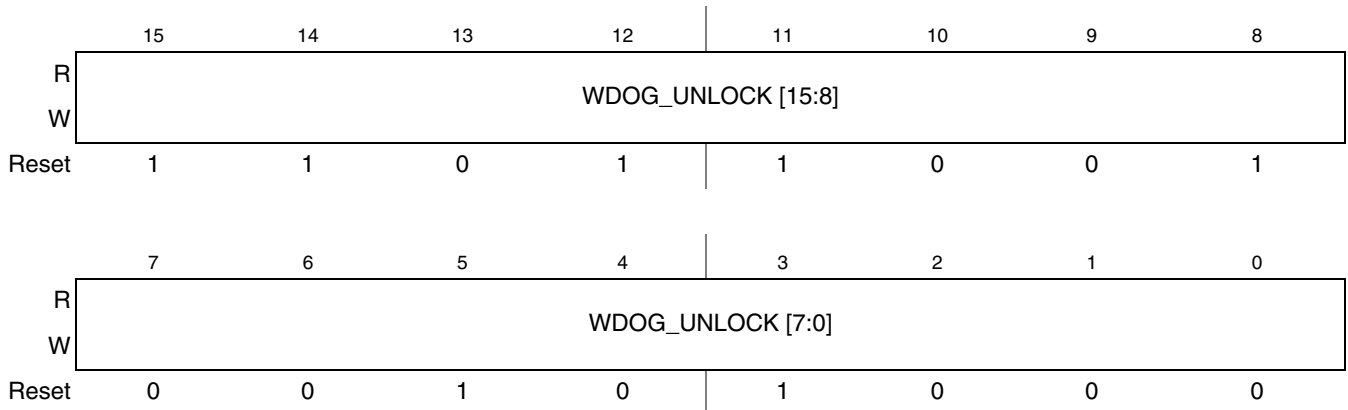


Figure 17-10. WDOG\_UNLOCK Register

Table 17-9. WDOG\_UNLOCK Register Field Descriptions

Field	Description
[15:0] WDOG_UNLOCK [15:0]	You can write the unlock sequence values to this register to make the watchdog write once registers writable again. The required unlock sequence is 0xC520 followed by 0xD928 within 20 bus clock cycles. A valid unlock sequence opens up a window equal in length to the WCT within which you can update the registers. Writing a value other than the above mentioned sequence or if the sequence is longer than 20 bus cycles, resets the system or if IRQ_RST_EN is set, it interrupts and then resets the system). The unlock sequence is effective only if ALLOW_UPDATE is set.

### 17.6.2.9 Watchdog Timer Output Register High (WDOG\_TIMER\_OUT\_H)

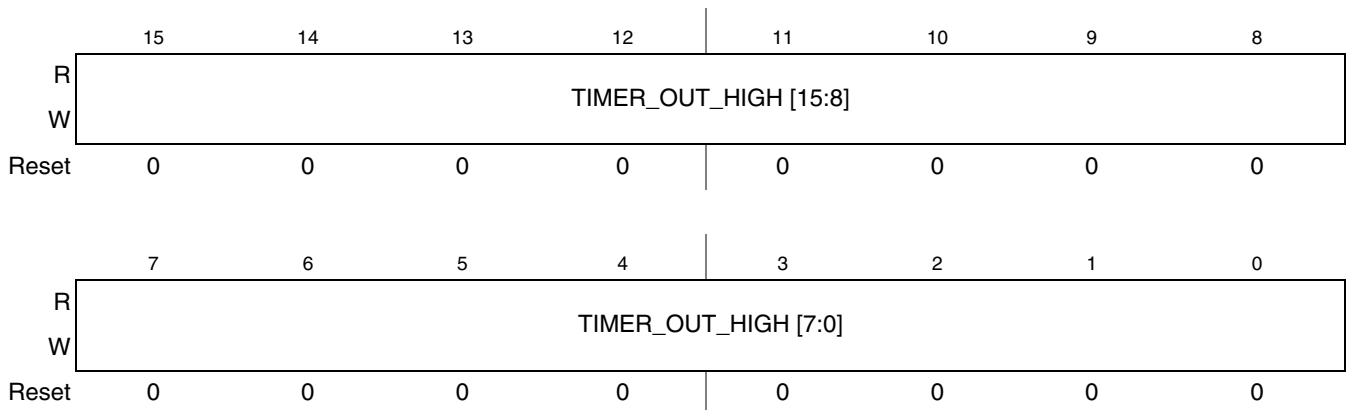


Figure 17-11. WDOG\_TIMER\_OUT\_H Register

Table 17-10. WDOG\_TIMER\_OUT\_H Register Field Descriptions

Field	Description
[15:0] TIMER_OUT_HIGH [15:0]	Shows the value of the upper 16 bits of the watchdog timer.

### 17.6.2.10 Watchdog Timer Output Register Low (WDOG\_TIMER\_OUT\_L)

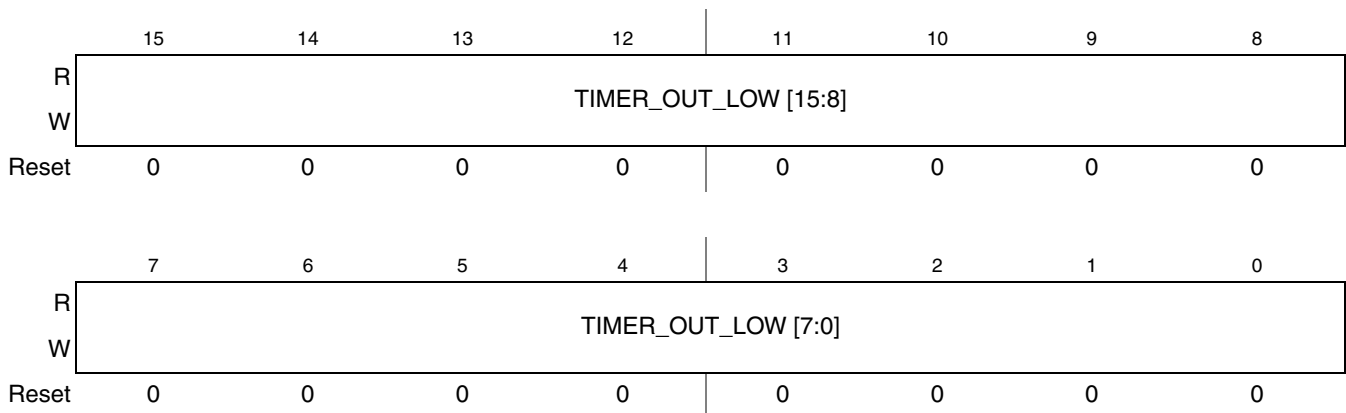


Figure 17-12. WDOG\_TIMER\_OUT\_L Register

Table 17-11. WDOG\_TIMER\_OUT\_L Register Field Descriptions

Field	Description
[15:0] TIMER_OUT_LOW [15:0]	Shows the value of the lower 16 bits of the watchdog timer.

During stop mode, the WDOG\_TIMER\_OUT will be caught at the pre-stop value of the watchdog timer. After exiting stop mode, a maximum delay of 1 WDOG\_CLK cycle + 3 bus clock cycles will occur before the WDOG\_TIMER\_OUT starts following the watchdog timer.

### 17.6.2.11 Watchdog Reset Count Register (WDOG\_RST\_CNT)

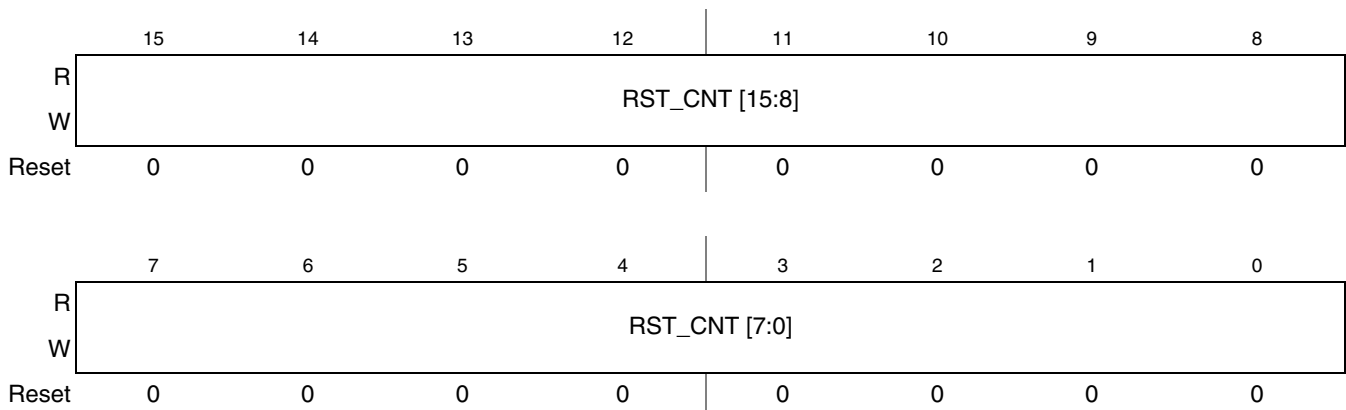


Figure 17-13. WDOG\_RST\_CNT Register

Table 17-12. WDOG\_RST\_CNT Register Field Descriptions

Field	Description
[15:0] RST_CNT [15:0]	Counts the number of times the watchdog resets the system. This register is reset only on a POR. Writing 1 to the bit to be cleared, enables you to clear the contents of this register

## 17.7 Watchdog Operation with 8-bit access

### 17.7.1 General Guideline

When performing 8-bit accesses to the watchdog's 16-bit registers where the intention is to access both the bytes of a register, you must try to place the two 8-bit accesses one after the other in your code.

### 17.7.2 Refresh and Unlock operations with 8-bit access

One exception condition that generates a reset to the system, is the write of any value other than those required for a legal refresh/update sequence to the respective refresh and unlock registers.

For an 8-bit access to these registers, writing a correct value requires at least two bus clock cycles that means there is an invalid value in the registers for one cycle. Therefore, the system is reset even if the intention is to write a correct value to the refresh/unlock register. Keeping this in mind the exception condition for 8-bit accesses is slightly modified. Whereas the match for a correct value for a refresh/unlock sequence is as per the original definition, the match for an incorrect value is done byte-wise on the refresh/unlock rather than for the whole 16-bit value. This means that if the high byte of the refresh/unlock register contains any value other than high bytes of the two values making up the sequence, it is treated as an exception condition, leading to a reset or interrupt-then-reset. The same holds true for the lower byte of the refresh or unlock register. Let us take the refresh operation that expects a write of 0xA602 followed by 0xB480 to the refresh register, as an example.

**Table 17-13. Refresh for 8-bit Access**

	WDOG_REFRESH[15:8]	WDOG_REFRESH[7:0]	Sequence value1 or value2 match	Mismatch exception
<b>Current Value</b>	0xB4	0x80	Value2 match	No
<b>Write 1</b>	0xB4	0x02	No match	No
<b>Write 2</b>	0xA6	0x02	Value1 match	No
<b>Write 3</b>	0xB4	0x02	No match	No
<b>Write 4</b>	0xB4	0x80	Value2 match. Sequence complete.	No
<b>Write 5</b>	0x02	0x80	No match	Yes

As shown in [Table 17-13](#), the refresh register holds its reset value initially. Thereafter, two 8-bit accesses are performed on the register to write the first value of the refresh sequence. No mismatch exception is registered on the intermediate write, Write1. The sequence is completed by performing two more 8-bit accesses, writing in the second value of the sequence for a successful refresh. It must be noted that the match of value2 takes place only when the complete 16-bit value is correctly written, write4. Hence, the requirement of writing value2 of the sequence within K bus clock cycles of value1 is checked by measuring the gap between write2 and write4.



It is reiterated that the condition for matching values 1 and 2 of the refresh or unlock sequence remains unchanged. It is just the criterion for detecting a wrong value in these registers which has been relaxed, as explained, for 8-bit accesses. Any 16-bit access still needs to adhere to the original guidelines, mentioned in the sections [Section 17.3.3, “Refreshing the Watchdog.”](#)

## 17.8 Restrictions on Watchdog Operation

This section mentions some exceptions to the watchdog operation that may not be apparent to you.

- Restriction on unlock / refresh operations—In the period between the closure of the WCT window (after unlock) and the actual reload of the watchdog timer, unlock and refresh operations need not be attempted.
- The update and reload of the watchdog timer happens two to three watchdog clocks after WCT window closes, following a successful configuration on unlock.
- Clock Switching Delay—The watchdog uses glitch free multiplexers at two places – one to choose between the LPO oscillator input and alternate clock input and the other to choose between the watchdog functional clock and fast clock input for watchdog functional test. A maximum time period of  $\sim 2$  clock A cycles plus  $\sim 2$  clock B cycles elapses from the time a switch is requested to the occurrence of the actual clock switch (clock A and B are the two input clocks to the clock mux).
- For the windowed mode, there is a two to three bus clock latency between the watchdog counter going past the window value and the same registering in the bus clock domain.
- For proper operation of the watchdog, the watchdog clock must be at least five times slower than the system bus clock at all times. An exception is the case when the watchdog clock is synchronous to the bus clock wherein the watchdog clock can be as fast as the bus clock.
- WCT must be equivalent to at least three watchdog clock cycles. If not ensured, this means that even after the close of the WCT window, you have to wait for the synchronized system reset to de-assert in the watchdog clock domain, before expecting the configuration updates to take effect.
- The time-out value of the watchdog should be set to a minimum of four watchdog clock cycles. This is to take into account the delay in new settings taking effect in the watchdog clock domain.
- You must take care not only to refresh the watchdog within the watchdog timer’s actual time-out period, but also provide enough allowance for the time it takes for the refresh sequence to be detected by the watchdog timer, on the watchdog clock.
- Updates cannot be made in the bus clock cycle immediately following the write of the unlock sequence, but one bus clock cycle later.
- It should be ensured that the time-out value for the watchdog is always greater than  $2 \times \text{WCT time} + 20$  bus clock cycles.
- An attempted refresh operation, in between the two writes of the unlock sequence and in the WCT time following a successful unlock, will go undetected.
- Trying to unlock the watchdog within the WCT time after an initial unlock has no effect.
- The refresh and unlock operations and interrupt are not automatically disabled in the watchdog functional test mode.
- After emerging from a reset due to a watchdog functional test, you are still expected to go through the mandatory steps of unlocking and configuring the watchdog. The watchdog continues to be in

its functional test mode and therefore you should pull the watchdog out of the functional test mode within WCT time of reset.

- After emerging from a reset due to a watchdog functional test, you still need to go through the mandatory steps of unlocking and configuring the watchdog.
- You must ensure that both the clock inputs to the glitchless clock multiplexers are alive during the switching of clocks. Failure to do so results in a loss of clock at their outputs.
- There is a gap of two to three watchdog clock cycles from the point that stop mode is entered to the watchdog timer actually pausing, due to synchronization. The same holds true for an exit from the stop mode, this time resulting in a two to three watchdog clock cycle delay in the timer restarting. In case the duration of the stop mode is less than one watchdog clock cycle, the watchdog timer is not guaranteed to pause.
- Consider the case when the first refresh value is written, following which the system enters stop mode (with system bus clk still on). Now, if the second refresh value is not written within 20 bus cycles of the first value, the system is reset (or interrupt-then-reset if enabled).

# Chapter 18

## External Watchdog Monitor (EWM)

### 18.1 Introduction

The watchdog is generally used to monitor the flow and execution of embedded software within an MCU. As with most embedded MCU's, the watchdog consists of a counter that if allowed to overflow, forces an internal reset (asynchronous) to all on-chip peripherals and optionally assert the  $\overline{\text{RESET}}$  pin to reset external devices/circuits. The overflow of the watchdog counter must not occur if the software code works well and services the watchdog to re-start the actual counter.

For safety, a redundant watchdog system, External Watchdog Monitor (EWM), is designed to monitor external circuits, as well as the MCU software flow. This provides a back-up mechanism to the internal watchdog that resets the MCU's CPU and peripherals.

The EWM differs from the internal watchdog in that it does not reset the MCU's CPU and peripherals. The EWM if allowed to time-out, provides an independent  $\overline{\text{EWM\_out}}$  pin that when asserted resets or places an external circuit into a safe mode. The CPU resets the EWM counter that is logically ANDed with an external digital input pin. This pin allows an external circuit to influence the  $\overline{\text{reset\_out}}$  signal.

#### 18.1.1 Features

Features of EWM module include:

- Independent 1 KHz LPO clock source
- Programmable time-out period specified in terms of number of EWM LPO clock cycles.
- Windowed refresh option
  - Provides robust check that program flow is faster than expected.
  - Programmable window.
  - Refresh outside window leads to assertion of  $\overline{\text{EWM\_out}}$ .
- Robust refresh mechanism
  - Write values of 0xB4 and 0x2C to EWM Refresh Register within 15 bus clock cycles.
- One output port,  $\overline{\text{EWM\_out}}$ , when asserted is used to reset or place the external circuit into safe mode.
- One Input port,  $\overline{\text{EWM\_in}}$ , allows an external circuit to control the  $\overline{\text{EWM\_out}}$  signal.

## 18.1.2 Modes of Operation

### 18.1.2.1 Stop Mode

When the EWM is in stop mode, the CPU services to the EWM cannot occur.

On entry to stop mode, the EWM is disabled, therefore the  $\overline{\text{EWM\_out}}$  pin accepts the condition set by the corresponding digital I/O module. For example, if  $\overline{\text{EWM\_out}}$  pin is left as digital input, it reaches high-impedance state. If the digital pin is configured as output, it continues to stay as push-pull output and output the value in the data register latch.

Therefore, if the EWM is forced into stop mode, the EWM function cannot be guaranteed to operate.

- On exit from stop mode through a reset, the EWM remains disabled.
- On exit from stop mode by an interrupt, the EWM is re-enabled, and the counter continues to be clocked from the same value prior to entry to stop mode.

If the EWM enters the stop mode during CPU service mechanism. At the exit from stop mode by an interrupt, refresh mechanism state machine starts from the previous state which means, if first service command is written correctly and EWM enters the stop mode immediately, the next command has to be written within next 15 CPU cycles after exiting from stop mode.

### 18.1.2.2 Wait Mode

EWM makes no difference between stop and wait mode. EWM functionality remains the same in both these modes.

### 18.1.2.3 Debug Mode

Entry to debug mode has no effect on the EWM.

- If the EWM is enabled prior to entry of debug mode, it remains enabled.
- If the EWM is disabled prior to entry of debug mode, it remains disabled.

### 18.1.3 Block Diagram

Figure 18-1 shows the EWM block diagram.

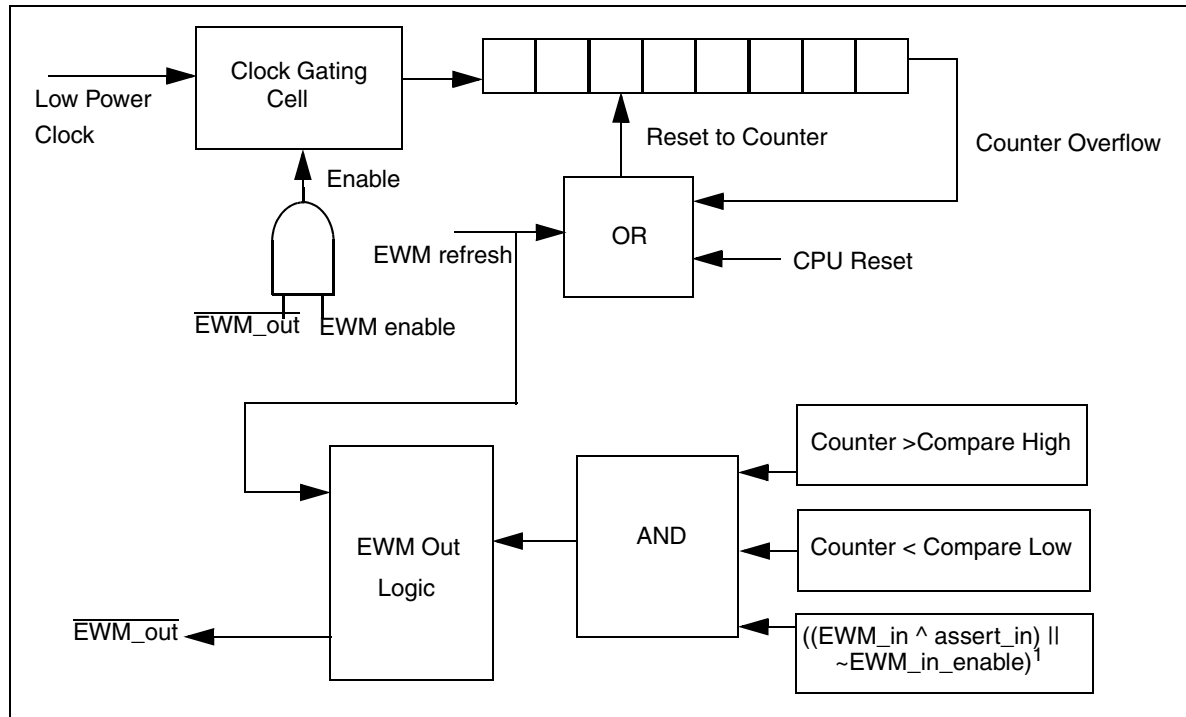


Figure 18-1. EWM Block Diagram

<sup>1</sup> Compare High > Counter > Compare Low

## 18.2 External Signal Description

The EWM has two external signal lines that are routed to I/O pads.

### 18.2.1 $\overline{\text{EWM\_out}}$

The  $\overline{\text{EWM\_out}}$  is a digital output signal used to gate an external circuit (application specific) that control safety critical functions. For example, the  $\overline{\text{EWM\_out}}$  is connected to the high voltage transistors circuits that control an ac motor in large appliance.

The  $\overline{\text{EWM\_out}}$  signal remains de-asserted when the EWM is being regularly serviced by the CPU within the programmable service window, indicating that the application code is executed as expected.

The  $\overline{\text{EWM\_out}}$  signal is asserted in any of the following conditions:

- Servicing the EWM when the counter value is less than EWMxCMPL value.
- If the EWM counter value reaches the EWMxCMPH value, and no EWM service has occurred.
- Servicing the EWM when the counter value is more than EWMxCMPL and less than EWMxCMPH values and EWM\_in signal is asserted.<sup>1</sup>

- After any reset.<sup>1</sup>

On a normal reset, the  $\overline{\text{EWM\_out}}$  is asserted. To de-assert the  $\overline{\text{EWM\_out}}$ , set EWMEN bit in the EWMxCTRL register to enable the EWM.

The assert level of the  $\overline{\text{EWM\_out}}$  is logic zero, and de-assert state is logic one. This assert level is fixed at design. If the  $\overline{\text{EWM\_out}}$  shares the PAD with a digital I/O pin, on reset this actual PAD defers to being an input signal. It takes the  $\overline{\text{EWM\_out}}$  output condition only after you enable the EWM by the EWMEN bit in the EWMxCTRL register.

Once the  $\overline{\text{EWM\_out}}$  pin is asserted, it can only be de-asserted by forcing a MCU reset.

**NOTE**

$\overline{\text{EWM\_out}}$  pad must be in pull down state when EWM functionality is used and when EWM is under Reset.

**18.2.2 EWM\_in**

The EWM\_in is a digital input signal that allows an external circuit to control the  $\overline{\text{EWM\_out}}$  signal. For example, in the application, an external circuit monitors a safety specific feature, and if there is fault with this circuit's behavior, it can then actively initiate the  $\overline{\text{EWM\_out}}$  signal that controls the gating circuit.

The EWM\_in signal is ignored if the EWM is disabled, or if INEN bit of EWMxCTRL register is de-asserted, as after any reset.

On enabling the EWM (setting EWMEN bit in the EWMxCTRL register) and enabling EWM\_in functionality (setting INEN bit in the EWMxCTRL register), the EWM\_in signal must be in the de-asserted state prior to the CPU servicing the EWM. It ensures that the  $\overline{\text{EWM\_out}}$  stays in the de-asserted state, otherwise the  $\overline{\text{EWM\_out}}$  pin is asserted.

**NOTE**

You must update the EWMxCMPH and EWMxCMPL registers prior to enabling the EWM. After enabling the EWM, the counter resets to zero, therefore providing a reasonable time after a power-on reset for the external monitoring circuit to stabilize and ensure that the EWM\_in pin is de-asserted.

**18.3 Memory Map**

**Table 18-1. EWM Memory Map**

Address offset	Register Name	Use	Recommended Access Type
0x00	EWMxCTRL	EWM Control Register	R/W, 8-bit, write once
0x01	EWMxSERV	EWM Service Register	W, 8-bit, write only
0x02	EWMxCMPL	EWM Compare Low Register	RW, 8-bit, write once

1. EWMxCMPL=EWMxCMPH is an invalid configuration for EWM.  
 1. This assertion is by the virtue of the external pull down device on the  $\overline{\text{EWM\_out}}$  pin.

Table 18-1. EWM Memory Map (continued)

Address offset	Register Name	Use	Recommended Access Type
0x03	EWMxCMPH	EWM Compare High Register	RW, 8-bit, write once

## 18.4 Register Definition

### 18.4.1 EWM Compare Low Register (EWMxCMPL)

The EWMxCMPL register is reset to zero after a CPU reset. This provide no minimum time for the CPU to service the EWM counter. You can write once after a CPU reset to the EWMxCMPL register. It is recommended that you always write to this register after a CPU reset, even if no minimum service time is required, to ensure that in the likelihood of code runaway, the EWM is not configured wrong.

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W								
Reset	0	0	0	0	0	0	0	0

All bits are write once after CPU reset.<sup>1</sup>

**Figure 18-2. EWMxCMPL Register**

<sup>1</sup> Writing the write once register more than once generates IPS transfer error.

### 18.4.2 EWM Compare High Register (EWMxCMPH)

The EWMxCMPH register is reset to 0xFF after a CPU reset. This provides a maximum of 256 clocks time, for the CPU to service the EWM counter. You can write once after a CPU reset to the EWMxCMPH register. It is recommended that you always write to this register after a CPU reset, even if maximum service time is required to ensure that in the likelihood of code runaway, the EWM is not configured wrong.

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W								
Reset	1	1	1	1	1	1	1	1

All bits are write once after CPU reset.

**Figure 18-3. EWMxCMPH Register**

#### NOTE

The valid values for EWMxCMPH are up to 0xFE because the EWM counter never expires when EWMxCMPH = 0xFF. The expiration happens only if EWM counter is greater than EWMxCMPH.

### 18.4.3 EWM Service Register (EWMxSERV)

The EWMxSERV register provides the interface from the CPU to the EWM module. It is a write only register and read of this register returns zero. The unique EWM service mechanism requires the CPU to write a first data byte of 0xB4, followed by write of second data byte of 0x2C within 15 CPU clock cycles to the EWMxSERV register .

The EWM service is illegal if the following conditions are true.

- The first or second data byte to EWMxSERV register is not written correctly.
- The second data byte is not written within 15bus cycles of the first data byte to the EWMxSERV register.

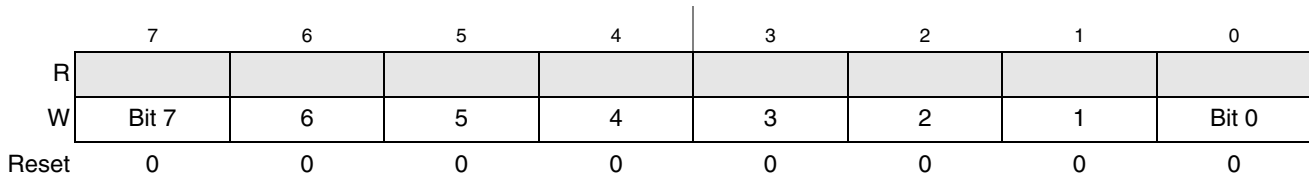
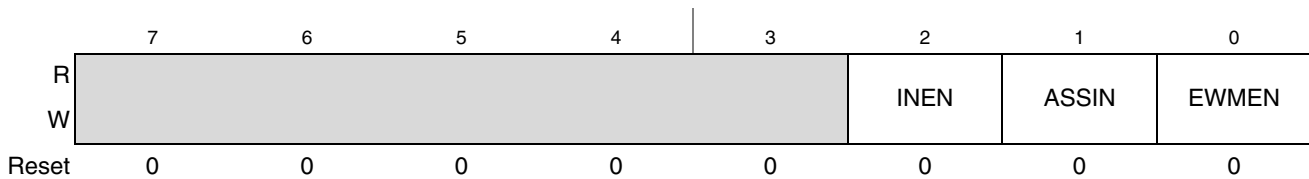


Figure 18-4. EWMxSERV Register

### 18.4.4 EWM Control Register (EWMxCTRL)

The EWMxCTRL register is cleared by any reset.

- EWMEN — This bit when set, enables the EWM module that resets the EWM counter to zero and de-asserts the  $\bar{EWM\_out}$  signal. Clearing EWMEN bit disables the EWM, and therefore it cannot be enabled until a reset occurs, due to the write once nature of this bit.
- ASSIN — Default assert of EWM\_in signal is logic zero. Setting ASSIN bit inverts the assert state to a logic one.
- INEN — This bit when set, enables the EWM\_in port.



All bits are write once after CPU reset.

= Unimplemented or Reserved

Figure 18-5. EWMxCTRL Register

## 18.5 Functional Description

The following sections describe the functional blocks that are used in the construction of EWM module.



### 18.5.1 EWM Counter

It is an 8-bit counter fed from a clock source that is independent of the CPU clock source. As the preferred time-out is between 1 ms and 100 ms the actual clock source should be in the KHz range. For S08 and ColdFire V1 MCUs, the clock source comes from the same 1 KHz RC oscillator that feeds the independent clocked watchdog.

The 8-bit ripple counter is reset to zero, after a CPU reset, or a EWM refresh cycle. The 8-bit ripple counter value is not accessible to the CPU.

### 18.5.2 EWM Compare Registers

The EWM 8-bit compare registers, EWMxCMPL and EWMxCMPH, are write once after a CPU reset and cannot be modified until another CPU reset occurs.

The EWM compare registers are used to create a service window, which is used by the CPU to service/refresh the EWM module.

- If the CPU services the EWM when the counter value lies between EWMxCMPL value and EWMxCMPH value, the counter is reset to zero. This is a legal service operation.
- If the CPU executes a EWM service/refresh action outside the legal service window,  $\overline{\text{EWM\_out}}$  is asserted.

It is illegal to program EWMxCMPL and EWMxCMPH with same value. In this case, as soon as counter reaches EWMxCMPL + 1,  $\overline{\text{EWM\_out}}$  is asserted.

### 18.5.3 EWM Refresh Mechanism

Other than the initial configuration of the EWM, the CPU can only access the EWM by the EWM Service Register. The CPU must access the EWM service register with correct write of unique data within the windowed time frame as determined by the EWMxCMPL and EWMxCMPH registers. Therefore, three possible conditions can occur:

**Table 18-2. EWM Refresh Mechanisms**

Condition	Mechanism
A unique EWM service occurs when $\text{EWMxCMPL} < \text{Counter} < \text{EWMxCMPH}$ .	The software behaves as expected and the counter of the EWM is reset to zero, and $\overline{\text{EWM\_out}}$ pin remains in the de-asserted state. <b>Note:</b> EWM_in pin is also assumed to be in the de-asserted state.
A unique EWM service occurs when $\text{Counter} < \text{EWMxCMPL}$	The software services the EWM and therefore resets the counter to zero and asserts the $\overline{\text{EWM\_out}}$ pin (irrespective of the EWM_in pin). The $\overline{\text{EWM\_out}}$ pin is expected to gate critical safety circuits.
Counter value reaches EWMxCMPH prior to a unique EWM service	The counter value reaches the EWMxCMPH value and no service of the EWM resets the counter to zero and assert the $\overline{\text{EWM\_out}}$ pin (irrespective of the EWM_in pin). The $\overline{\text{EWM\_out}}$ pin is expected to gate critical safety circuits.

Any Illegal service on EWM has no effect on  $\overline{\text{EWM\_out}}$ .

---

# Chapter 19

## Internal Clock Source (S08ICSV3)

### 19.1 Introduction

The internal clock source (ICS) module provides several clock source choices for this device. The module contains a frequency-locked loop (FLL) that is controllable by either an internal or an external reference clock. The module can select either of the FLL clocks, or either of the internal or external reference clocks as a source for the MCU system clock. The selected clock source is passed through a reduced bus divider that allows a lower output clock frequency to be derived. The ICS also controls a crystal oscillator (XOSC) that allows an external crystal, ceramic resonator, or another external clock source to produce the external reference clock.

## 19.1.1 Features

Key features of the ICS module are:

- Frequency-locked loop (FLL) is trimmable for accuracy
- Internal or external reference clocks can be used to control the FLL
- Reference divider is provided for external clock
- Internal reference clock has 9 trim bits available
- Internal or external reference clocks can be selected as the clock source for the MCU
- Whichever clock is selected as the source can be divided down
  - 2-bit select for clock divider is provided
    - Allowable dividers are: 1, 2, 4, 8
- Control signals for a low power oscillator clock generator (OSCOUT) as the ICS external reference clock are provided
  - HGO, RANGE, EREFS, ERCLKEN, EREFSTEN
- FLL Engaged Internal mode is automatically selected out of reset
- BDC clock is provided as a constant divide by 2 of the low range DCO output
- Three selectable digitally-controlled oscillators (DCO) optimized for different frequency ranges.
- Option to maximize output frequency for a 32768 Hz external reference clock source.

## 19.1.2 Block Diagram

Figure 19-1 is the ICS block diagram.

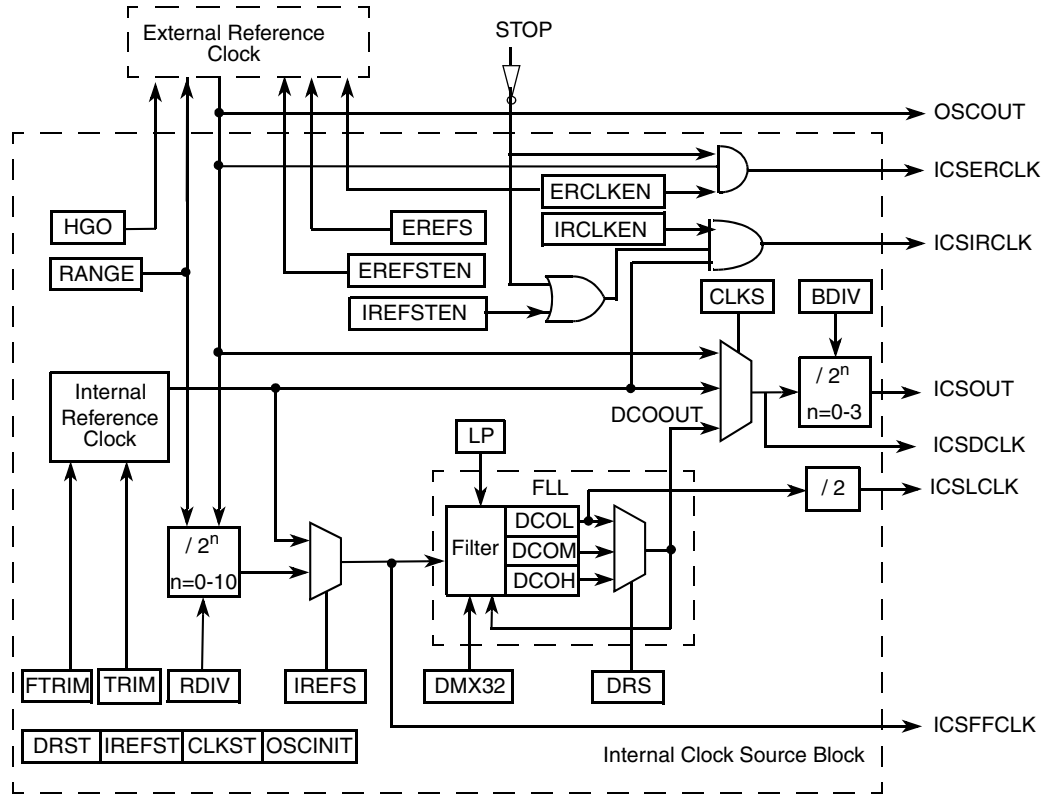


Figure 19-1. Internal Clock Source (ICS) Block Diagram

### 19.1.3 Modes of Operation

There are seven modes of operation for the ICS: FEI, FEE, FBI, FBILP, FBE, FBELP, and stop.

#### 19.1.3.1 FLL Engaged Internal (FEI)

In FLL engaged internal mode, which is the default mode, the ICS supplies a clock derived from the FLL which is controlled by the internal reference clock. The BDC clock is supplied from the FLL.

#### 19.1.3.2 FLL Engaged External (FEE)

In FLL engaged external mode, the ICS supplies a clock derived from the FLL which is controlled by an external reference clock source. The BDC clock is supplied from the FLL.

#### 19.1.3.3 FLL Bypassed Internal (FBI)

In FLL bypassed internal mode, the FLL is enabled and controlled by the internal reference clock, but is bypassed. The ICS supplies a clock derived from the internal reference clock. The BDC clock is supplied from the FLL.

### 19.1.3.4 FLL Bypassed Internal Low Power (FBILP)

In FLL bypassed internal low power mode, the FLL is disabled and bypassed, and the ICS supplies a clock derived from the internal reference clock. The BDC clock is not available.

### 19.1.3.5 FLL Bypassed External (FBE)

In FLL bypassed external mode, the FLL is enabled and controlled by an external reference clock, but is bypassed. The ICS supplies a clock derived from the external reference clock source. The BDC clock is supplied from the FLL.

### 19.1.3.6 FLL Bypassed External Low Power (FBELP)

In FLL bypassed external low power mode, the FLL is disabled and bypassed, and the ICS supplies a clock derived from the external reference clock. The BDC clock is not available.

### 19.1.3.7 Stop (STOP)

In stop mode, the FLL is disabled and the internal or the ICS external reference clocks source (OSCOUT) can be selected to be enabled or disabled. The BDC clock is not available and the ICS does not provide an MCU clock source.

#### NOTE

The DCO frequency changes from the pre-stop value to its reset value and the FLL will need to re-acquire the lock before the frequency is stable. Timing sensitive operations should wait for the FLL acquisition time,  $t_{Acquire}$ , before executing.

## 19.2 External Signal Description

There are no ICS signals that connect off chip.

## 19.3 Register Definition

Figure 19-1 is a summary of ICS registers.

Table 19-1. ICS Register Summary

Name		7	6	5	4	3	2	1	0
ICSC1	R	CLKS		RDIV			IREFS	IRCLKEN	IREFSTEN
	W								
ICSC2	R	BDIV		RANGE	HGO	LP	EREFS	ERCLKEN	EREFSTEN
	W								
ICSTRM	R	TRIM							
	W								

Table 19-1. ICS Register Summary (continued)

Name		7	6	5	4	3	2	1	0
ICSSC	R	DRST		DMX32	IREFST	CLKST		OSCINIT	FTRIM
	W	DRS							

### 19.3.1 ICS Control Register 1 (ICSC1)

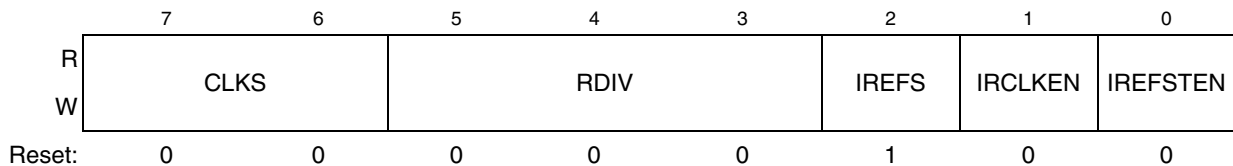


Figure 19-2. ICS Control Register 1 (ICSC1)

Table 19-2. ICS Control Register 1 Field Descriptions

Field	Description
7:6 CLKS	<b>Clock Source Select</b> — Selects the clock source that controls the bus frequency. The actual bus frequency depends on the value of the BDIV bits. 00 Output of FLL is selected. 01 Internal reference clock is selected. 10 External reference clock is selected. 11 Reserved, defaults to 00.
5:3 RDIV	<b>Reference Divider</b> — Selects the amount to divide down the external reference clock. Resulting frequency must be in the range 31.25 kHz to 39.0625 kHz. See <a href="#">Table 19-3</a> for the divide-by factors.
2 IREFS	<b>Internal Reference Select</b> — The IREFS bit selects the reference clock source for the FLL. 1 Internal reference clock selected. 0 External reference clock selected.
1 IRCLKEN	<b>Internal Reference Clock Enable</b> — The IRCLKEN bit enables the internal reference clock for use as ICSIRCLK. 1 ICSIRCLK active. 0 ICSIRCLK inactive.
0 IREFSTEN	<b>Internal Reference Stop Enable</b> — The IREFSTEN bit controls whether or not the internal reference clock remains enabled when the ICS enters stop mode. 1 Internal reference clock stays enabled in stop if IRCLKEN is set before entering stop. 0 Internal reference clock is disabled in stop.

Table 19-3. Reference Divide Factor

RDIV	RANGE=0	RANGE=1
0	1 <sup>1</sup>	32
1	2	64
2	4	128
3	8	256

**Table 19-3. Reference Divide Factor**

<b>RDIV</b>	<b>RANGE=0</b>	<b>RANGE=1</b>
<b>4</b>	16	512
<b>5</b>	32	1024
<b>6</b>	64	Reserved
<b>7</b>	128	Reserved

<sup>1</sup> Reset default

## 19.3.2 ICS Control Register 2 (ICSC2)

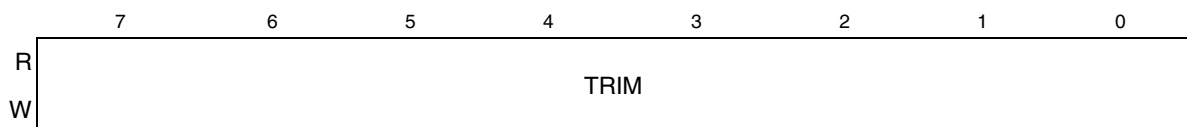


Figure 19-3. ICS Control Register 2 (ICSC2)

Table 19-4. ICS Control Register 2 Field Descriptions

Field	Description
7:6 BDIV	<b>Bus Frequency Divider</b> — Selects the amount to divide down the clock source selected by the CLKS bits. This controls the bus frequency. 00 Encoding 0 — Divides selected clock by 1. 01 Encoding 1 — Divides selected clock by 2 (reset default). 10 Encoding 2 — Divides selected clock by 4. 11 Encoding 3 — Divides selected clock by 8.
5 RANGE	<b>Frequency Range Select</b> — Selects the frequency range for the external oscillator. 1 High frequency range selected for the external oscillator. 0 Low frequency range selected for the external oscillator.
4 HGO	<b>High Gain Oscillator Select</b> — The HGO bit controls the external oscillator mode of operation. 1 Configure external oscillator for high gain operation. 0 Configure external oscillator for low power operation.
3 LP	<b>Low Power Select</b> — The LP bit controls whether the FLL is disabled in FLL bypassed modes. 1 FLL is disabled in bypass modes unless BDM is active. 0 FLL is not disabled in bypass mode.
2 EREFs	<b>External Reference Select</b> — The EREFs bit selects the source for the external reference clock. 1 Oscillator requested. 0 External Clock Source requested.
1 ERCLKEN	<b>External Reference Enable</b> — The ERCLKEN bit enables the external reference clock for use as ICsERCLK. 1 ICsERCLK active. 0 ICsERCLK inactive.
0 EREFSTEN	<b>External Reference Stop Enable</b> — The EREFSTEN bit controls whether or not the external reference clock source (OSCOUT) remains enabled when the ICS enters stop mode. 1 External reference clock source stays enabled in stop if ERCLKEN is set before entering stop. 0 External reference clock source is disabled in stop.

## 19.3.3 ICS Trim Register (ICSTRM)



Reset: Note: TRIM is loaded during reset from a factory programmed location when not in BDM mode. If in a BDM mode, a default value of 0x80 is loaded.

Figure 19-4. ICS Trim Register (ICSTRM)



Table 19-5. ICS Trim Register Field Descriptions

Field	Description
7:0 TRIM	<p><b>ICS Trim Setting</b> — The TRIM bits control the internal reference clock frequency by controlling the internal reference clock period. The bits' effect are binary weighted (in other words, bit 1 adjusts twice as much as bit 0). Increasing the binary value in TRIM will increase the period, and decreasing the value will decrease the period.</p> <p>An additional fine trim bit is available in ICSSC as the FTRIM bit.</p>

### 19.3.4 ICS Status and Control (ICSSC)

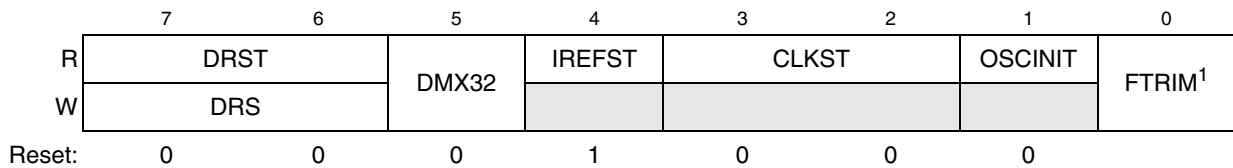


Figure 19-5. ICS Status and Control Register (ICSSC)

<sup>1</sup> FTRIM is loaded during reset from a factory programmed location when not in any BDM mode. If in a BDM mode, FTRIM gets loaded with a value of 1'b0.

Table 19-6. ICS Status and Control Register Field Descriptions

Field	Description
7-6 DRST DRS	<p><b>DCO Range Status</b> — The DRST read field indicates the current frequency range for the FLL output, DCOOUT. See <a href="#">Table 19-7</a>. The DRST field does not update immediately after a write to the DRS field due to internal synchronization between clock domains. Writing the DRS bits to 2'b11 is ignored and the DRST bits remain with the current setting.</p> <p><b>DCO Range Select</b> — The DRS field selects the frequency range for the FLL output, DCOOUT. Writes to the DRS field while the LP bit is set are ignored.</p> <p>00 Low range. 01 Mid range. 10 High range. 11 Reserved.</p>
5 DMX32	<p><b>DCO Maximum frequency with 32.768 kHz reference</b> — The DMX32 bit controls whether or not the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference. See <a href="#">Table 19-7</a>.</p> <p>0 DCO has default range of 25%. 1 DCO is fined tuned for maximum frequency with 32.768 kHz reference.</p>
4 IREFST	<p><b>Internal Reference Status</b> — The IREFST bit indicates the current source for the reference clock. The IREFST bit does not update immediately after a write to the IREFS bit due to internal synchronization between clock domains.</p> <p>0 Source of reference clock is external clock. 1 Source of reference clock is internal clock.</p>

Table 19-6. ICS Status and Control Register Field Descriptions (continued)

Field	Description
3-2 CLKST	<b>Clock Mode Status</b> — The CLKST bits indicate the current clock mode. The CLKST bits don't update immediately after a write to the CLKS bits due to internal synchronization between clock domains. 00 Output of FLL is selected. 01 FLL Bypassed, Internal reference clock is selected. 10 FLL Bypassed, External reference clock is selected. 11 Reserved.
1 OSCINIT	<b>OSC Initialization</b> — If the external reference clock is selected by ERCLKEN or by the ICS being in FEE, FBE, or FBELP mode, and if EREFS is set, then this bit is set after the initialization cycles of the external oscillator clock have completed. This bit is only cleared when either ERCLKEN or EREFS are cleared.
0 FTRIM	<b>ICS Fine Trim</b> — The FTRIM bit controls the smallest adjustment of the internal reference clock frequency. Setting FTRIM will increase the period and clearing FTRIM will decrease the period by the smallest amount possible.

Table 19-7. DCO frequency range<sup>1</sup>

DRS	DMX32	Reference range	FLL factor	DCO range
00	0	31.25 - 39.0625 kHz	512	16 - 20 MHz
	1	32.768 kHz	608	19.92 MHz
01	0	31.25 - 39.0625 kHz	1024	32 - 40 MHz
	1	32.768 kHz	1216	39.85 MHz
10	0	31.25 - 39.0625 kHz	1536	48 - 60 MHz
	1	32.768 kHz	1824	59.77 MHz
11	Reserved			

<sup>1</sup> The resulting bus clock frequency should not exceed the maximum specified bus clock frequency of the device.

## 19.4 Functional Description

### 19.4.1 Operational Modes

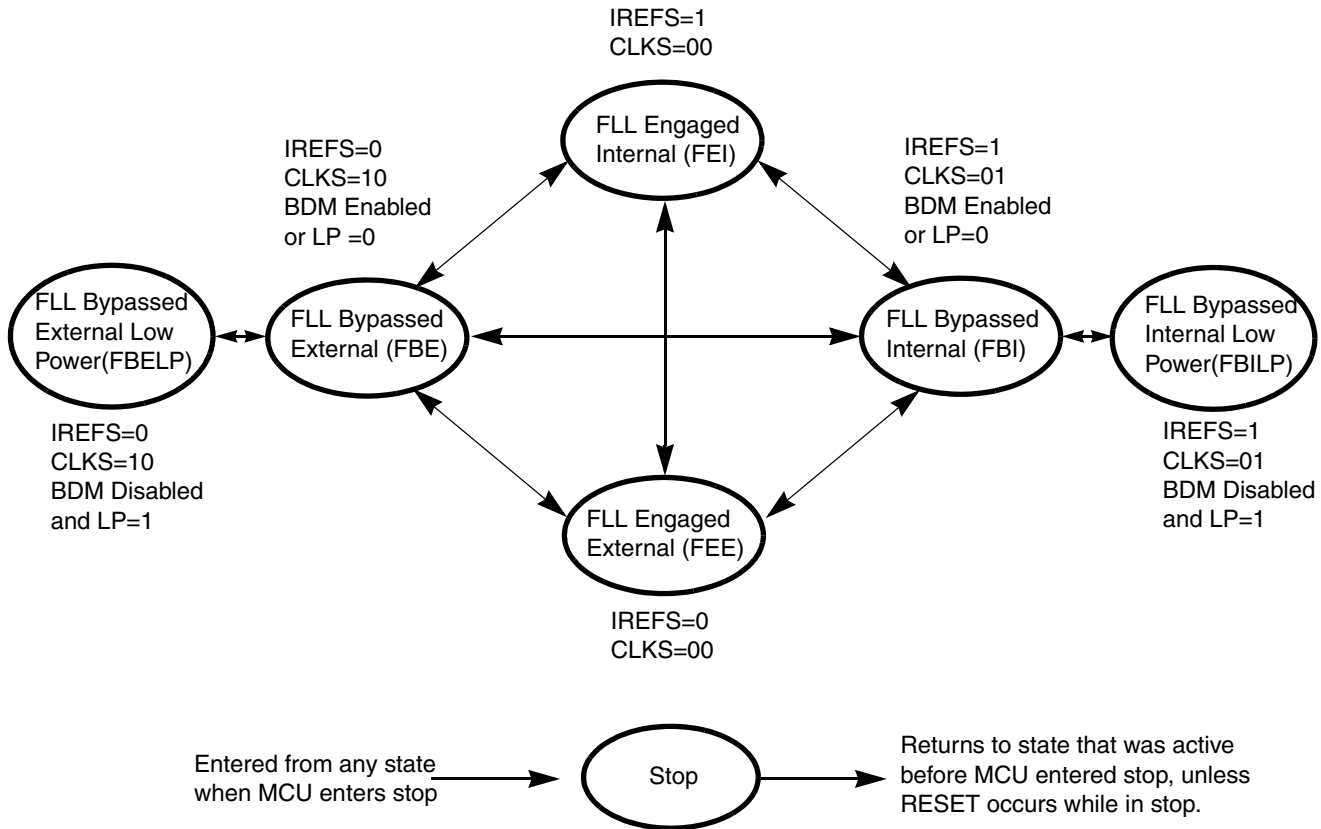


Figure 19-6. Clock Switching Modes

The seven states of the ICS are shown as a state diagram and are described below. The arrows indicate the allowed movements between the states.

#### 19.4.1.1 FLL Engaged Internal (FEI)

FLL engaged internal (FEI) is the default mode of operation and is entered when all the following conditions occur:

- CLKS bits are written to 00.
- IREFS bit is written to 1.

In FLL engaged internal mode, the ICSOUT clock is derived from the FLL clock, which is controlled by the internal reference clock. The FLL loop locks the frequency to the FLL factor times the internal reference frequency. The ICSLCLK is available for BDC communications, and the internal reference clock is enabled.

### 19.4.1.2 FLL Engaged External (FEE)

The FLL engaged external (FEE) mode is entered when all the following conditions occur:

- CLKS bits are written to 00.
- IREFS bit is written to 0.
- RDIV bits are written to divide external reference clock to be within the range of 31.25 kHz to 39.0625 kHz.

In FLL engaged external mode, the ICSOUT clock is derived from the FLL clock which is controlled by the external reference clock source. The FLL loop locks the frequency to the FLL factor times the external reference frequency, as selected by the RDIV bits. The ICSLCLK is available for BDC communications, and the external reference clock is enabled.

### 19.4.1.3 FLL Bypassed Internal (FBI)

The FLL bypassed internal (FBI) mode is entered when all the following conditions occur:

- CLKS bits are written to 01.
- IREFS bit is written to 1.
- BDM mode is active or LP bit is written to 0.

In FLL bypassed internal mode, the ICSOUT clock is derived from the internal reference clock. The FLL clock is controlled by the internal reference clock, and the FLL loop locks the FLL frequency to the FLL factor times the internal reference frequency. The ICSLCLK will be available for BDC communications, and the internal reference clock is enabled.

### 19.4.1.4 FLL Bypassed Internal Low Power (FBILP)

The FLL bypassed internal low power (FBILP) mode is entered when all the following conditions occur:

- CLKS bits are written to 01.
- IREFS bit is written to 1.
- BDM mode is not active and LP bit is written to 1.

In FLL bypassed internal low power mode, the ICSOUT clock is derived from the internal reference clock and the FLL is disabled. The ICSLCLK will be not be available for BDC communications, and the internal reference clock is enabled.

### 19.4.1.5 FLL Bypassed External (FBE)

The FLL bypassed external (FBE) mode is entered when all the following conditions occur:

- CLKS bits are written to 10.
- IREFS bit is written to 0.
- RDIV bits are written to divide external reference clock to be within the range of 31.25 kHz to 39.0625 kHz.
- BDM mode is active or LP bit is written to 0.

In FLL bypassed external mode, the ICSOUT clock is derived from the external reference clock source. The FLL clock is controlled by the external reference clock, and the FLL loop locks the FLL frequency to the FLL factor times the external reference frequency, as selected by the RDIV bits, so that the ICSLCLK will be available for BDC communications, and the external reference clock is enabled.

#### 19.4.1.6 FLL Bypassed External Low Power (FBELP)

The FLL bypassed external low power (FBELP) mode is entered when all the following conditions occur:

- CLKS bits are written to 10.
- IREFS bit is written to 0.
- BDM mode is not active and LP bit is written to 1.

In FLL bypassed external low power mode, the ICSOUT clock is derived from the external reference clock source and the FLL is disabled. The ICSLCLK will be not be available for BDC communications. The external reference clock source is enabled.

#### 19.4.1.7 Stop

Stop mode is entered whenever the MCU enters a STOP state. In this mode, all ICS clock signals are static except in the following cases:

ICSIRCLK will be active in stop mode when all the following conditions occur:

- IRCLKEN bit is written to 1.
- IREFSTEN bit is written to 1.

OSCOUT will be active in stop mode when all the following conditions occur:

- ERCLKEN bit is written to 1.
- EREFSTEN bit is written to 1.

### 19.4.2 Mode Switching

The IREF bit can be changed at anytime, but the actual switch to the newly selected clock is shown by the IREFST bit. When switching between FLL engaged internal (FEI) and FLL engaged external (FEE) modes, the FLL begins locking again after the switch is completed.

The CLKS bits can also be changed at anytime, but the actual switch to the newly selected clock is shown by the CLKST bits. If the newly selected clock is not available, the previous clock remains selected.

The DRS bits can be changed at anytime except when LP bit is 1. If the DRS bits are changed while in FLL engaged internal (FEI) or FLL engaged external (FEE), the bus clock remains at the previous DCO range until the new DCO starts. When the new DCO starts the bus clock switches to it. After switching to the new DCO the FLL remains unlocked for several reference cycles. Once the selected DCO startup time is over, the FLL is locked. The completion of the switch is shown by the DRST bits.

### 19.4.3 Bus Frequency Divider

The BDIV bits can be changed at anytime and the actual switch to the new frequency occurs immediately.

### 19.4.4 Low Power Bit Usage

The low power bit (LP) is provided to allow the FLL to be disabled and thus conserve power when it is not being used. The DRS bits can not be written while LP bit is 1.

However, in some applications it may be desirable to allow the FLL to be enabled and to lock for maximum accuracy before switching to an FLL engaged mode. To do this, write the LP bit to 0.

### 19.4.5 DCO Maximum Frequency with 32.768 kHz Oscillator

The FLL has an option to change the clock multiplier for the selected DCO range such that it results in the maximum bus frequency with a common 32.768 kHz crystal reference clock.

### 19.4.6 Internal Reference Clock

When IRCLKEN is set the internal reference clock signal is presented as ICSIRCLK, which can be used as an additional clock source. To re-target the ICSIRCLK frequency, write a new value to the TRIM bits in the ICSTRM register to trim the period of the internal reference clock:

- Writing a larger value slows down the ICSIRCLK frequency.
- Writing a smaller value to the ICSTRM register speeds up the ICSIRCLK frequency.

The TRIM bits effect the ICSOUT frequency if the ICS is in FLL engaged internal (FEI), FLL bypassed internal (FBI), or FLL bypassed internal low power (FBILP) mode.

Until ICSIRCLK is trimmed, programming low reference divider (RDIV) factors may result in ICSOUT frequencies that exceed the maximum chip-level frequency and violate the chip-level clock timing specifications (see the [Device Overview](#) chapter).

If IREFSTEN is set and the IRCLKEN bit is written to 1, the internal reference clock keeps running during stop mode in order to provide a fast recovery upon exiting stop.

All MCU devices are factory programmed with a trim value in a reserved memory location. This value is uploaded to the ICSTRM register and ICS FTRIM register during any reset initialization. For finer precision, trim the internal oscillator in the application and set the FTRIM bit accordingly.

## 19.4.7 External Reference Clock

The ICS module supports an external reference clock with frequencies between 31.25 kHz to 40 MHz in all modes. When the ERCLKEN is set, the external reference clock signal is presented as ICSECLK, which can be used as an additional clock source in run mode. When IREFS = 1, the external reference clock is not used by the FLL and will only be used as ICSECLK. In these modes, the frequency can be equal to the maximum frequency the chip-level timing specifications support (see the [Device Overview](#) chapter).

If EREFSTEN is set and the ERCLKEN bit is written to 1, the external reference clock source (OSCOUT) keeps running during stop mode in order to provide a fast recovery upon exiting stop.

## 19.4.8 Fixed Frequency Clock

The ICS presents the divided FLL reference clock as ICSFFCLK for use as an additional clock source. ICSFFCLK frequency must be no more than 1/4 of the ICSOUT frequency to be valid. Because of this requirement, in bypass modes the ICSFFCLK is valid only in bypass external modes (FBE and FBELP) for the following combinations of BDIV, RDIV and RANGE values:

- RANGE=1
- BDIV=00 (divide by 1), RDIV  $\geq$  010
- BDIV=01 (divide by 2), RDIV  $\geq$  011
- BDIV=10 (divide by 4), RDIV  $\geq$  100
- BDIV=11 (divide by 8), RDIV  $\geq$  101

## 19.4.9 Local Clock

The ICS presents the low range DCO output clock divided by two as ICSLCLK for use as a clock source for BDC communications. ICSLCLK is not available in FLL bypassed internal low power (FBILP) and FLL bypassed external low power (FBELP) modes.

# Chapter 20

## Rapid GPIO (RGPIO)

### 20.1 Introduction

The Rapid GPIO (RGPIO) module provides a 16-bit general-purpose I/O module directly connected to the processor's high-speed 32-bit local bus. This connection plus support for single-cycle, zero wait-state data transfers allows the RGPIO module to provide improved pin performance when compared to more traditional GPIO modules located on the internal slave peripheral bus.

Many of the pins associated with a device may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. The definition of the exact pin functions and the affected signals is specific to each device. Every GPIO port, including the RGPIO module, has registers that configure, monitor, and control the port pins.

#### 20.1.1 Overview

The RGPIO module provides 16-bits of high-speed GPIO functionality, mapped to the processor's bus. The key features of this module include:

- 16 bits of high-speed GPIO functionality connected to the processor's local 32-bit bus
- Memory-mapped device connected to the ColdFire core's local bus
  - Support for all access sizes: byte, word, and longword
  - All reads and writes complete in a single data phase cycle for zero wait-state response
- Data bits can be accessed directly or via alternate addresses to provide set, clear, and toggle functions
  - Alternate addresses allow set, clear, toggle functions using simple store operations without the need for read-modify-write references
- Unique data direction and pin enable control registers
- Package pin toggle rates typically 1.5–3.5x faster than comparable pin mapped onto peripheral bus

A simplified block diagram of the RGPIO module is shown in [Figure 20-1](#). The details of the pin muxing and pad logic are device-specific.



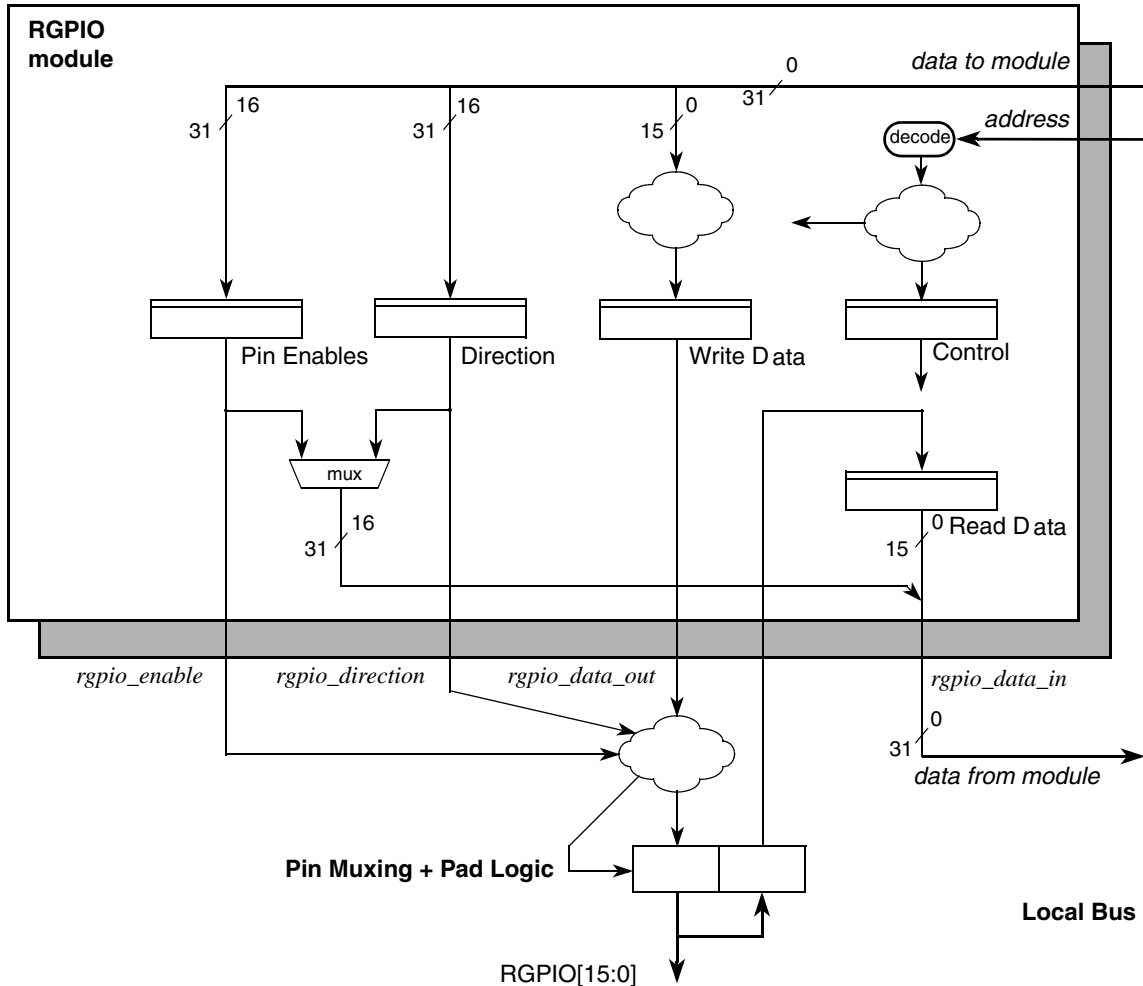


Figure 20-1. RGPIO Block Diagram

## 20.1.2 Features

The major features of the RGPIO module providing 16 bits of high-speed general-purpose input/output are:

- Small memory-mapped device connected to the processor's local bus
  - All memory references complete in a single cycle to provide zero wait-state responses
  - Located in processor's high-speed clock domain
- Simple programming model
  - Four 16-bit registers, mapped as three program-visible locations
    - Register for pin enables
    - Register for controlling the pin data direction
    - Register for storing output pin data
    - Register for reading current pin state

- The two data registers (read, write) are mapped to a single program-visible location
- Alternate addresses to perform data set, clear, and toggle functions using simple writes
- Separate read and write programming model views enable simplified driver software
  - Support for any access size (byte, word, or longword)

### 20.1.3 Modes of Operation

The RGPIO module does not support any special modes of operation. As a memory-mapped device located on the processor's high-speed local bus, it responds based strictly on memory address and does not consider the operating mode (supervisor, user) of its references.

## 20.2 External Signal Description

### 20.2.1 Overview

As shown in [Figure 20-1](#), the RGPIO module's interface to external logic is indirect via the device pin-muxing and pad logic. For a list of the associated RGPIO input/output signals, see [Table 20-1](#).

**Table 20-1. RGPIO Module External I/O Signals**

Signal Name	Type	Description
RGPIO[15:0]	I/O	RGPIO Data Input/Output

### 20.2.2 Detailed Signal Descriptions

[Table 20-2](#) provides descriptions of the RGPIO module's input and output signals.

**Table 20-2. RGPIO Detailed Signal Descriptions**

Signal	I/O	Description						
RGPIO[15:0]	I/O	Data Input/Output. When configured as an input, the state of this signal is reflected in the read data register. When configured as an output, this signal is the output of the write data register.						
		<table border="1"> <thead> <tr> <th>State Meaning</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Asserted—</td> <td>Input: Indicates the RGPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven high.</td> </tr> <tr> <td>Negated—</td> <td>Input: Indicates the RGPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven low.</td> </tr> </tbody> </table>	State Meaning	Description	Asserted—	Input: Indicates the RGPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven high.	Negated—	Input: Indicates the RGPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven low.
		State Meaning	Description					
Asserted—	Input: Indicates the RGPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven high.							
Negated—	Input: Indicates the RGPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven low.							
<table border="1"> <thead> <tr> <th>Timing</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Assertion/Negation—</td> <td>Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.</td> </tr> </tbody> </table>	Timing	Description	Assertion/Negation—	Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.				
Timing	Description							
Assertion/Negation—	Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.							

## 20.3 Memory Map/Register Definition

The RGPIO module provides a compact 16-byte programming model based at a system memory address of 0x(00)C0\_0000 (noted as RGPIO\_BASE throughout the chapter). As previously noted, the programming model views are different between reads and writes as this enables simplified software for manipulation of the RGPIO pins.

Additionally, the RGPIO programming model is defined with a 32-bit organization. The basic size of each program-visible register is 16 bits, but the programming model may be referenced using byte (8-bit), word (16-bit) or longword (32-bit) accesses. Performance is typically maximized using 32-bit accesses.

### NOTE

Writes to the two-byte fields at RGPIO\_BASE + 0x8 and RGPIO\_BASE + 0xC are allowed, but do not affect any program-visible register within the RGPIO module.

**Table 20-3. RGPIO Write Memory Map**

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	16	W	0x0000	<a href="#">20.3.1/20-5</a>
0x02	RGPIO Write Data Register (RGPIO_DATA)	16	W	0x0000	<a href="#">20.3.2/20-5</a>
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	16	W	0x0000	<a href="#">20.3.3/20-6</a>
0x06	RGPIO Write Data Clear Register (RGPIO_CLR)	16	W	N/A	<a href="#">20.3.4/20-6</a>
0x0A	RGPIO Write Data Set Register (RGPIO_SET)	16	W	N/A	<a href="#">20.3.5/20-7</a>
0x0E	RGPIO Write Data Toggle Register (RGPIO_TOG)	16	W	N/A	<a href="#">20.3.6/20-7</a>

**Table 20-4. RGPIO Read Memory Map**

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">20.3.1/20-5</a>
0x02	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">20.3.2/20-5</a>
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	16	R	0x0000	<a href="#">20.3.3/20-6</a>
0x06	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">20.3.2/20-5</a>
0x08	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">20.3.1/20-5</a>
0x0A	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">20.3.2/20-5</a>
0x0C	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0x0000	<a href="#">20.3.1/20-5</a>
0x0E	RGPIO Write Data Register (RGPIO_DATA)	16	R	0x0000	<a href="#">20.3.2/20-5</a>

### 20.3.1 RGPIO Data Direction (RGPIO\_DIR)

The read/write RGPIO\_DIR register defines whether a properly-enabled RGPIO pin is configured as an input or output:

- Setting any bit in RGPIO\_DIR configures a properly-enabled RGPIO port pin as an output
- Clearing any bit in RGPIO\_DIR configures a properly-enabled RGPIO port pin as an input

At reset, all bits in the RGPIO\_DIR are cleared.

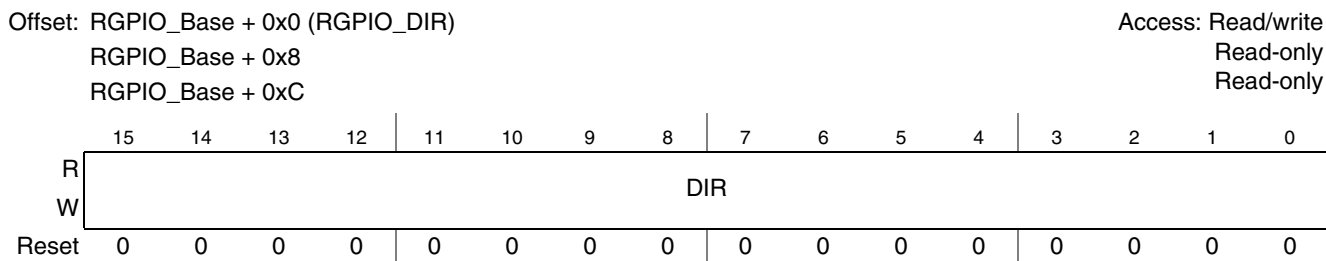


Figure 20-2. RGPIO Data Direction Register (RGPIO\_DIR)

Table 20-5. RGPIO\_DIR Field Descriptions

Field	Description
15–0 DIR	Data direction. 0 A properly-enabled RGPIO pin is configured as an input 1 A properly-enabled RGPIO pin is configured as an output

### 20.3.2 RGPIO Data (RGPIO\_DATA)

The RGPIO\_DATA register specifies the write data for a properly-enabled RGPIO output pin or the sampled read data value for a properly-enabled input pin. An attempted read of the RGPIO\_DATA register returns undefined data for disabled pins, since the data value is dependent on the device-level pin muxing and pad implementation. The RGPIO\_DATA register is read/write. At reset, all bits in the RGPIO\_DATA registers are cleared.

To set bits in a RGPIO\_DATA register, directly set the RGPIO\_DATA bits or set the corresponding bits in the RGPIO\_SET register. To clear bits in the RGPIO\_DATA register, directly clear the RGPIO\_DATA bits, or clear the corresponding bits in the RGPIO\_CLR register. Setting a bit in the RGPIO\_TOG register inverts (toggles) the state of the corresponding bit in the RGPIO\_DATA register.

## Rapid GPIO (RGPIO)

Offset: RGPIO\_Base + 0x2 (RGPIO\_DATA)  
 RGPIO\_Base + 0x6  
 RGPIO\_Base + 0xA  
 RGPIO\_Base + 0xE

Access: Read/write  
 Read/Indirect Write  
 Read/Indirect Write  
 Read/Indirect Write

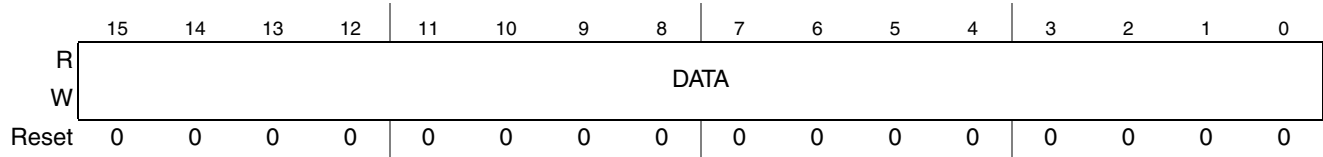


Figure 20-3. RGPIO Data Register (RGPIO\_DATA)

Table 20-6. RGPIO\_DATA Field Descriptions

Field	Description
15–0 DATA	RGPIO data. 0 A properly-enabled RGPIO output pin is driven with a logic 0, or a properly-enabled RGPIO input pin was read as a logic 0 1 A properly-enabled RGPIO output pin is driven with a logic 1, or a properly-enabled RGPIO input pin was read as a logic 1

### 20.3.3 RGPIO Pin Enable (RGPIO\_ENB)

The RGPIO\_ENB register configures the corresponding package pin as a RGPIO pin instead of the normal GPIO pin mapped onto the peripheral bus.

The RGPIO\_ENB register is read/write. At reset, all bits in the RGPIO\_ENB are cleared, disabling the RGPIO functionality.

Offset: RGPIO\_Base + 0x4 (RGPIO\_ENB)

Access: Read/write

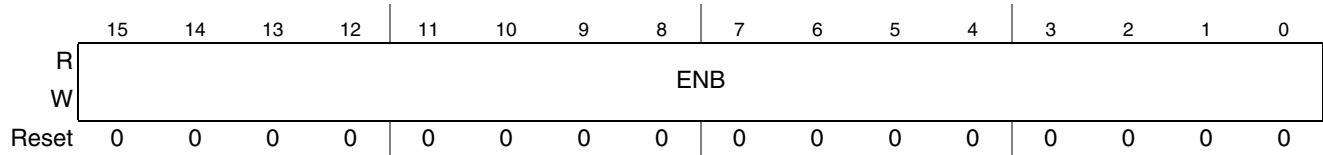


Figure 20-4. RGPIO Enable Register (RGPIO\_ENB)

Table 20-7. RGPIO\_ENB Field Descriptions

Field	Description
15–0 ENB	Enable pin for RGPIO 0 The corresponding package pin is configured for use as a normal GPIO pin, not a RGPIO 1 The corresponding package pin is configured for use as a RGPIO pin

### 20.3.4 RGPIO Clear Data (RGPIO\_CLR)

The RGPIO\_CLR register provides a mechanism to clear specific bits in the RGPIO\_DATA by performing a simple write. Clearing a bit in RGPIO\_CLR clears the corresponding bit in the RGPIO\_DATA register.

Setting it has no effect. The RGPIO\_CLR register is write-only; reads of this address return the RGPIO\_DATA register.

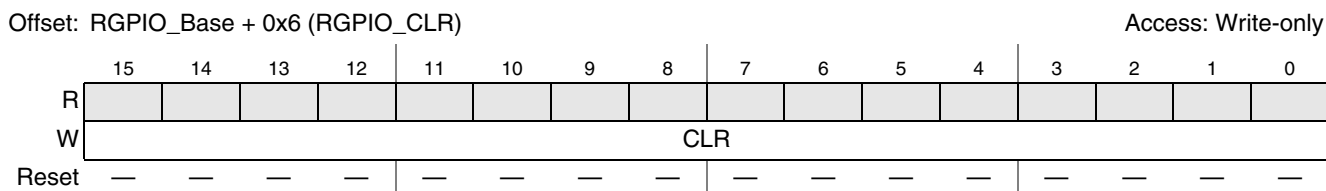


Figure 20-5. RGPIO Clear Data Register (RGPIO\_CLR)

Table 20-8. RGPIO\_CLR Field Descriptions

Field	Description
15–0 CLR	Clear data bit 0 Clears the corresponding bit in the RGPIO_DATA register 1 No effect

### 20.3.5 RGPIO Set Data (RGPIO\_SET)

The RGPIO\_SET register provides a mechanism to set specific bits in the RGPIO\_DATA register by performing a simple write. Setting a bit in RGPIO\_SET asserts the corresponding bit in the RGPIO\_DATA register. Clearing it has no effect. The RGPIO\_SET register is write-only; reads of this address return the RGPIO\_DATA register.

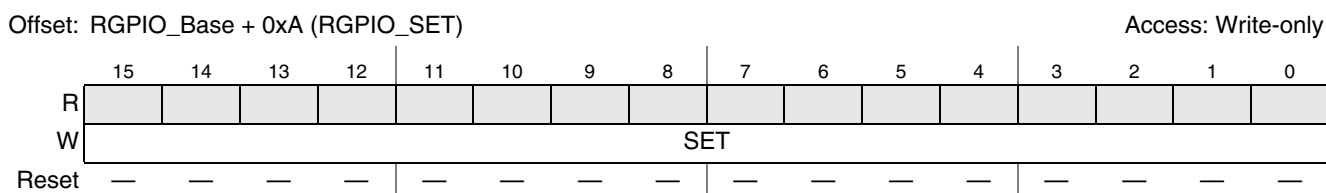


Figure 20-6. RGPIO Set Data Register (RGPIO\_SET)

Table 20-9. RGPIO\_SET Field Descriptions

Field	Description
15–0 SET	Set data bit 0 No effect 1 Sets the corresponding bit in the RGPIO_DATA register

### 20.3.6 RGPIO Toggle Data (RGPIO\_TOG)

The RGPIO\_TOG register provides a mechanism to invert (toggle) specific bits in the RGPIO\_DATA register by performing a simple write. Setting a bit in RGPIO\_TOG inverts the corresponding bit in the RGPIO\_DATA register. Clearing it has no effect. The RGPIO\_TOG register is write-only; reads of this address return the RGPIO\_DATA register.

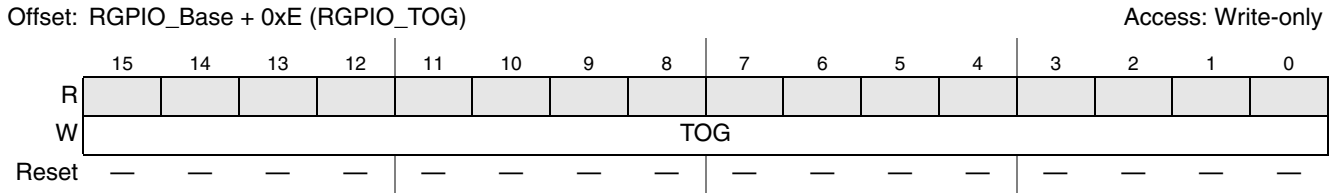


Figure 20-7. RGPIO Toggle Data Register (RGPIO\_TOG)

Table 20-10. RGPIO\_TOG Field Descriptions

Field	Description
15–0 TOG	Toggle data. 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

## 20.4 Functional Description

The RGPIO module is a relatively-simple design with its behavior controlled by the program-visible registers defined within its programming model.

The RGPIO module is connected to the processor’s local two-stage pipelined bus with the stages of the ColdFire core’s operand execution pipeline (OEP) mapped directly onto the bus. This structure allows the processor access to the RGPIO module for single-cycle pipelined reads and writes with a zero wait-state response (as viewed in the system bus data phase stage).

## 20.5 Initialization Information

The reset state of the RGPIO module disables the entire 16-bit data port. Prior to using the RGPIO port, software typically:

- Enables the appropriate pins in RGPIO\_ENB
- Configures the pin direction in RGPIO\_DIR
- Defines the contents of the data register (RGPIO\_DATA)

## 20.6 Application Information

This section examines the relative performance of the RGPIO output pins for two simple applications

- The processor executes a loop to toggle an output pin for a specific number of cycles, producing a square-wave output
- The processor transmits a 16-bit message using a three-pin SPI-like interface with a serial clock, serial chip select, and serial data bit.

In both applications, the relative speed of the GPIO output is presented as a function of the location of the output bit (RGPIO versus peripheral bus GPIO).

## 20.6.1 Application 1: Simple Square-Wave Generation

In this example, several different instruction loops are executed, each generating a square-wave output with a 50% duty cycle. For this analysis, the executed code is mapped into the processor's RAM. This configuration is selected to remove any jitter from the output square wave caused by the limitations defined by the two-cycle flash memory accesses and restrictions on the initiation of a flash access. The following instruction loops were studied:

- **BCHG\_LOOP** — In this loop, a bit change instruction was executed using the GPIO data byte as the operand. This instruction performs a read-modify-write operation and inverts the addressed bit. A pulse counter is decremented until the appropriate number of square-wave pulses have been generated.
- **SET+CLR\_LOOP** — For this construct, two store instructions are executed: one to set the GPIO data pin and another to clear it. Single-cycle NOP instructions (the tpf opcode) are included to maintain the 50% duty cycle of the generated square wave. The pulse counter is decremented until the appropriate number of square-wave pulse have been generated.

The square-wave output frequency was measured and the relative performance results are presented in [Table 20-11](#). The relative performance is stated as a fraction of the processor's operating frequency, defined as  $f$  MHz. The performance of the BCHG loop operating on a GPIO output is selected as the reference.

**Table 20-11. Square-Wave Output Performance**

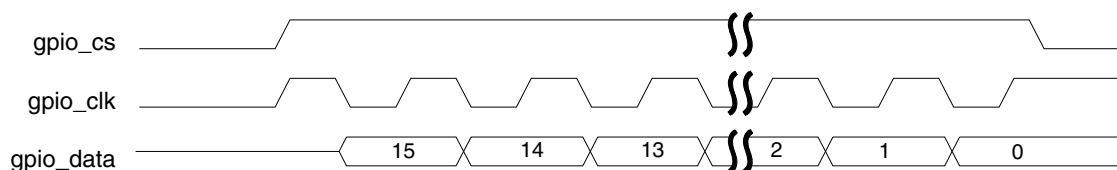
Loop	Peripheral Bus-mapped GPIO			RGPIO		
	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed
<i>bchg</i>	$(1/24) \times f$ MHz	2.083 MHz	1.00x	$(1/14) \times f$ MHz	3.571 MHz	1.71x
<i>set+clr (+toggle)</i>	$(1/12) \times f$ MHz	4.167 MHz	2.00x	$(1/8) \times f$ MHz	6.250 MHz	3.00x

### NOTE

The square-wave frequency is measured from rising-edge to rising-edge, where the output wave has a 50% duty cycle.

## 20.6.2 Application 2: 16-bit Message Transmission using SPI Protocol

In this second example, a 16-bit message is transmitted using three programmable output pins. The output pins include a serial clock, an active-high chip select, and the serial data bit. The software is configured to sample the serial data bit at the rising-edge of the clock with the data sent in a most-significant to least-significant bit order. The resulting 3-bit output is shown in [Figure 20-8](#).



**Figure 20-8. GPIO SPI Example Timing Diagram**



## Rapid GPIO (RGPIO)

For this example, the processing of the SPI message is considerably more complex than the generation of a simple square wave of the previous example. The code snippet used to extract the data bit from the message and build the required GPIO data register writes is shown in [Figure 20-9](#).

```
# subtest: send a 16-bit message via a SPI interface using a RGPIO

        # the SPI protocol uses a 3-bit value: clock, chip-select, data
        # the data is centered around the rising-edge of the clock

        align 16
send_16b_spi_message_rgpio:
00510: 4fef fff4          lea    -12(%sp),%sp      # allocate stack space
00514: 48d7 008c          movm.l &0x8c,(%sp)      # save d2,d3,d7
00518: 3439 0080 0582      mov.w  RAM_BASE+message2,%d2 # get 16-bit message
0051e: 760f              movq.l &15,%d3          # static shift count
00520: 7e10              movq.l &16,%d7          # message bit length
00522: 207c 00c0 0003      mov.l  &RGPIO_DATA+1,%a0 # pointer to low-order data byte
00528: 203c 0000 ffff      mov.l  &0xffff,%d0       # data value for _ENB and _DIR regs
0052e: 3140 fffd          mov.w  %d0,-3(%a0)      # set RGPIO_DIR register
00532: 3140 0001          mov.w  %d0,1(%a0)       # set RGPIO_ENB register

00536: 223c 0001 0000      mov.l  &0x10000,%d1      # d1[17:16] = {clk, cs}
0053c: 2001              mov.l  %d1,%d0          # copy into temp reg
0053e: e6a8              lsr.l  %d3,%d0          # align in d0[2:0]
00540: 5880              addq.l &4,%d0            # set clk = 1
00542: 1080              mov.b  %d0,(%a0)        # initialize data
00544: 6002              bra.b  L%1

L%1:
00548: 3202              mov.w  %d2,%d1          # d1[17:15] = {clk, cs, data}
0054a: 2001              mov.l  %d1,%d0          # copy into temp reg
0054c: e6a8              lsr.l  %d3,%d0          # align in d0[2:0]
0054e: 1080              mov.b  %d0,(%a0)        # transmit data with clk = 0
00550: 5880              addq.l &4,%d0            # force clk = 1
00552: e38a              lsl.l  &1,%d2           # d2[15] = new message data bit
00554: 51fc              tpf                                # preserve 50% duty cycle
00556: 51fc              tpf
00558: 51fc              tpf
0055a: 51fc              tpf
0055c: 1080              mov.b  %d0,(%a0)        # transmit data with clk = 1
0055e: 5387              subq.l &1,%d7           # decrement loop counter
00560: 66e6              bne.b  L%1

00562: c0bc 0000 fff5      and.l  &0xfff5,%d0      # negate chip-select
00568: 1080              mov.b  %d0,(%a0)        # update gpio

0056a: 4cd7 008c          movm.l (%sp),&0x8c      # restore d2,d3,d7
0056e: 4fef 000c          lea    12(%sp),%sp      # deallocate stack space
00572: 4e75              rts
```

**Figure 20-9. GPIO SPI Code Example**

The resulting SPI performance, as measured in the effective Mbps transmission rate for the 16-bit message, is shown in [Table 20-12](#).

**Table 20-12. Emulated SPI Performance using GPIO Outputs**

Peripheral Bus-mapped GPIO		RGPIO	
SPI Speed @ CPU $f = 50$ MHz	Relative Speed	SPI Speed @ CPU $f = 50$ MHz	Relative Speed
2.063 Mbps	1.00x	3.809 Mbps	1.29x

# Chapter 21

## Real-Time Counter (S08RTCV2)

### 21.1 Introduction

The Real-Time Counter (RTC) module consists of one 8-bit counter, one 8-bit comparator, several binary-based and decimal-based prescaler dividers, two clock sources, and one programmable periodic interrupt. This module can be used for time-of-day, calendar, or any task scheduling functions. It can also serve as a cyclic wakeup from low power modes without the need of external components.

### 21.2 Low Power Mode Operation

The RTC module is capable of functioning in all modes of operation. The overflow from the RTC is available to wake the MCU from stop modes.

For details on low-power mode operation, refer to [Table 3-1](#) in [Chapter 3](#), “Modes of Operation”.

#### 21.2.1 RTC Clock Gating

The bus clock to the RTC can be gated on and off using the SCGC2[RTC] bits (see [Section 5.8.10](#), “System Clock Gating Control 2 Register (SCGC2)”). This bit is cleared after any reset that disables the bus clock to this module. The SCGC2[RTC] bits must be set before operation. See [Section 5.7](#), “Peripheral Clock Gating,” for details.

## 21.2.2 Features

Features of the RTC module include:

- 8-bit up-counter
  - 8-bit modulo match limit
  - Software controllable periodic interrupt or DMA request on match
- Three software selectable clock sources for input to prescaler with selectable binary-based and decimal-based divider values
  - 1-kHz internal low-power oscillator (LPO)
  - External clock (ERCLK)
  - 32-kHz internal clock (IRCLK)

## 21.2.3 Modes of Operation

This section defines the operation in stop, wait and background debug modes.

### 21.2.3.1 Wait Mode

The RTC continues to run in wait mode if enabled before executing the appropriate instruction. Therefore, the RTC can bring the MCU out of wait mode if the real-time interrupt is enabled. Either the interrupt or DMA request can bring the MCU out of wait modes. For lowest possible current consumption, the RTC should be stopped by software if not needed as an interrupt source during wait mode.

### 21.2.3.2 Stop Modes

The RTC continues to run in stop2 or stop3 mode if the RTC is enabled before executing the STOP instruction. Therefore, the RTC can bring the MCU out of stop modes with no external components, if the real-time interrupt is enabled. The interrupt request can bring the MCU out of stop modes.

The LPO clock can be used in stop2 and stop3 modes. ERCLK and IRCLK clocks are only available in stop3 mode. Power consumption is lower when all clock sources are disabled, but in that case, the real-time interrupt cannot wake the MCU from stop modes.

### 21.2.3.3 Active Background Mode

The RTC suspends all counting during active background mode until the microcontroller returns to normal user operating mode. Counting resumes from the suspended value as long as the RTCMOD register is not written and the RTCPS and RTCLKS bits are not altered.

### 21.2.4 Block Diagram

The block diagram for the RTC module is shown in Figure 21-1.

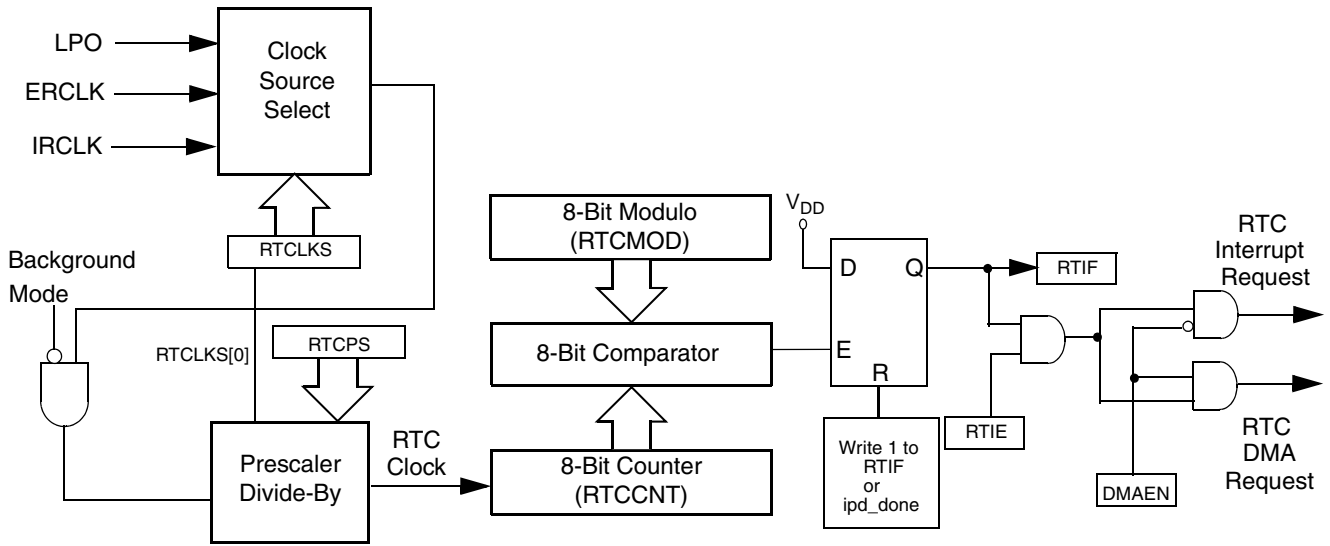


Figure 21-1. Real-Time Counter (RTC) Block Diagram

### 21.3 External Signal Description

The RTC does not include any off-chip signals.

### 21.4 Register Definition

The RTC includes a status and control register, an 8-bit counter register, and an 8-bit modulo register.

Refer to the direct-page register summary in the memory section of this document for the absolute address assignments for all RTC registers. This section refers to registers and control bits only by their names and relative address offsets.

Table 21-1 is a summary of RTC registers.

Table 21-1. RTC Register Summary

Name		7	6	5	4	3	2	1	0
RTCSC	R	RTIF	RTCLKS		RTIE	RTCPS			
	W								
RTCCNT	R	RTCCNT							
	W								
RTCMOD	R	RTCMOD							
	W								
RTCSC1	R	DMAEN							
	W								

## 21.4.1 RTC Status and Control Register (RTCSC)

RTCSC contains the real-time interrupt status flag (RTIF), the clock select bits (RTCLKS), the real-time interrupt enable bit (RTIE), and the prescaler select bits (RTCPS).

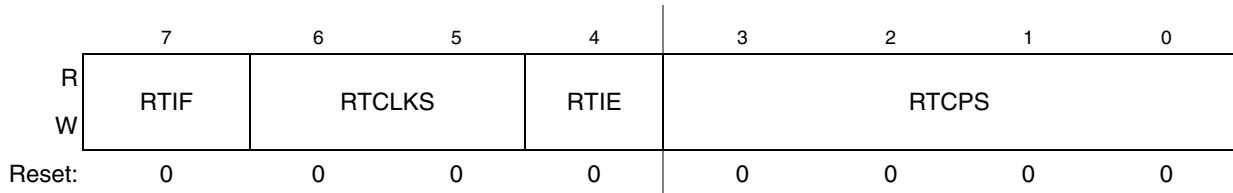


Figure 21-2. RTC Status and Control Register (RTCSC)

Table 21-2. RTCSC Field Descriptions

Field	Description
7 RTIF	Real-Time Interrupt Flag This status bit indicates the RTC counter register reached the value in the RTC modulo register. Writing a logic 0 has no effect. Writing a logic 1 or detecting the corresponding DMA data transfer has been completed(ipd_done) clears this bit and the real-time interrupt/DMA request. Reset clears RTIF. 0 RTC counter has not reached the value in the RTC modulo register. 1 RTC counter has reached the value in the RTC modulo register.
6–5 RTCLKS	Real-Time Clock Source Select. These two read/write bits select the clock source input to the RTC prescaler. Changing the clock source clears the prescaler and RTCCNT counters. When selecting a clock source, ensure that the clock source is properly enabled (if applicable) to ensure correct operation of the RTC. Reset clears RTCLKS. 00 Real-time clock source is the 1-kHz low power oscillator (LPO) 01 Real-time clock source is the external clock (ERCLK) 1x Real-time clock source is the internal clock (IRCLK)
4 RTIE	Real-Time Interrupt Enable. This read/write bit enables both the real-time interrupts and DMA request. If both RTIE and RTCSC1.DMAEN are set, then an DMA request is generated when RTIF is set. If RTIE is set but RTCSC1.DMAEN is unset, then an interrupt is generated when RTIF is set. Reset clears RTIE. 0 Real-time interrupt requests and DMA request are disabled. Use software polling. 1 Real-time interrupt requests and DMA request are enabled.
3–0 RTCPS	Real-Time Clock Prescaler Select. These four read/write bits select binary-based or decimal-based divide-by values for the clock source. See <a href="#">Table 21-3</a> . Changing the prescaler value clears the prescaler and RTCCNT counters. Reset clears RTCPS.

Table 21-3. RTC Prescaler Divide-by values

RTCLKS[0]	RTCPS															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Off	2 <sup>3</sup>	2 <sup>5</sup>	2 <sup>6</sup>	2 <sup>7</sup>	2 <sup>8</sup>	2 <sup>9</sup>	2 <sup>10</sup>	1	2	2 <sup>2</sup>	10	2 <sup>4</sup>	10 <sup>2</sup>	5x10 <sup>2</sup>	10 <sup>3</sup>
1	Off	2 <sup>10</sup>	2 <sup>11</sup>	2 <sup>12</sup>	2 <sup>13</sup>	2 <sup>14</sup>	2 <sup>15</sup>	2 <sup>16</sup>	10 <sup>3</sup>	2x10 <sup>3</sup>	5x10 <sup>3</sup>	10 <sup>4</sup>	2x10 <sup>4</sup>	5x10 <sup>4</sup>	10 <sup>5</sup>	2x10 <sup>5</sup>

### 21.4.2 RTC Counter Register (RTCCNT)

RTCCNT is the read-only value of the current RTC count of the 8-bit counter.

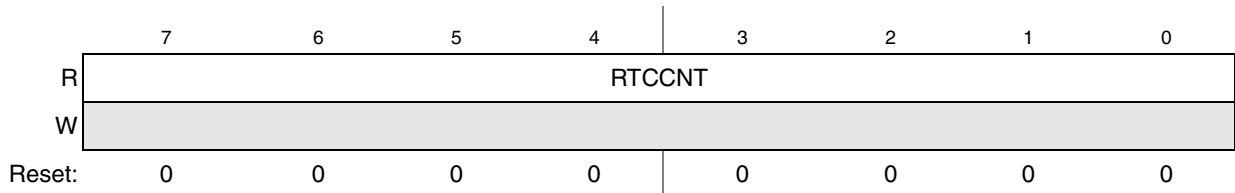


Figure 21-3. RTC Counter Register (RTCCNT)

Table 21-4. RTCCNT Field Descriptions

Field	Description
7:0 RTCCNT	RTC Count. These eight read-only bits contain the current value of the 8-bit counter. Writes have no effect to this register. Reset, writing to RTCMOD, or writing different values to RTCLKS and RTCPS clear the count to 0x00.

### 21.4.3 RTC Modulo Register (RTCMOD)

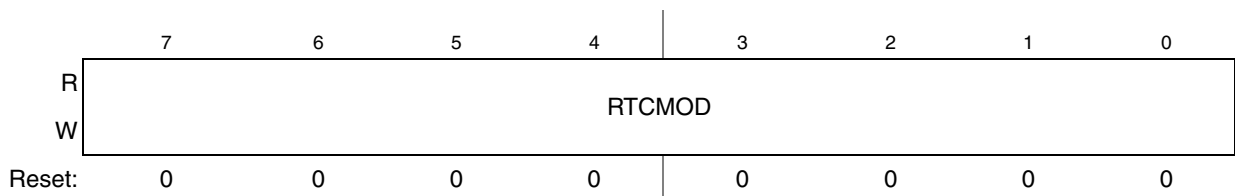


Figure 21-4. RTC Modulo Register (RTCMOD)

Table 21-5. RTCMOD Field Descriptions

Field	Description
7:0 RTCMOD	RTC Modulo. These eight read/write bits contain the modulo value used to reset the count to 0x00 upon a compare match and set the RTIF status bit. A value of 0x00 sets the RTIF bit on each rising edge of the prescaler output. Writing to RTCMOD resets the prescaler and the RTCCNT counters to 0x00. Reset sets the modulo to 0x00.

### 21.4.4 RTC DMA Enable Register (RTCSC1)

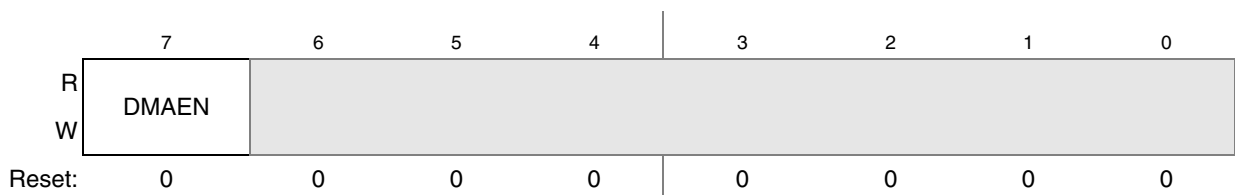


Figure 21-5. RTC DMA Enable Register (RTCDMAE)

Table 21-6. RTCMOD Field Descriptions

Field	Description
7 DMAEN	RTC DMA Enable. . This read/write bit enables DMA request. If both DMAEN and RTCSC.RTIE are set, then an ipd_req is generated when RTIF is set . Reset clears DMAEN. 0 DMA requests are disabled. Use software polling. 1 DMA requests are enabled.

## 21.5 Functional Description

The RTC is composed of a main 8-bit up-counter with an 8-bit modulo register, a clock source selector, and a prescaler block with binary-based and decimal-based selectable values. The module also contains software selectable interrupt logic.

After any MCU reset, the counter is stopped and reset to 0x00, the modulus register is set to 0x00, and the prescaler is off. The 1-kHz internal oscillator clock is selected as the default clock source. To start the prescaler, write any value other than zero to the prescaler select bits (RTCPS).

Three clock sources are software selectable: the low power oscillator clock (LPO), the external clock (ERCLK), and the internal clock (IRCLK). The RTC clock select bits (RTCLKS) select the desired clock source. If a different value is written to RTCLKS, the prescaler and RTCCNT counters are reset to 0x00.

RTCPS and the RTCLKS[0] bit select the desired divide-by value. If a different value is written to RTCPS, the prescaler and RTCCNT counters are reset to 0x00. [Table 21-7](#) shows different prescaler period values.

Table 21-7. Prescaler Period

RTCPS	1-kHz Internal Clock (RTCLKS = 00)	1-MHz External Clock (RTCLKS = 01)	32-kHz Internal Clock (RTCLKS = 10)	32-kHz Internal Clock (RTCLKS = 11)
0000	Off	Off	Off	Off
0001	8 ms	1.024 ms	250 $\mu$ s	32 ms
0010	32 ms	2.048 ms	1 ms	64 ms
0011	64 ms	4.096 ms	2 ms	128 ms
0100	128 ms	8.192 ms	4 ms	256 ms
0101	256 ms	16.4 ms	8 ms	512 ms
0110	512 ms	32.8 ms	16 ms	1.024 s
0111	1.024 s	65.5 ms	32 ms	2.048 s
1000	1 ms	1 ms	31.25 $\mu$ s	31.25 ms
1001	2 ms	2 ms	62.5 $\mu$ s	62.5 ms
1010	4 ms	5 ms	125 $\mu$ s	156.25 ms
1011	10 ms	10 ms	312.5 $\mu$ s	312.5 ms
1100	16 ms	20 ms	0.5 ms	0.625 s
1101	0.1 s	50 ms	3.125 ms	1.5625 s



Table 21-7. Prescaler Period (continued)

RTCPS	1-kHz Internal Clock (RTCLKS = 00)	1-MHz External Clock (RTCLKS = 01)	32-kHz Internal Clock (RTCLKS = 10)	32-kHz Internal Clock (RTCLKS = 11)
1110	0.5 s	0.1 s	15.625 ms	3.125 s
1111	1 s	0.2 s	31.25 ms	6.25 s

The RTC modulo register (RTCMOD) allows the compare value to be set to any value from 0x00 to 0xFF. When the counter is active, the counter increments at the selected rate until the count matches the modulo value. When these values match, the counter resets to 0x00 and continues counting. The real-time interrupt flag (RTIF) is set when a match occurs. The flag sets on the transition from the modulo value to 0x00. Writing to RTCMOD resets the prescaler and the RTCCNT counters to 0x00.

The **RTC** allows for an interrupt or a DMA request to be generated when RTIF is set. To enable the real-time interrupt, set the real-time interrupt enable bit (RTIE) in **RTCS** and unset the DMA request enable bit (DMAEN) in **RTCS1**. To enable the real-time DMA request, set the real-time interrupt enable bit (RTIE) in **RTCS** and the DMA request enable bit (DMAEN) in **RTCS1**. RTIF is cleared by writing a 1 to RTIF, or detecting the corresponding DMA data transfer has been completed (ipd\_done is asserted).

### 21.5.1 RTC Operation Example

This section shows an example of the RTC operation as the counter reaches a matching value from the modulo register.

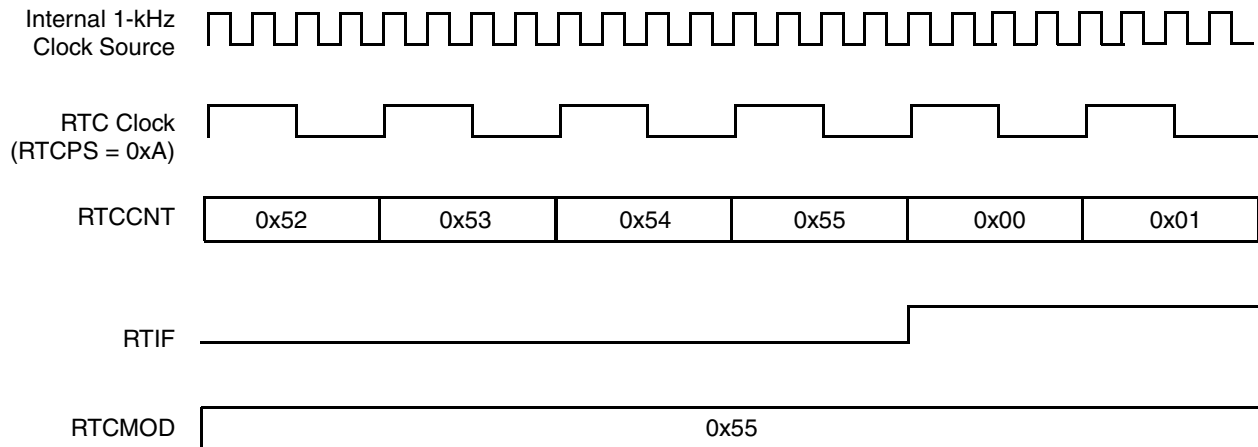


Figure 21-6. RTC Counter Overflow Example

In the example of [Figure 21-6](#), the selected clock source is the 1-kHz internal oscillator clock source. The prescaler (RTCPS) is set to 0xA or divide-by-4. The modulo value in the RTCMOD register is set to 0x55. When the counter, RTCCNT, reaches the modulo value of 0x55, the counter overflows to 0x00 and continues counting. The real-time interrupt/DMA request flag, RTIF, sets when the counter value changes from 0x55 to 0x00. A real-time interrupt or DMA request is generated when RTIF is set, if RTIE is set.

## 21.6 Initialization/Application Information

This section provides example code to give some basic direction to a user on how to initialize and configure the RTC module. The example software is implemented in C language.

The example below shows how to implement time of day with the RTC using the 1-kHz clock source to achieve the lowest possible power consumption. Because the 1-kHz clock source is not as accurate as a crystal, software can be added for any adjustments. For accuracy without adjustments at the expense of additional power consumption, the external clock (ERCLK) or the internal clock (IRCLK) can be selected with appropriate prescaler and modulo values.

```

/* Initialize the elapsed time counters */
Seconds = 0;
Minutes = 0;
Hours = 0;
Days=0;

/* Configure RTC to interrupt every 1 second from 1-kHz clock source */
RTCMOD.byte = 0x00;
RTCSC.byte = 0x1F;

/*****
Function Name : RTC_ISR
Notes : Interrupt service routine for RTC module.
*****/
#pragma TRAP_PROC
void RTC_ISR(void)
{
    /* Clear the interrupt flag */
    RTCSC.byte = RTCSC.byte | 0x80;
    /* RTC interrupts every 1 Second */
    Seconds++;
    /* 60 seconds in a minute */
    if (Seconds > 59){
        Minutes++;
        Seconds = 0;
    }
    /* 60 minutes in an hour */
    if (Minutes > 59){
        Hours++;
        Minutes = 0;
    }
    /* 24 hours in a day */
    if (Hours > 23){
        Days ++;
        Hours = 0;
    }
}

```

---

**Real-Time Counter (S08RTCV1)**

```
    }  
}
```

# Chapter 22

## Serial Communication Interface (SCI\_FlexV1)

### 22.1 Introduction

The MCF51AG128 series of MCUs include two independent serial communications interface (SCI) modules that are sometimes called universal asynchronous receiver/transmitters (UARTs). Typically, these systems are used to connect to the RS232 serial input/output (I/O) port of a personal computer or workstation, but they can also be used to communicate with other embedded controllers.

#### NOTE

- MCF51AG128 devices do not include stop1 low-power mode. Ignore references to stop1 in this chapter. For details on low-power mode operation, refer to [Table 3-1 in Chapter 3, “Modes of Operation.”](#)
- Ignore all references to LBKDDMAS, TCDMAS, and ILDMAS bits in this chapter.

#### 22.1.1 SCI Clock Gating

The bus clock to the SCI can be gated on and off using the SCGC1[SCIx] bits (see [Section 5.8.7, “System Power Management Status and Control 2 Register \(SPMSC2\)”](#)). This bit is set after any reset that enables the bus clock to this module. To conserve power, the SCGC1[SCIx] bits can be cleared to disable the clock to this module when not in use. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

#### 22.1.2 Module Block Diagram

[Figure 22-1](#) shows a block diagram of the SCI module.

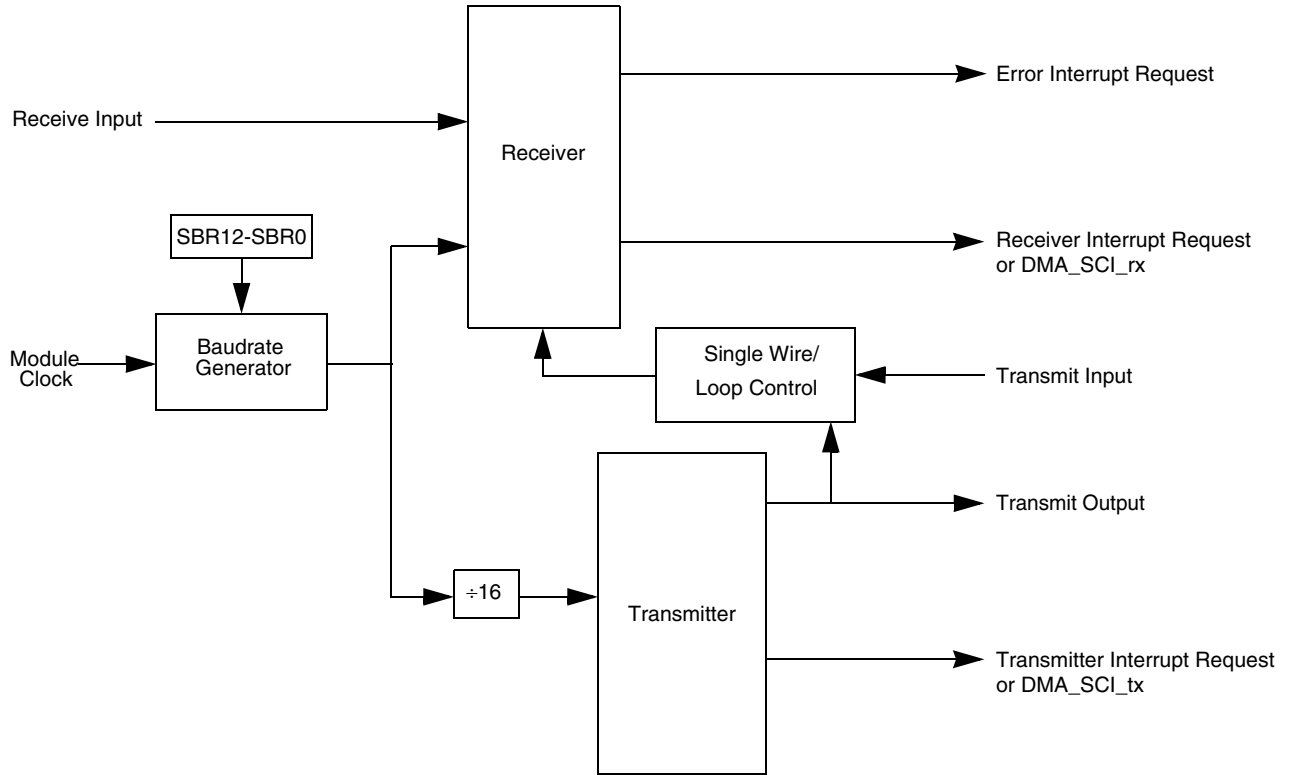


Figure 22-1. SCI Module Block Diagram

### 22.1.3 Features

The SCI includes these distinctive features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection with /32 fractional divide, based on bus frequency
- Programmable 8-bit or 9-bit data format
- Separately enabled transmitter and receiver
- Programmable transmitter output polarity
- Programmable receive input polarity
- 13-bit break character option
- 11-bit break character detection option
- Two receiver wakeup methods:
  - Idle line wakeup
  - Address mark wakeup
- Address match feature in receiver to reduce address mark wakeup ISR overhead
- Interrupt-driven operation with 10 flags:
  - Transmitter data register empty
  - Transmission complete
  - Receiver data register full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
  - Active edge on receive pin
  - LIN Break detect
- Receiver framing error detection
- Hardware parity generation and checking
- 1/16 bit-time noise detection
- 2 DMA requests and 3 interrupt signals

### 22.1.4 Modes of Operation

The SCI functions the same in normal, special, and emulation modes.

It has two low power modes, wait and stop modes.

### 22.1.4.1 Run Mode

Normal mode of operation.

### 22.1.4.2 Wait Mode

SCI operation in wait mode depends on the state of the SCISWAI bit in the SCI control register 1 (SCIC1). The SCISWAI bit and the wait mode signal bit are mapped to a clock enable signal, which is an output from the SCI.

- If the SCISWAI bit is cleared, the SCI operates normally when the CPU is in wait mode.
- If the SCISWAI bit is set, SCI clock generation ceases and the SCI module enters a power-conservation state when the CPU is in wait mode.

Setting SCISWAI does not affect the state of the receiver enable bit RE, or the transmitter enable bit TE.

If SCISWAI is set, any transmission or reception in progress stops at wait mode entry. The transmission or reception resumes when either an internal or external interrupt brings the CPU out of wait mode. Exiting wait mode by reset aborts any transmission or reception in progress and resets the SCI.

### 22.1.4.3 Stop Mode

The SCI is inactive during stop mode for reduced power consumption. The STOP instruction does not affect the SCI register states, but the SCI module clock will be disabled. The SCI operation resumes from where it left off after an external interrupt brings the CPU out of stop mode. Exiting stop mode by reset aborts any transmission or reception in progress and resets the SCI.

## 22.2 Memory Map and Registers

This section provides a detailed description of all memory and registers.

### 22.2.1 Module Memory Map

The memory map for the SCI module is given below in [Table 22-1](#). The Address listed for each register is the address offset. The total address for each register is the sum of the base address for the SCI module and the address offset for each register.

**Table 22-1. Module Memory Map**

Offset	Use	Access
0x00	SCI Baud Rate Register High (SClxBDH)	Read/Write
0x01	SCI Baud Rate Register Low (SClxBDL)	Read/Write
0x02	SCI Control Register1 (SClxC1)	Read/Write
0x03	SCI Control Register 2 (SClxC2)	Read/Write
0x04	SCI Status Register 1 (SClXS1)	Read
0x05	SCI Status Register 2(SClXS2)	Read/Write
0x06	SCI Control Register 3 (SClxC3)	Read/Write
0x07	SCI Data Register (SClxD)	Read/Write
0x08	SCI Match Address Register 1 (SClXMA1)	Read/Write

Table 22-1. Module Memory Map

0x09	SCI Match Address Register 2 (SCIxMA2)	Read/Write
0x0A	SCI Control Register 4 (SCIxC4)	Read/Write
0x0B	SCI Control Register 5 (SCIxC5)	Read/Write

## 22.2.2 Register Quick Reference

Register Name	Bit 7	6	5	4	3	2	1	Bit 0	Addr. offset
SCIxBDH Read: Write:	LBKDIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8	0x00
SCIxBDL Read: Write:	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	0x01
SCIxC1 Read: Write:	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT	0x02
SCIxC2 Read: Write:	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	0x03
SCIxS1 Read: Write:	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0x04
SCIxS2 Read: Write:	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF	0x05
SCIxC3 Read: Write:	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE	0x06
SCIxD Read: Write:	R7 T7	R6 T6	R5 T5	R4 T4	R3 T3	R2 T2	R1 T1	R0 T0	0x07
SCIxMA1 Read: Write:	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0	0x08
SCIxMA2 Read: Read:	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0	0x09
SCIxC4 Read: Write:	MAEN1	MAEN2	0	BRFA4	BRFA3	BRFA2	BRFA1	BRFA0	0x0A
SCIxC5 Read: Write:	TDMAS	0	RDMAS	0	0	0	0	0	0x0B


 = Unimplemented or Reserved

Figure 22-2. SCI Register Quick Reference



## 22.2.3 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

### 22.2.3.1 SCI Baud Rate Registers (SCIxBDH, SCIxBDL)

This pair of registers controls the prescale divisor for SCI baud rate generation. To update the 13-bit baud rate setting (SBR[12:0]), first write to SCIxBDH to buffer the high half of the new value and then write to SCIxBDL. The working value in SCIxBDH does not change until SCIxBDL is written.:

SCIxBDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled (RE or TE bits in SCIxC2 are written to 1).:

Register address: 0x00

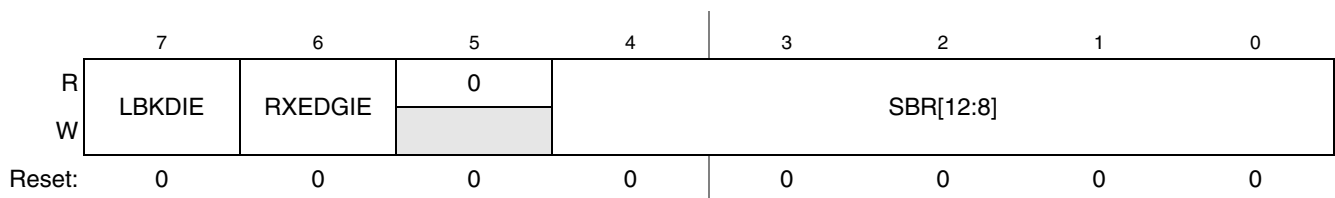


Figure 22-3. SCI Baud Rate Register (SCIxBDH):

Table 22-2. SCIxBDH Field Descriptions

Field	Description
7 LBKDIE	Lin Break Detect Interrupt Enable. LBKDIE enables the LIN Break Detect flag, LBKDIF, to generate interrupt requests. 0 LBKDIF interrupt request disabled. 1 LBKDIF interrupt request enabled.
6 RXEDGIE	RxD Input Active Edge Interrupt Enable. RXEDGIE enables the Receive Input Active Edge, RXEDGIF, to generate a interrupt requests. 0 Hardware interrupts from RXEDGIF disabled (use polling). 1 RXEDGIF interrupt request enabled.
4–0 SBR[12:8]	SCI Baud Rate Bits. The baud rate for the SCI is determined by these 13 bits. See <a href="#">Section 22.3.2</a> for detail of usage of the SBR bits. <b>Note:</b> The baud rate generator is disabled until the TE bit or the RE bit is set for the first time after reset. The baud rate generator is disabled when SBR = 0. <b>Note:</b> <i>Writing to SCIxBDH has no effect without writing to SCIxBDL, since writing to SCIxBDH puts the data in a temporary location until SCIxBDL is written.</i>

Register address: 0x01

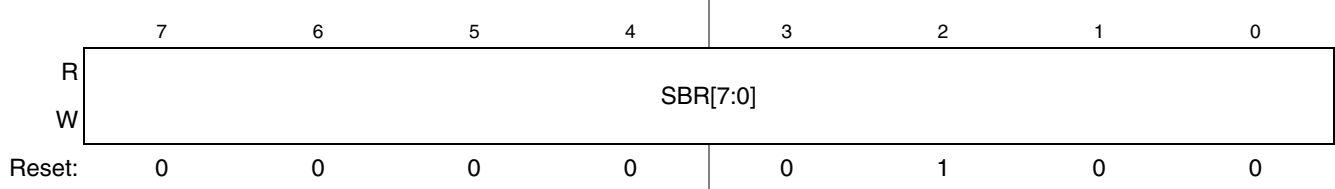


Figure 22-4. SCI Baud Rate Register (SCIxBDL):

Table 22-3. SCIxBDL Field Descriptions

Field	Description
7–0 SBR[7:0]	<p>SCI Baud Rate Bits. The baud rate for the SCI is determined by these 13 bits. See <a href="#">Section 22.3.2</a> for detail of usage of the SBR bits.</p> <p><b>Note:</b> The baud rate generator is disabled until the TE bit or the RE bit is set for the first time after reset. The baud rate generator is disabled when SBR = 0.</p> <p><b>Note:</b> <i>Writing to SCIxBDH</i> has no effect without writing to SCIxBDL, since writing to SCIxBDH puts the data in a temporary location until SCIxBDL is written.</p>

### 22.2.3.2 SCI Control Register 1 (SCIxC1)

This read/write register controls various optional features of the SCI system.

Register address: 0x02

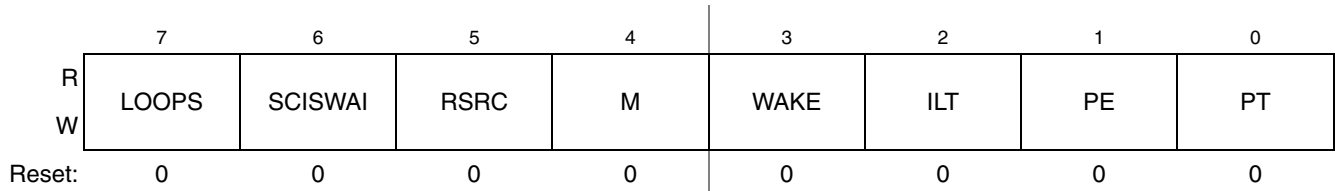


Figure 22-5. SCI Control Register 1 (SCIxC1)

Table 22-4. SCIxC1 Field Descriptions

Field	Description
7 LOOPS	<p>Loop Mode Select. When LOOPS is set, the RxD pin is disconnected from the SCI and the transmitter output is internally connected to the receiver input. The transmitter and the receiver must be enabled to use the loop function.</p> <p>0 Normal operation 1 Loop mode where transmitter outputs are internally connected to receiver input. The receiver input is determined by the RSRC bit.</p>
6 SCISWAI	<p>SCI Stops in Wait Mode. The receiver input is determined by the RSRC bit.</p> <p>0 SCI clock continues to run in wait mode. 1 SCI clock freezes while CPU is in wait mode.</p>

Table 22-4. SC1xC1 Field Descriptions (continued)

Field	Description
5 RSRC	Receiver Source Select. This bit has no meaning or effect unless the LOOPS bit is set. When LOOPS is set, the RSRC bit determines the source for the receiver shift register input. 0 Provided LOOPS is set, RSRC is cleared, selects internal loop back mode and receiver input is internally connected to transmitter output. 1 Single-wire SCI mode where the receiver input is connected to the transmit pin input signal.
4 M	9-bit or 8-bit Mode Select 0 Normal — start + 8 data bits (lsb first) + stop. 1 Use start + 9 data bits (lsb first) + stop.
3 WAKE	Receiver Wakeup Method Select. WAKE determines which condition wakes the SCI: address mark in the most significant bit position of a received data character or an idle condition on the receive pin input signal. 0 Idle-line wakeup. 1 Address-mark wakeup.
2 ILT	Idle Line Type Select. ILT determines when the receiver starts counting logic 1s as idle character bits. The counting begins either after a valid start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding the stop bit can cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires properly synchronized transmissions. 0 Idle character bit count starts after start bit. 1 Idle character bit count starts after stop bit. <b>Note:</b> In the case where SCI is programmed with ILT=1, a logic of 1'b0 is automatically shifted after a received a stop bit thus resetting the idle count. <b>Note:</b> In the case where SCI is programmed for IDLE line wakeup (RWU=1 and Wake=0), ILT has no affect when receiver starts counting logic 1s as idle character bits. In Idle Line Wakeup an idle character is recognized at anytime the receiver sees 10 or 11 1s depending on M-bit.
1 PE	Parity Enable. Enables the parity function. When parity is enabled, parity function inserts a parity bit in the most significant bit position. 0 Parity function disabled. 1 Parity function enabled.
0 PT	Parity Type. PT determines whether the SCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit. 0 Even parity. 1 Odd parity.

### 22.2.3.3 SCI Control Register 2 (SC1xC2)

This register can be read or written at any time.

Register address: 0x03

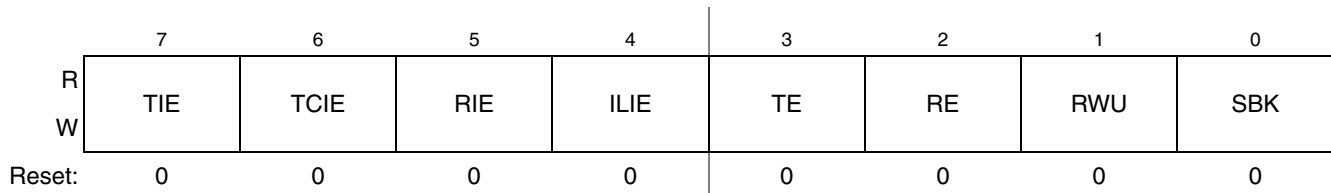


Figure 22-6. SCI Control Register 2 (SC1xC2)

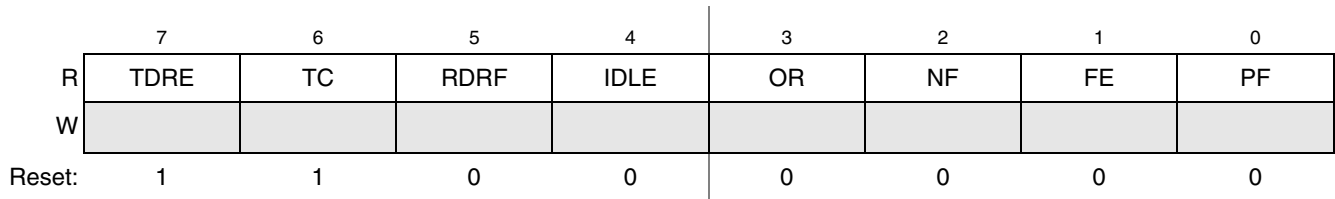
Table 22-5. SCIx2 Field Descriptions

Field	Description
7 TIE	Transmitter Interrupt or DMA Transfer Enable. TIE enables the transmit data register empty flag, TDRE, to generate interrupt requests or DMA transfer requests, based on the state of TDMAS. 0 TDRE interrupt and DMA transfer requests disabled. 1 TDRE interrupt or DMA transfer requests enabled.
6 TCIE	Transmission Complete Interrupt Enable. TCIE enables the transmission complete flag, TC, to generate interrupt requests. 0 TC interrupt requests disabled. 1 TC interrupt requests enabled.
5 RIE	Receiver Full Interrupt or DMA Transfer Enable. RIE enables the receive data register full flag, RDRF, to generate interrupt requests or DMA transfer requests, based on the state of RDMAS. 0 RDRF interrupt and DMA transfer requests disabled. 1 RDRF interrupt or DMA transfer requests enabled.
4 ILIE	Idle Line Interrupt Enable. ILIE enables the idle line flag, IDLE, to generate interrupt requests. 0 IDLE interrupt requests disabled. 1 IDLE interrupt requests enabled.
3 TE	Transmitter Enable. TE enables the SCI transmitter and configures the TXD pin as being controlled by the SCI. The TE bit can be used to queue an idle preamble. 0 Transmitter off. 1 Transmitter on.
2 RE	Receiver Enable. RE enables the SCI receiver. 0 Receiver off. 1 Receiver on.
1 RWU	Receiver Wakeup Control. This bit can be set to place the SCI receiver in a standby state. 0 Normal operation. 1 RWU enables the wakeup function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU.
0 SBK	Send Break. Toggling SBK sends one break character (10 or 11 logic 0s, if BRK13 is cleared; 13 or 14 logic 0s, if BRK13 is set). Toggling implies clearing the SBK bit before the break character has finished transmitting. As long as SBK is set, the transmitter continues to send complete break characters (10 or 11 bits, or 13 or 14 bits). 0 Normal transmitter operation. 1 Queue break character(s) to be sent.

### 22.2.3.4 SCI Status Register 1 (SCIxS1)

The SCIxS1 register provides inputs to the MCU for generation of SCI interrupts or DMA requests. Also this register can be polled by the MCU to check the status of these bits. The flag-clearing procedures for SCIxS1 flags require that the status register be read followed by a read or write (depending on interrupt flag type) to the SCI Data Register. It is permissible to execute other instructions between the two steps as long as it does not compromise the handling of I/O, but the order of operations is important for flag clearing. When a flag is configured to trigger a DMA request, assertion of the associated DMA done signal from the DMA controller, clears SCIxS1.

Register address: 0x04



**Figure 22-7. SCI Status Register 1 (SCIxS1)**

**Table 22-6. SCIxS1 Field Descriptions**

Field	Description
7 TDRE	<p>Transmit Data Register Empty Flag. TDRE is set when the transmit shift register receives a byte from the SCI data register. When TDRE is set, the transmit data register (SCIxD and T8 bit in SCIxC3[6]) is empty and can receive a new value to transmit. To clear TDRE, read SCIxS1 with TDRE set and then write to the SCI data register (SCIxD).</p> <p>0 No byte transferred to transmit shift register. 1 Byte transferred to transmit shift register; transmit data register empty.</p>
6 TC	<p>Transmit Complete Flag. TC is set low when there is a transmission in progress or when a preamble or break character is loaded. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the transmit data output signal becomes idle (logic 1).</p> <p>0 Transmitter active (sending data, a preamble, or a break). 1 Transmitter idle (transmission activity complete).</p> <p>TC is cleared automatically by reading SCIxS1 with TC set and then doing one of the following:</p> <ul style="list-style-type: none"> <li>• Write to the SCI data register (SCIxD) to transmit new data</li> <li>• Queue a preamble</li> <li>• Queue a break character by writing 1 to SBK in SCIxC2</li> </ul>
5 RDRF	<p>Receive Data Register Full Flag. RDRF is set when the data in the receive shift register transfers to the SCI data register. RDRF is prevented from setting while LBKDE is set. To clear RDRF, read SCIxS1 with RDRF set and then read the SCI data register (SCIxD).</p> <p>0 Data not available in SCI data register. 1 Received data available in SCI data register.</p>
4 IDLE	<p>Idle Line Flag. IDLE is set when 10 consecutive logic 1s (if M=0) or 11 consecutive logic 1s (if M=1) appear on the receiver input. After the IDLE flag is cleared, a valid frame must again set the RDRF flag or a LIN break character must set the LBKDIF flag before an idle condition can set the IDLE flag. To clear IDLE, read SCI status SCIxS1 with IDLE set and then read SCIxD.</p> <p>0 Receiver input is either active now or has never become active since the IDLE flag was last cleared. 1 Receiver input has become idle.</p> <p><b>Note:</b> When the receiver wakeup bit (RWU) is set and WAKE is cleared, an idle line condition sets the IDLE flag if RWUID is set, else the IDLE flag does not get set.</p>

Table 22-6. SCIxS1 Field Descriptions (continued)

Field	Description
3 OR	Receiver Overrun Flag. OR is set when software fails to read the SCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame and all the other error flags (FE,NF and PF) are prevented from setting. The data in the shift register is lost, but the data already in the SCI data registers is not affected. To clear OR, read SCIxS1 with OR set and then read SCI data register (SClxD). 0 No overrun. 1 Overrun. If LBKDE is enabled the OR flag only sets under two conditions. <ul style="list-style-type: none"> <li>• If RDRF sets prior to enabling LBKDE and software fails to read the SCID register before the receive shift register receives the next frame.</li> <li>• If a LIN Break Character is detected asserting LBKDIF flag and software fails to clear the LBKDIF flag before the receive shift register receives the next frame.</li> </ul> In both of these cases, the OR flag is set immediately after the STOP bit is completely received for the second frame.
2 NF	Noise Flag. NF is set when the SCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun or while the LIN Break Detect feature is enabled (LBKDE=1). To clear NF, read SCIxS1 and then read the SCI data register (SClxD). 0 No noise detected. 1 Noise detected in the received character in SClxD.
1 FE	Framing Error Flag. FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun or while the LIN Break Detect feature is enabled (LBKDE=1). FE inhibits further data reception until it is cleared. To clear FE, read SCIxS1 with FE set and then read the SCI data register (SClxD). 0 No framing error detected. 1 Framing error.
0 PF	Parity Error Flag. PF is set when PE is set, LBKDE is disabled, and the parity of the received data does not match its parity bit. To clear PF, read SCIxS1 and then read the SCI data register (SClxD). 0 No parity error. 1 Parity error.

### 22.2.3.5 SCI Status Register 2 (SCIxS2)

The SCIxS2 register provides inputs to the MCU for generation of SCI interrupts or DMA requests. Also, this register can be polled by the MCU to check the status of these bits. This register can be read or written at any time. Writing to RAF bit does not have any effect.

Register address: 0x05

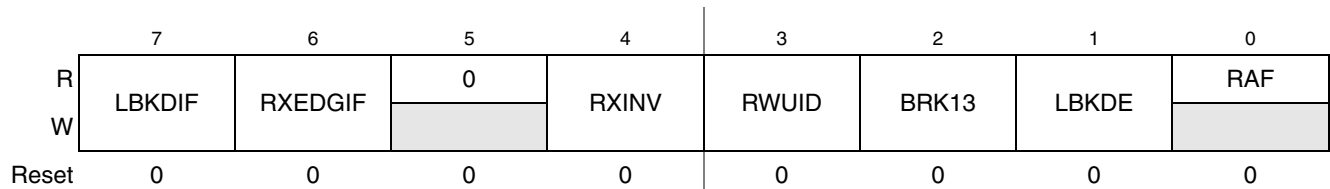


Figure 22-8. SCI Status Register 2 (SCIxS2)

Table 22-7. SCIS2 Field Descriptions

Field	Description
7 LBKDIF	LIN Break Detect Interrupt Flag. LBKDIF is set when LBKDE is set and a LIN break character is detected when 11 consecutive logic 0s (if M=0) or 12 consecutive logic 0s (if M=1) appear on the receiver input. LBKDIF is cleared by writing a 1 to it. 0 No LIN break character is detected. 1 LIN break character is detected.
6 RXEDGIF	RxD Pin Active Edge Interrupt Flag. RXEDGIF is set when an active edge (falling if RXINV = 0, rising if RXINV=1) on the RxD pin occurs. RXEDGIF is cleared by writing a 1 to it. 0 No active edge on the receive pin has occurred. 1 An active edge on the receive pin has occurred.
4 RXINV	Receive Data Inversion. Setting this bit, reverses the polarity of the received data input. 0 Receive data is not inverted. 1 Receive data is inverted. <b>Note:</b> Setting RXINV inverts the RxD input for all cases: data bits, start and stop bits, break, and idle.
3 RWUID	Receive Wakeup Idle Detect. When RWU is set, this bit controls whether the idle character that wakes the receiver sets the IDLE bit. 0 The IDLE bit does not get set upon detection of an idle character. 1 The IDLE bit gets set upon detection of an idle character.
2 BRK13	Break Transmit Character Length. This bit determines whether the transmit break character is 10 or 11 bits long, or 13 or 14 bits long. The detection of a framing error is not affected by this bit. 0 Break character is 10 or 11 bits long. 1 Break character is 13 or 14 bits long.
1 LBKDE	LIN Break Detection Enable. LBKDE selects a longer break character detection length. While LBKDE is set, RDRF, NF, FE, and PF flags of SCIC1 register are prevented from setting. However OR and IDLE are not affected by LBKDE. 0 Break character is detected at length of 10 bit times (11 if M = 1). 1 Break character is detected at length of 11 bit times (12 if M = 1).
0 RAF	Receiver Active Flag. RAF is set when the SCI receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character. 0 SCI receiver idle waiting for a start bit. 1 SCI receiver active (RxD input not idle).

### 22.2.3.6 SCI Control Register 3 (SCIC3)

This register can be read and written at anytime. Writing to R8 bit does not have any effect.

Register address: 0x06

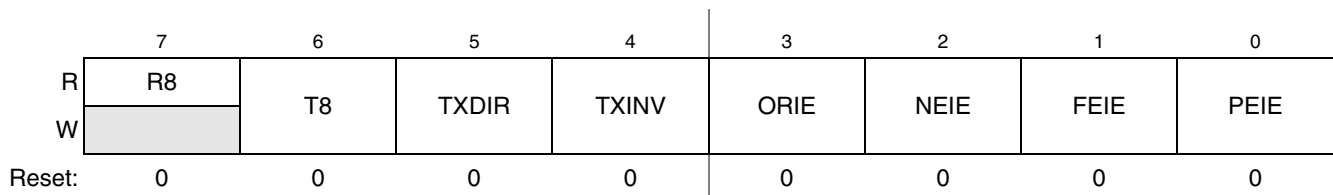


Figure 22-9. SCI Control Register 3 (SCIC3)

Table 22-8. SCIx3 Field Descriptions

Field	Description
7 R8	Received Bit 8.R8 is the ninth data bit received when the SCI is configured for 9-bit data for 9-bit data format (M = 1).
6 T8	Transmit Bit 8.T8 is the ninth data bit transmitted when the SCI is configured for 9-bit data format (M = 1). <b>Note:</b> If the value of T8 is the same as in the previous transmission, T8 does not have to be rewritten. The same value is transmitted until T8 is rewritten.
5 TXDIR	Transmitter Pin Data Direction in Single-Wire mode. This bit determines whether the TXD pin is used as an input or output in the Single-Wire mode of operation. This bit is only relevant in the Single-Wire mode of operation. 0 TXD pin is an input in single-wire mode. 1 TXD pin is an output in single-wire mode.
4 TXINV	Transmit Data Inversion. Setting this bit reverses the polarity of the transmitted data output. 0 Transmit data is not inverted. 1 Transmit data is inverted. <b>Note:</b> Setting TXINV inverts all transmitted values, including idle, break, start, and stop bits. In LOOP mode, if TXINV is set, the receiver gets the transmit inversion bit when RXINV is disabled.
3 ORIE	Overrun Error Interrupt Enable. This bit enables the overrun error flag (OR) to generate interrupt requests. 0 OR interrupts are disabled. 1 OR interrupt requests are enabled.
2 NEIE	Noise Error Interrupt Enable. This bit enables the noise flag (NF) to generate interrupt requests. 0 NF interrupt requests are disabled. 1 NF interrupt requests are enabled.
1 FEIE	Framing Error Interrupt Enable. This bit enables the framing error flag (FE) to generate interrupt requests. 0 FE interrupt requests are disabled. 1 FE interrupt requests are enabled.
0 PEIE	Parity Error Interrupt Enable. This bit enables the parity error flag (PF) to generate interrupt requests. 0 PF interrupt requests are disabled. 1 PF interrupt requests are enabled.

### 22.2.3.7 SCI Data Register (SCIxD)

This register is actually two separate registers. Reads return the contents of the read-only receive data register and writes go to the write-only transmit data register.

Register address: 0x07

	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W	T7	T6	T5	T4	T3	T2	T1	T0
Reset:	0	0	0	0	0	0	0	0

Figure 22-10. SCI Data Register (SCIxD)

#### NOTE

- In 8-bit or 9-bit data format, only SCI data register (SCIxD) needs to be accessed in order to clear RDRF bit.



- When transmitting in 9-bit data format and using 8-bit write instructions, write first to transmit bit 8 in SCI control register 3 (SCIxC3[6]), then SCIxD. A write to SCIxC3[6] stores the data in a temporary register. If SCIxD is written first then the new data on data bus is stored in SCIxD register, while the temporary value (written by last write to SCIxC3[6]) gets stored in SCIxC3[6] register.

### 22.2.3.8 SCI Match Address Registers 1 and 2 (SCIxMA1, SCIxMA2)

These registers can be read and written at anytime. The SCIxMA1 and SCIxMA2 registers compare input data addresses when most significant bit is set and associated MAEN bit in SCIxC4 register is set. If match occurs, following data is transferred to the data register. If match fails, following data is discarded. Refer to [Section 22.3.4.9](#) for more information.

Register address: 0x08 — SCIxMA1

Register address: 0x09 — SCIxMA2

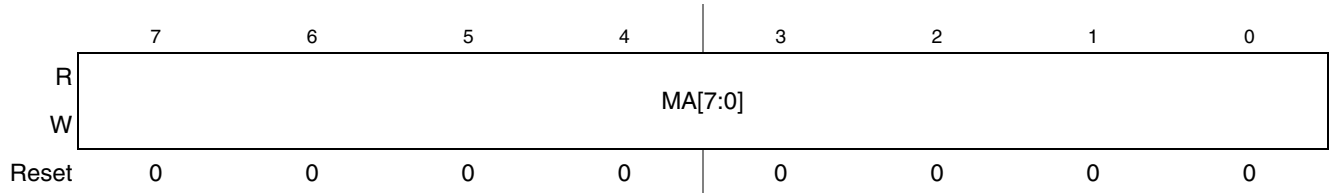


Figure 22-11. SCI Match Address Registers 1 and 2 (SCIxMA1, SCIxMA2)

### 22.2.3.9 SCI Control Register 4 (SCIxC4)

Register address: 0x0A

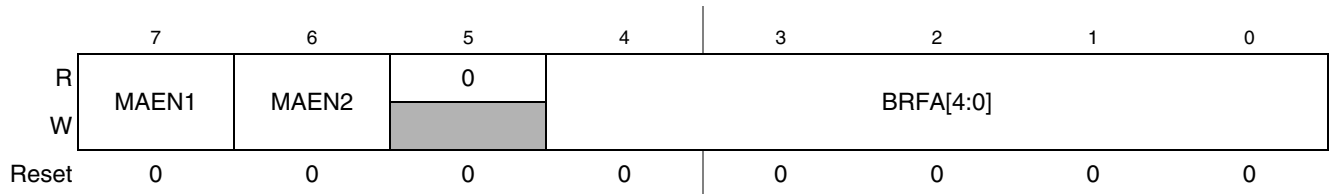


Figure 22-12. SCI Control Register 4 (SCIxC4)

Table 22-9. SC1xC4 Field Descriptions

Field	Description
7 MAEN1	Match Address Mode Enable 1. 0 All data received is transferred to the data register and RDRF is set if MAEN2 is cleared. 1 All data received with most significant bit cleared, is discarded. All data received with most significant bit set, is compared with contents of SC1xMA1 register. If no match occurs, the data is discarded and RDRF is not set. If match occurs, data is transferred to the data register and RDRF is set. Refer to <a href="#">Section 22.3.4.9</a> for more information.
7–6 MAEN2	Match Address Mode Enable 2. 0 All data received is transferred to the data register and RDRF is set if MAEN1 is cleared. 1 All data received with most significant bit cleared, is discarded. All data received with most significant bit set, is compared with contents of SC1xMA2 register. If no match occurs, the data is discarded and RDRF is not set. If match occurs, data is transferred to the data register and RDRF is set. Refer to <a href="#">Section 22.3.4.9</a> for more information.
4–0 BRFA[4:0]	Baud rate fine adjust. This bit field is used to add more timing resolution to the average baud frequency, in increments of 1/32. Refer to <a href="#">Section 22.3.2</a> for more information.

### 22.2.3.10 SCI Control Register 5 (SC1xC5)

Register address: 0x0B

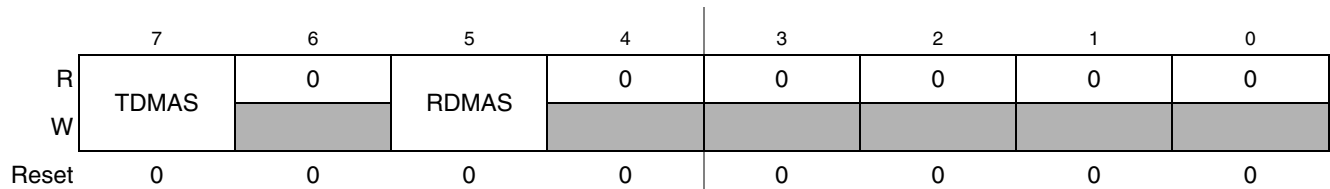


Figure 22-13. SCI Control Register 4 (SC1xC5)

Table 22-10. SC1xC5 Field Descriptions

Field	Description
7 TDMAS	Transmitter DMA Select. TDMAS configures the transmit data register empty flag, TDRE, to generate interrupt or DMA requests if TIE is set. 0 If TIE is set and the TDRE flag is set, the TDRE interrupt request signal is asserted to request interrupt service. 1 If TIE is set and the TDRE flag is set, the TDRE DMA request signal is asserted to request a DMA transfer. <b>Note:</b> If TIE is cleared, TDRE DMA and TDRE interrupt request signals are not asserted when the TDRE flag is set, regardless of the state of TDMAS.
5 RDMAS	Receiver Full DMA Select. RDMAS configures the receiver data register full flag, RDRF, to generate interrupt or DMA requests if RIE is set. 0 If RIE is set and the RDRF flag is set, the RDRF interrupt request signal is asserted to request interrupt service. 1 If RIE is set and the RDRF flag is set, the RDRF DMA request signal is asserted to request a DMA transfer. <b>Note:</b> If RIE is cleared, the RDRF DMA and RDRF interrupt request signals are not asserted when the RDRF flag is set, regardless of the state of RDMAS.

## 22.3 Functional Description

This section provides a complete functional description of the SCI block, detailing the operation of the design from the end user perspective in a number of subsections.

Figure 22-1 shows the structure of the SCI module. The SCI allows full duplex, asynchronous, NRZ serial communication between the CPU and remote devices, including other CPUs. The SCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the SCI, writes the data to be transmitted, and processes received data.

### 22.3.1 Data Format

The SCI uses the standard NRZ mark/space data format illustrated in Figure 22-14 below.

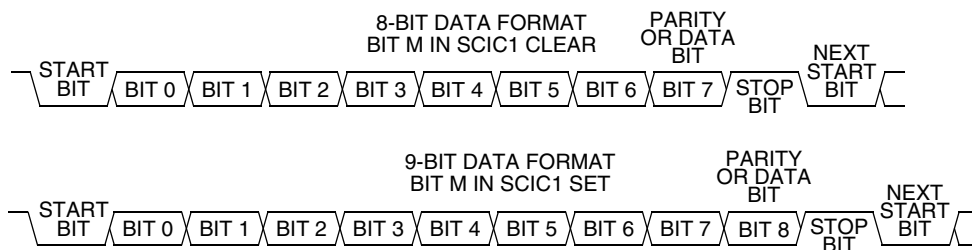


Figure 22-14. SCI Data Formats

Each data character is contained in a frame that includes a start bit, eight or nine data bits, and a stop bit. Clearing the M bit in SCI control register 1 configures the SCI for 8-bit data characters. A frame with eight data bits has a total of 10 bits. Setting the M bit configures the SCI for nine-bit data characters. A frame with nine data bits has a total of 11 bits.

Table 22-11. Example of 8-bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	8	0	0	1
1	7	0	1	1
1	7	1 <sup>1</sup>	0	1

<sup>1</sup> The address bit identifies the frame as an address character. See Section 22.3.4.8.

When the SCI is configured for 9-bit data characters, the ninth data bit is the T8 bit in SCI control register 3 (SCIxC3[6]). It remains unchanged after transmission and can be used repeatedly without rewriting it. A frame with nine data bits has a total of 11 bits.

Table 22-12. Example of 9-bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	9	0	0	1
1	8	0	1	1
1	8	1 <sup>1</sup>	0	1

<sup>1</sup> The address bit identifies the frame as an address character. See [Section 22.3.4.8](#).

## 22.3.2 Baud Rate Generation

A 13-bit modulus counter and a 5-bit fractional fine-adjust counter in the baud rate generator derive the baud rate for both the receiver and the transmitter. The value from 1 to 8191 written to the SBR[12:0] bits determines the module clock divisor. The SBR bits are in the SCI baud rate registers (SCIxBDH and SCIxBDL). The baud rate clock is synchronized with the bus clock and drives the receiver. The fractional fine-adjust counter adds fractional delays to the baud rate clock to allow fine trimming of the baud rate to match the system baud rate. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

Baud rate generation is subject to two sources of error:

- Integer division of the module clock may not give the exact target frequency. This error can be reduced by means of the fine-adjust counter.
- Synchronization with the bus clock can cause phase shift.

[Table 22-13](#) lists some examples of achieving target baud rates with a module clock frequency of 10.2 MHz, with and without fractional fine adjustment.

[Table 22-14](#) lists the available baud divisor fine adjust values.

$$\text{SCI baud rate} = \text{SCI module clock} / (16 * (\text{SBR}[12:0] + \text{BRFD}))$$

*Eqn. 22-1*

**Table 22-13. Baud Rates (Example: Module Clock = 10.2 Mhz)**

Bits SBR (decimal)	Bits BRFA	BRFD Value	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
17	00000	0	600,000.0	37,500.0	38,400	2.3
16	10011	19/32=0.59375	614,689.3	38,418.08	38,400	0.047
33	00000	0	309,090.9	19,318.2	19,200	0.62
33	00110	6/32=0.1875	307,344.6	19,209.04	19,200	0.047
66	00000	0	154,545.5	9659.1	9600	0.62
133	00000	0	76,691.7	4793.2	4800	0.14
266	00000	0	38,345.9	2396.6	2400	0.14
531	00000	0	19,209.0	1200.6	1200	0.11
1062	00000	0	9604.5	600.3	600	0.05
2125	00000	0	4800.0	300.0	300	0.00
4250	00000	0	2400.0	150.0	150	0.00
5795	00000	0	1760.1	110.0	110	0.00

**Table 22-14. Baud Rate Fine Adjust**

BRFA	Baud Rate Fractional Divisor (BRFD)
0 0 0 0 0	0/32 = 0

Table 22-14. Baud Rate Fine Adjust (continued)

BRFA	Baud Rate Fractional Divisor (BRFD)
0 0 0 0 1	$1/32 = 0.03125$
0 0 0 1 0	$2/32 = 0.0625$
0 0 0 1 1	$3/32 = 0.09375$
0 0 1 0 0	$4/32 = 0.125$
0 0 1 0 1	$5/32 = 0.15625$
0 0 1 1 0	$6/32 = 0.1875$
0 0 1 1 1	$7/32 = 0.21875$
0 1 0 0 0	$8/32 = 0.25$
0 1 0 0 1	$9/32 = 0.28125$
0 1 0 1 0	$10/32 = 0.3125$
0 1 0 1 1	$11/32 = 0.34375$
0 1 1 0 0	$12/32 = 0.375$
0 1 1 0 1	$13/32 = 0.40625$
0 1 1 1 0	$14/32 = 0.4375$
0 1 1 1 1	$15/32 = 0.46875$
1 0 0 0 0	$16/32 = 0.5$
1 0 0 0 1	$17/32 = 0.53125$
1 0 0 1 0	$18/32 = 0.5625$
1 0 0 1 1	$19/32 = 0.59375$
1 0 1 0 0	$20/32 = 0.625$
1 0 1 0 1	$21/32 = 0.65625$
1 0 1 1 0	$22/32 = 0.6875$
1 0 1 1 1	$23/32 = 0.71875$
1 1 0 0 0	$24/32 = 0.75$
1 1 0 0 1	$25/32 = 0.78125$
1 1 0 1 0	$26/32 = 0.8125$
1 1 0 1 1	$27/32 = 0.84375$
1 1 1 0 0	$28/32 = 0.875$
1 1 1 0 1	$29/32 = 0.90625$
1 1 1 1 0	$30/32 = 0.9375$
1 1 1 1 1	$31/32 = 0.96875$

## 22.3.3 Transmitter

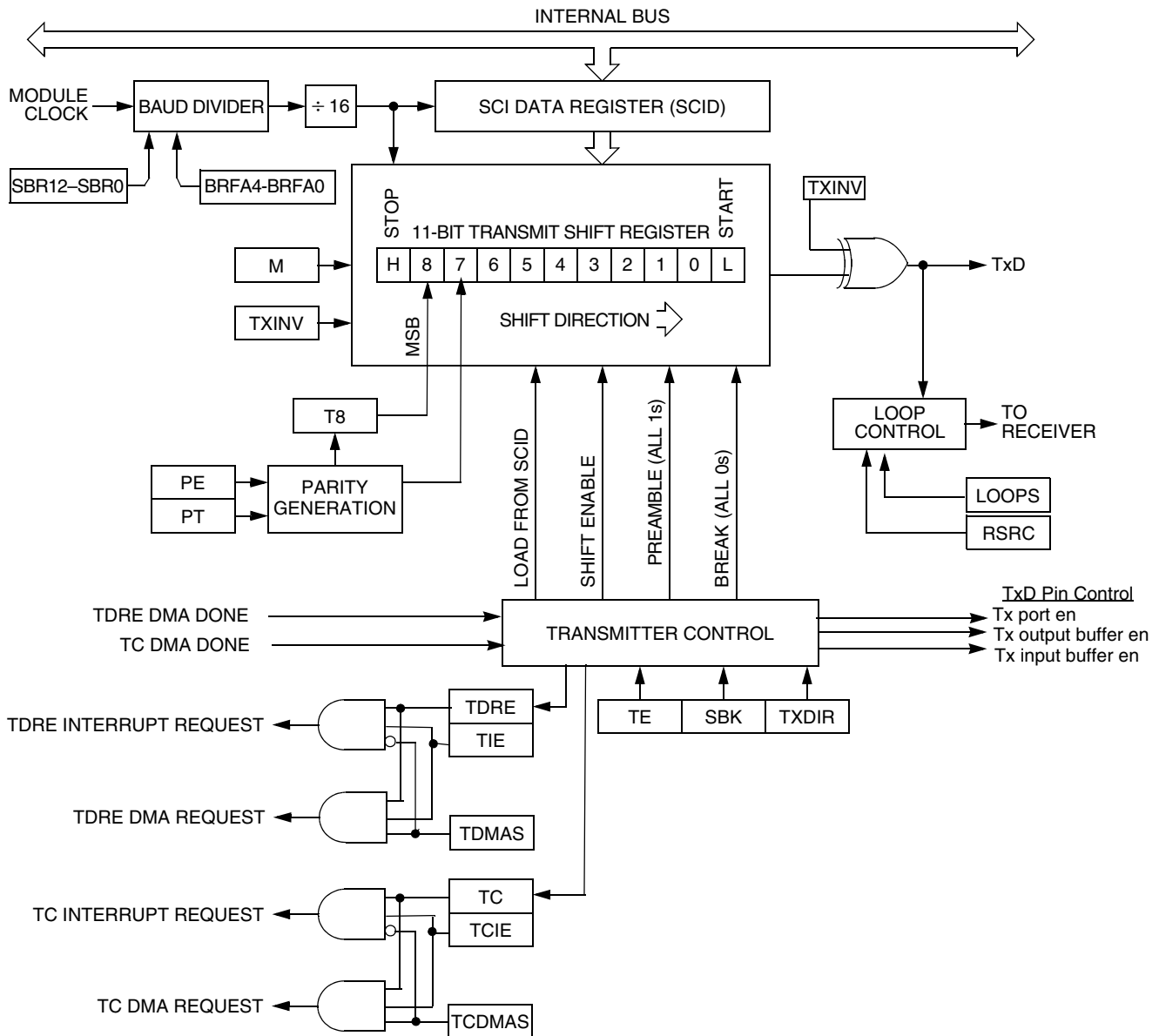


Figure 22-15. Transmitter Block Diagram

### 22.3.3.1 Transmitter Character Length

The SCI transmitter can accommodate either 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCIxCI1) determines the length of data characters. When transmitting 9-bit data, bit T8 in SCI control register 3 (SCIxCI3[6]) is the ninth bit (bit 8).

### 22.3.3.2 Character Transmission

To transmit data, the MCU writes the data bits to the SCI data registers (SCIxC3[6]/SCIxD), which in turn are transferred to the transmitter shift register. The transmit shift register then shifts a frame out through the transmit data output signal after it has prefaced it with a start bit and appended it with a stop bit. The SCI data registers (SCIxC3[6] and SCIxD) are the write-only buffers between the internal data bus and the transmit shift register.

The SCI also sets a flag, the transmit data register empty flag (TDRE), every time it transfers data from the buffer (SCIxD[T8]) to the transmitter shift register. The transmit driver routine may respond to this flag by writing another byte to the Transmitter buffer (SCIxC3[6]/SCIxD), while the shift register is still shifting out the first byte.

To initiate an SCI transmission:

1. Configure the SCI:
  - a) Select a baud rate. Write this value to the SCI baud registers (SCIxBDH/L) to begin the baud rate generator. Remember that the baud rate generator is disabled when the baud rate is zero. Writing to the SCIxBDH has no effect without also writing to SCIxBDL.
  - b) Write to SCIxC1 to configure word length, parity, and other configuration bits (LOOPS, RSRC, M, WAKE, ILT, PE, PT).
  - c) Enable the transmitter, interrupts, receiver, and wakeup as required by writing to the SCIxC2 register bits (TIE, TCIE, RIE, ILIE, TE, RE, RWU, SBK) and SCIxC3 register bits (ORIE, NEIE, PEIE, FEIE). A preamble or idle character is then shifted out of the transmitter shift register.
2. Transmit Procedure for Each Byte:
  - a) Poll the TDRE flag by reading the SCIS1 or responding to the TDRE interrupt. Keep in mind that the TDRE bit resets to one.
  - b) If the TDRE flag is set, write the data to be transmitted to SCIxC3[6]/SCID, where the ninth bit is written to the T8 bit in SCIxC3 if the SCI is in 9-bit data format. A new transmission will not result until the TDRE flag has been cleared.
3. Repeat step 2 for each subsequent transmission.

#### NOTE

The TDRE flag is set when the shift register is loaded with the next data to be transmitted from SCIxC3[6]/SCID, which occurs 9/16ths of a bit time after the start of the stop bit of the previous frame.

Writing the TE bit from 0 to a 1 automatically loads the transmit shift register with a preamble of 10 logic 1s (if M = 0) or 11 logic 1s (if M = 1). After the preamble shifts out, control logic transfers the data from the SCI data register into the transmit shift register. A logic 0 start bit automatically goes into the least significant bit position of the transmit shift register. A logic 1 stop bit goes into the most significant bit position.

Hardware supports odd or even parity. When parity is enabled, the most significant bit (msb) of the data character is the parity bit.

The transmit data register empty flag, TDRE, in SCI status register 1 (SCIxS1) becomes set when the SCI data register transfers a byte to the transmit shift register. The TDRE flag indicates that the SCI data register can accept new data from the internal data bus. If the transmit interrupt enable bit, TIE, in SCI control register 2 (SCIxC2) is also set, the TDRE flag generates a transmitter interrupt request.

When the transmit shift register is not transmitting a frame, the transmit data output signal goes to the idle condition, logic 1. If at any time software clears the TE bit in SCI control register 2 (SCIxC2), the transmitter enable signal goes low and the transmit signal goes idle.

If software clears TE while a transmission is in progress, the frame in the transmit shift register continues to shift out. Then the transmitter enable signal gets de-asserted and the transmit data output signal goes idle even if there is data pending in the SCI data register. To avoid accidentally cutting off the last frame in a message, always wait for TDRE to go high after the last frame before clearing TE.

To separate messages with preambles with minimum idle line time, use this sequence between messages:

1. Write the last byte of the first message to SCIxC3[6]/SCIxD.
2. Wait for the TDRE flag to go high, indicating the transfer of the last frame to the transmit shift register.
3. Queue a preamble by clearing and then setting the TE bit.
4. Write the first byte of the second message to SCIxC3[6]/SCIxD.

### 22.3.3.3 Transmitting Break Characters

Writing a logic 1 to the send break bit, SBK, in SCI control register 2 (SCIxC2) loads the transmit shift register with a break character. A break character contains all logic 0s and has no start, stop, or parity bit. Break character length depends on the M bit in SCI control register 1 (SCIxC1) and the BRK13 bit in SCIxC2. As long as SBK is at logic 1, transmitter logic continuously loads break characters into the transmit shift register. After software clears the SBK bit, the shift register finishes transmitting the last break character and then transmits at least one logic 1. The automatic logic 1 at the end of a break character guarantees the recognition of the start bit of the next frame.

### 22.3.3.4 Idle Characters

An idle character contains all logic 1s and has no start, stop, or parity bit. Idle character length depends on the M bit in SCI control register 1 (SCIxC1). The preamble is a synchronizing idle character that begins the first transmission initiated after writing the TE bit from 0 to 1.

If the TE bit is cleared during a transmission, the transmit data output signal becomes idle after completion of the transmission in progress. Clearing and then setting the TE bit during a transmission queues an idle character to be sent after the frame currently being transmitted.



### NOTE

When queueing an idle character, return the TE bit to logic 1 before the stop bit of the current frame shifts out through the transmit data output signal. Setting TE after the stop bit appears on transmit data output causes data previously written to the SCI data register to be lost. Toggle the TE bit for a queued idle character while the TDRE flag is set and immediately before writing the next byte to the SCI data register.

If the TE bit is cleared and the transmission is completed, the SCI is not the master of the TXD pin.

#### 22.3.3.5 Inversion of Transmitted Output

The transmit inversion bit (TXINV) in SCI control register 3 (SCIxC3) reverses the polarity of transmitted data. All transmitted values, including idle, break, start, and stop bits, are inverted when TXINV is at logic 1.

### 22.3.4 Receiver

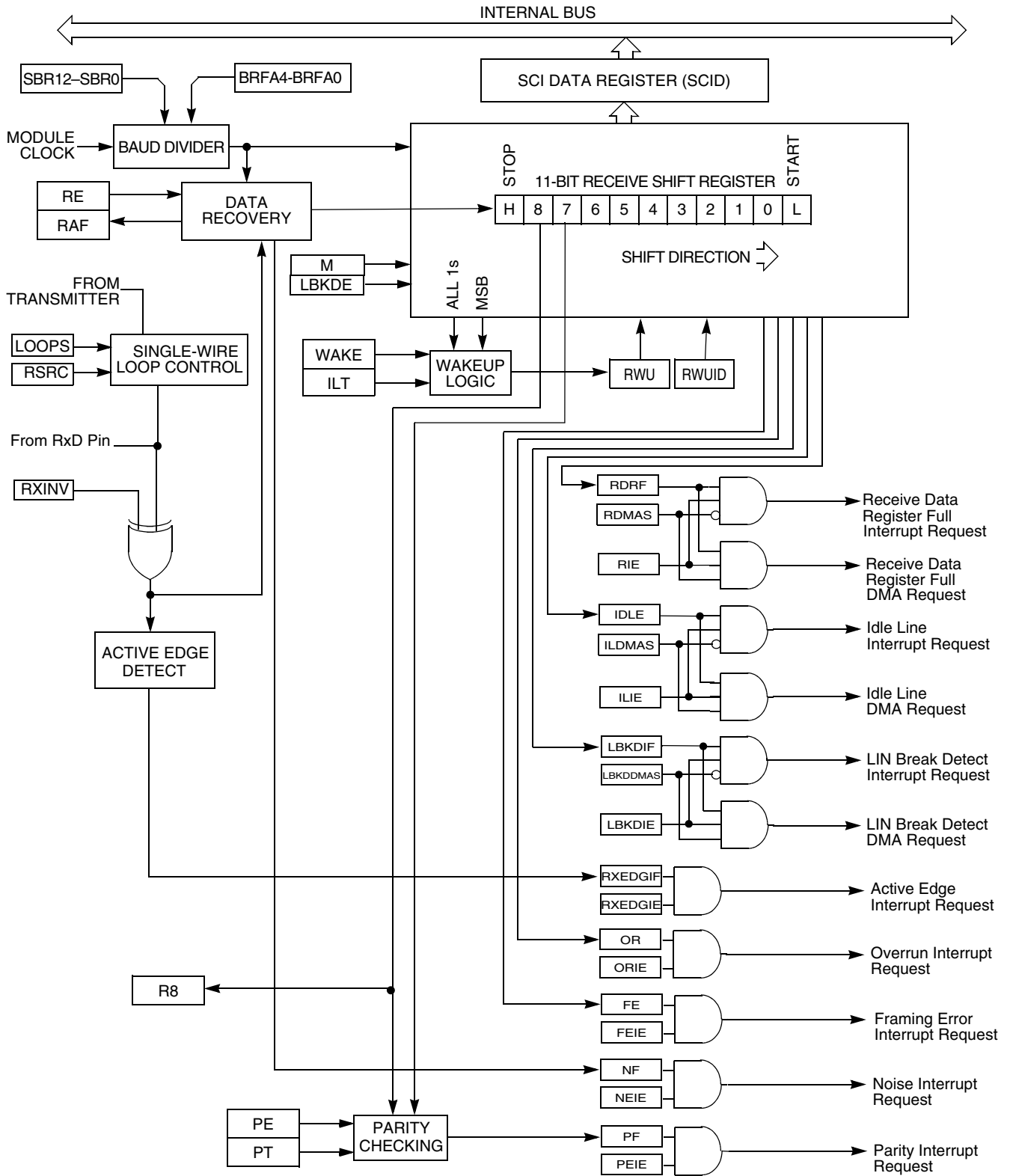


Figure 22-16. SCI Receiver Block Diagram

### 22.3.4.1 Receiver Character Length

The SCI receiver can accommodate 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCIxC1) determines the length of data characters. When receiving 9-bit data, bit R8 in SCI data register high (SCIxC3[6]) is the ninth bit (bit 8).

### 22.3.4.2 Character Reception

During an SCI reception, the receive shift register shifts a frame in from the unsynchronized receiver input signal. The SCI data register is the read-only buffer between the internal data bus and the receive shift register.

After a complete frame shifts into the receive shift register, the data portion of the frame transfers to the SCI data register. The receive data register full flag, RDRF, in SCI status register 1 (SCIxS1) becomes set, indicating that the received byte can be read. If the receive interrupt enable bit, RIE, in SCI control register 2 (SCIxC2) is also set, the RDRF flag generates an RDRF interrupt request.

### 22.3.4.3 Data Sampling

The receiver samples the unsynchronized receiver input signal at the RT clock rate. The RT clock is an internal signal with a frequency 16 times the baud rate. To adjust for baud rate mismatch, the RT clock (see [Figure 22-17](#)) is re-synchronized:

- After every start bit.
- After the receiver detects a data bit change from logic 1 to logic 0 (after the majority of data bit samples at RT8, RT9, and RT10 returns a valid logic 1 and the majority of the next RT8, RT9, and RT10 samples returns a valid logic 0).

To locate the start bit, data recovery logic does an asynchronous search for a logic 0 preceded by three logic 1s. When the falling edge of a possible start bit occurs, the RT clock begins to count to 16.

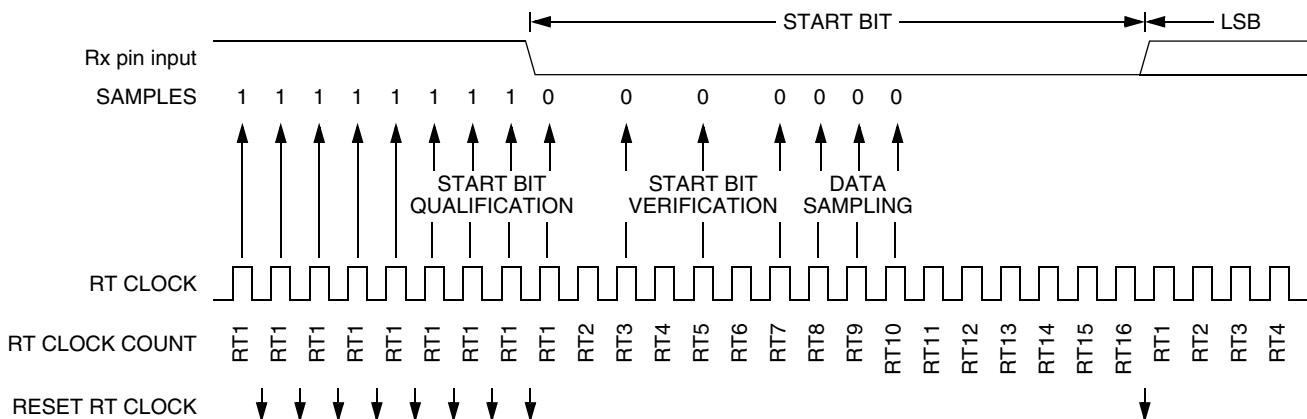


Figure 22-17. Receiver Data Sampling

To verify the start bit and to detect noise, data recovery logic takes samples at RT3, RT5, and RT7. [Table 22-15](#) summarizes the results of the start bit verification samples.

**Table 22-15. Start Bit Verification**

RT3, RT5, and RT7 Samples	Start Bit Verification	Noise Flag
000	Yes	0
001	Yes	1
010	Yes	1
011	No	0
100	Yes	1
101	No	0
110	No	0
111	No	0

If start bit verification is not successful, the RT clock is reset and a new search for a start bit begins.

To determine the value of a data bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. [Table 22-16](#) summarizes the results of the data bit samples.

**Table 22-16. Data Bit Recovery**

RT8, RT9, and RT10 Samples	Data Bit Determination	Noise Flag
000	0	0
001	0	1
010	0	1
011	1	1
100	0	1
101	1	1
110	1	1
111	1	0

**NOTE**

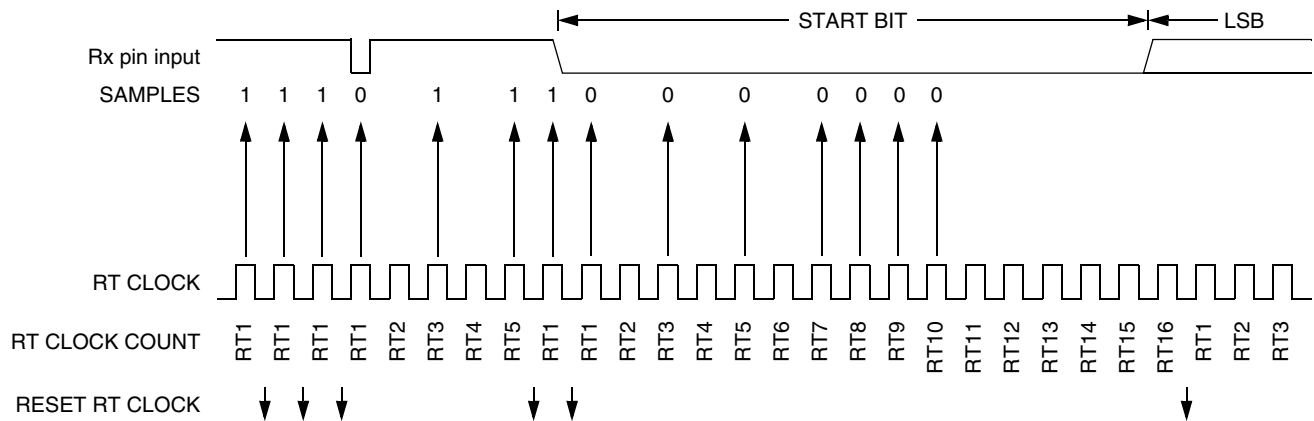
The RT8, RT9, and RT10 samples do not affect start bit verification. If any or all of the RT8, RT9, and RT10 start bit samples are logic 1s following a successful start bit verification, the noise flag (NF) is set and the receiver assumes that the bit is a start bit (logic 0).

To verify a stop bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. [Table 22-17](#) summarizes the results of the stop bit samples.

**Table 22-17. Stop Bit Recovery**

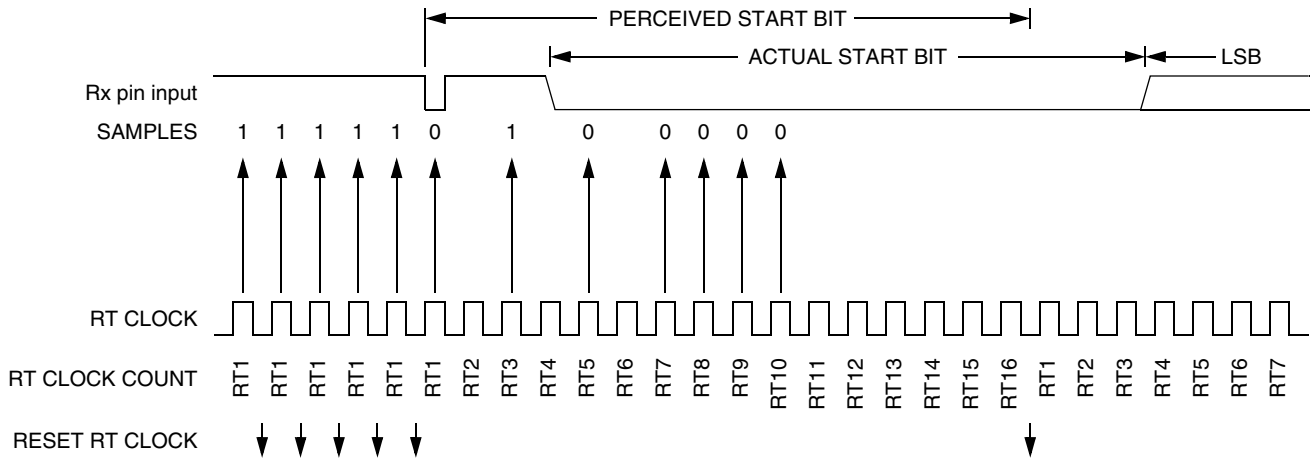
RT8, RT9, and RT10 Samples	Framing Error Flag	Noise Flag
000	1	0
001	1	1
010	1	1
011	0	1
100	1	1
101	0	1
110	0	1
111	0	0

In Figure 22-18, the verification samples RT3 and RT5 determine that the first low detected was noise and not the beginning of a start bit. The RT clock is reset and the start bit search begins again. The noise flag is not set because the noise occurred before the start bit was found.



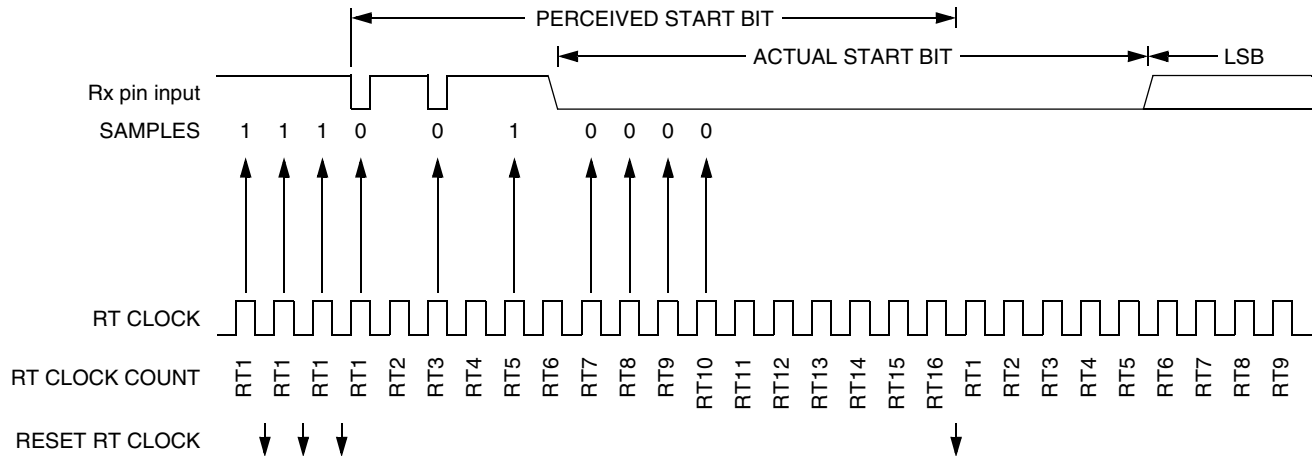
**Figure 22-18. Start Bit Search Example 1**

In Figure 22-19, verification sample at RT3 is high. The RT3 sample sets the noise flag. Although the perceived bit time is misaligned, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.



**Figure 22-19. Start Bit Search Example 2**

In [Figure 22-20](#), a large burst of noise is perceived as the beginning of a start bit, although the test sample at RT5 is high. The RT5 sample sets the noise flag. Although this is a worst-case misalignment of perceived bit time, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.



**Figure 22-20. Start Bit Search Example 3**

[Figure 22-21](#) shows the effect of noise early in the start bit time. Although this noise does not affect proper synchronization with the start bit time, it does set the noise flag.

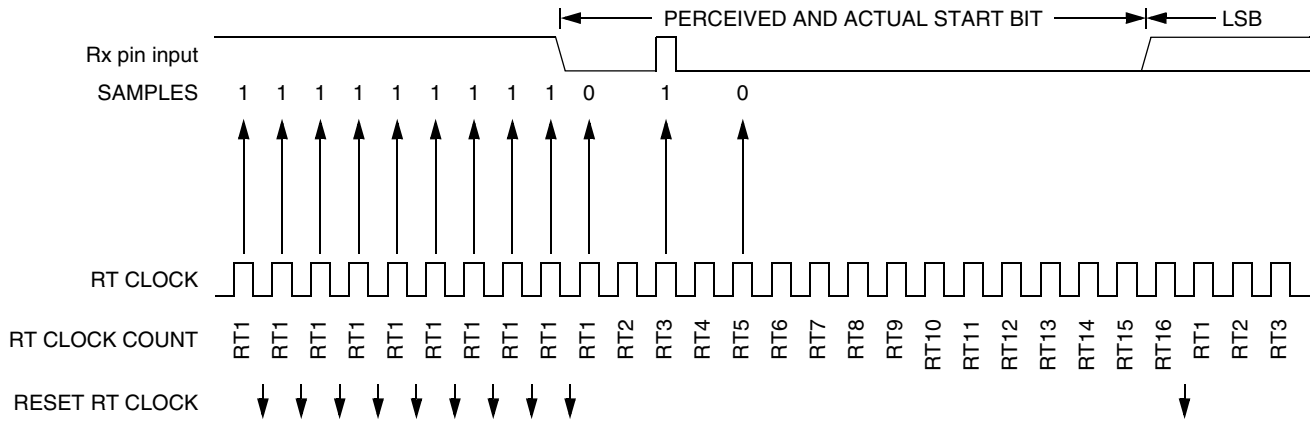


Figure 22-21. Start Bit Search Example 4

Figure 22-22 shows a burst of noise near the beginning of the start bit that resets the RT clock. The sample after the reset is low but is not preceded by three high samples that would qualify as a falling edge. Depending on the timing of the start bit search and on the data, the frame may be missed entirely or it may set the framing error flag.

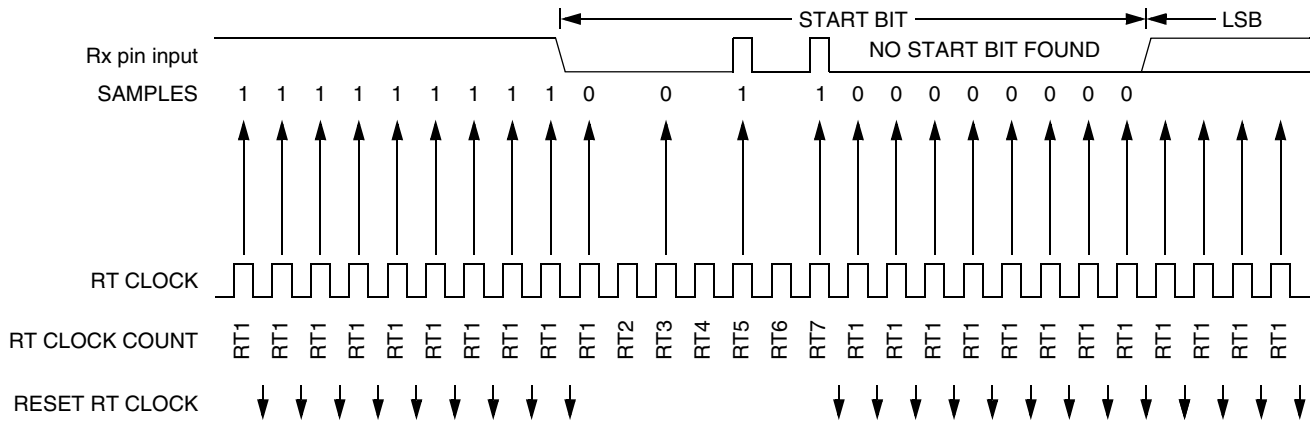


Figure 22-22. Start Bit Search Example 5

In Figure 22-23, a noise burst makes the majority of data samples RT8, RT9, and RT10 high. This sets the noise flag but does not reset the RT clock. In start bits only, the RT8, RT9, and RT10 data samples are ignored.

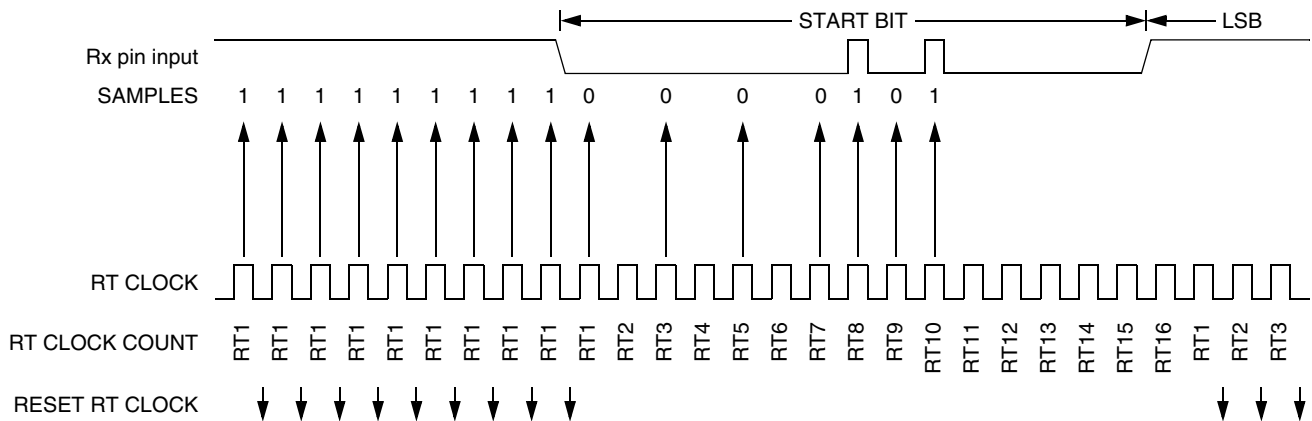


Figure 22-23. Start Bit Search Example 6

#### 22.3.4.4 Framing Errors

If the data recovery logic does not detect a logic 1 where the stop bit should be in an incoming frame, it sets the framing error flag, FE, in SCI status register 1 (SCIxS1) if LBKDE is disabled. A break character when LBKDE is disabled also sets the FE flag because a break character has no stop bit. The FE flag is set at the same time that the RDRF flag is set.

#### 22.3.4.5 Receiving Break Characters

The SCI recognizes a break character when a start bit is followed by eight or nine logic 0 data bits and a logic 0 where the stop bit should be. Receiving a break character has these effects on SCI registers:

- Sets the framing error flag, FE.
- Sets the receive data register full flag, RDRF.
- Clears the SCI data registers (SCIxC3[6]/SCIxD).
- May set the overrun flag, OR, noise flag, NF, parity error flag, PE, or the receiver active flag, RAF (see [Section 22.2.3.4](#) and [Section 22.2.3.5](#)).

When using an internal oscillator in a LIN system, it is necessary to raise the break detection threshold by one bit time. When LIN Break Detect feature is enabled (LBKDE=1) the SCI break detection threshold is changed from 10 bits to 11 bits if M=0 and 11 bits to 12 bits if M=1. While LBKDE is set, it will have these effects on the SCI registers:

- Prevents the RDRF, FE, NF, and PF flags from getting set.
- Sets the LIN Break Detect Interrupt Flag, LBKDIF if a LIN Break Character is received.

#### 22.3.4.6 Inversion of Receiver Input

The receive inversion bit (RXINV) in SCI status register 2 (SCIxS2) reverses the polarity of receive data. All receive values, including idle, break, start, and stop bits, are inverted when RXINV is at logic 1.



### 22.3.4.7 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the actual stop bit. A noise error will occur if the RT8, RT9, and RT10 samples are not all the same logical values. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RT8, RT9, and RT10 stop bit samples are a logic zero.

As the receiver samples an incoming frame, it re-synchronizes the RT clock on any valid falling edge within the frame. Resynchronization within frames will correct a misalignment between transmitter bit times and receiver bit times.

#### 22.3.4.7.1 Slow Data Tolerance

Figure 22-24 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

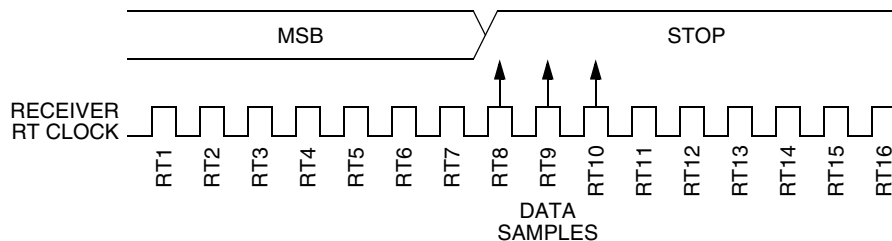


Figure 22-24. Slow Data

For an 8-bit data character, data sampling of the stop bit takes the receiver 154 RT cycles (9 bit times  $\times$  16 RT cycles + 10 RT cycles).

With the misaligned character shown in Figure 22-24, the receiver counts 154 RT cycles at the point when the count of the transmitting device is 147 RT cycles (9 bit times  $\times$  16 RT cycles + 3 RT cycles).

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is:

$$((154 - 147) \div 154) \times 100 = 4.54\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 170 RT cycles (10 bit times  $\times$  16 RT cycles + 10 RT cycles).

With the misaligned character shown in Figure 22-24, the receiver counts 170 RT cycles at the point when the count of the transmitting device is 163 RT cycles (10 bit times  $\times$  16 RT cycles + 3 RT cycles).

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is:

$$((170 - 163) \div 170) \times 100 = 4.12\%$$

### 22.3.4.7.2 Fast Data Tolerance

Figure 22-25 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but is still sampled at RT8, RT9, and RT10.

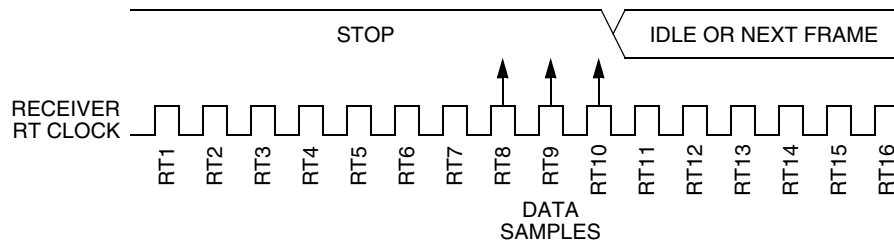


Figure 22-25. Fast Data

For an 8-bit data character, data sampling of the stop bit takes the receiver 154 RT cycles (9 bit times  $\times$  16 RT cycles + 10 RT cycles).

With the misaligned character shown in Figure 22-25, the receiver counts 154 RT cycles at the point when the count of the transmitting device is 160 RT cycles (10 bit times  $\times$  16 RT cycles).

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is:

$$((154 - 160) \div 154) \times 100 = 3.90\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 170 RT cycles (10 bit times  $\times$  16 RT cycles + 10 RT cycles).

With the misaligned character shown in Figure 22-25, the receiver counts 170 RT cycles at the point when the count of the transmitting device is 176 RT cycles (11 bit times  $\times$  16 RT cycles).

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is:

$$((170 - 176) \div 170) \times 100 = 3.53\%$$

### 22.3.4.8 Receiver Wakeup

To enable the SCI to ignore transmissions intended only for other receivers in multiple-receiver systems, the receiver can be put into a standby state. Setting the receiver wakeup bit, RWU, in SCI control register 2 (SCIxC2) puts the receiver into a standby state during which receiver interrupts (with the exception of the idle flag, IDLE, when RWUID bit is set) are inhibited from setting. The SCI will still load the receive data into the SCIxC3[7]/SCIxD registers, but it will not set the RDRF flag.

The transmitting device can address messages to selected receivers by including addressing information in the initial frame or frames of each message.

The WAKE bit in SCI control register 1 (SCIxC1) determines how the SCI is brought out of the standby state to process an incoming message. The WAKE bit enables either idle line wakeup or address mark wakeup.

### 22.3.4.8.1 Idle Input Line Wakeup (WAKE = 0)

In this wakeup method, any idle condition on the unsynchronized receiver input signal clears the RWU bit and wakes the SCI. The initial frame or frames of every message contain addressing information. All receivers evaluate the addressing information, and receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another idle character appears on the unsynchronized receiver input signal.

Idle line wakeup requires that messages be separated by at least one idle character and that no message contains idle characters.

When RWU is one and RWUID is zero, the idle character that wakes the receiver does not set the IDLE flag or the receive data register full flag, RDRF. The receiver wakes and waits for the first data character of the next message which will set the RDRF flag and generate an interrupt if enabled. When RWUID and RWU bits are set and WAKE is cleared, any idle condition sets the IDLE flag and generates an interrupt if enabled.

#### NOTE

With the WAKE bit clear, setting the RWU bit after the unsynchronized receiver input signal has been idle can cause the receiver to wake immediately.

### 22.3.4.8.2 Address Mark Wakeup (WAKE = 1)

In this wakeup method, a logic 1 in the most significant bit (msb) position of a frame clears the RWU bit and wakes the SCI. The logic 1 in the msb position marks a frame as an address frame that contains addressing information. All receivers evaluate the addressing information, and the receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the unsynchronized receiver input signal.

The logic 1 msb of an address frame clears the receiver's RWU bit before the stop bit is received and sets the RDRF flag.

Address mark wakeup allows messages to contain idle characters but requires that the msb be reserved for use in address frames.

If module is in standby mode and nothing triggers to wake the SCI, no error flag is set even though an invalid error condition is detected on the receiving data line.

#### NOTE

With the WAKE bit clear, setting the RWU bit after the unsynchronized receiver input signal has been idle can cause the receiver to wakeup immediately.

### 22.3.4.9 Match Address Operation

Match address operation provides a means to ignore transmissions intended only for other receivers, in cases where the address mark wakeup method is used, without interrupting the CPU to determine if the marked address is valid.

Match Address operation is enabled when the MAEN1 or MAEN2 bit is asserted. In this function, a frame received by the RX pin with a logic 1 in the most significant bit (msb) position is considered an address and is compared with the associated SCIxMA1 or SCIxMA2 register. The frame is only transferred to the receive buffer, and RDRF is set, if the comparison matches. All subsequent frames received with a logic 0 in the most significant bit positions are considered to be data associated with the address and are transferred to the receive buffer, and the RDRF flag is set. If no marked address (msb = 1) match occurs then no transfer is made to the receive buffer, RDRF is not set, and all following frames with logic zero in the msb position are also discarded. If both the MAEN1 and MAEN2 bits are negated, the receiver operates normally and all data received is transferred to the receive buffer, and RDRF is set.

Match Address operation functions in the same way for both SCIxMA1 and SCIxMA2. The two match address registers allow a second match address function for a broadcast or general call address to the serial bus, as an example.

When both the MAEN1 and MAEN2 bits are negated, no address comparisons are performed and all data received is transferred to the receive buffer, and RDRF is set.

- If only one of MAEN1 and MAEN2 is asserted, a marked address (msb = 1) is compared only with the associated match register and data is transferred and RDRF set only on a match.
- If MAEN1 and MAEN2 are asserted, a marked address is compared with both match registers and data is transferred only on a match with either register, and RDRF is set.

### 22.3.5 Single-Wire Operation

Normally, the SCI uses two pins for transmitting and receiving. In single-wire operation, the RXD pin is disconnected from the SCI and the SCI implements a half-duplex serial connection. The SCI uses the TXD pin for both receiving and transmitting.

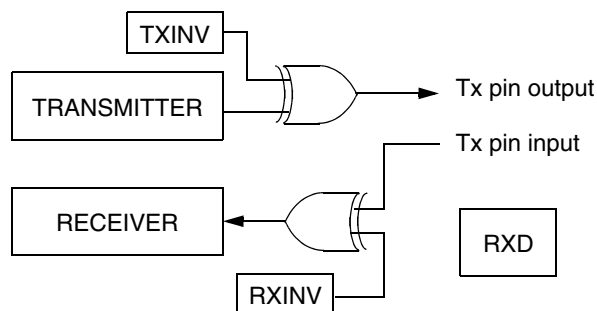
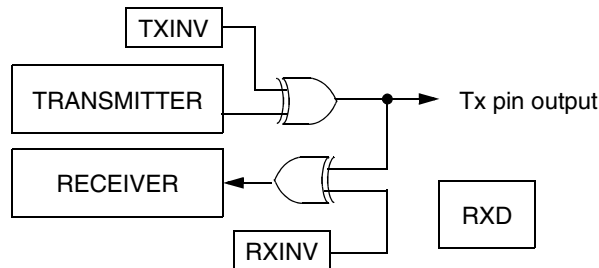


Figure 22-26. Single-Wire Operation (LOOPS = 1, RSRC = 1)

Enable single-wire operation by setting the LOOPS bit and the receiver source bit, RSRC, in SCI control register 1 (SCIxCI1). Setting the LOOPS bit disables the path from the unsynchronized receiver input signal to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver. Both the transmitter and receiver must be enabled (TE=1 and RE=1).

## 22.3.6 Loop Operation

In loop operation the transmitter output goes to the receiver input. The unsynchronized receiver input signal is disconnected from the SCI.



**Figure 22-27. Loop Operation (LOOPS = 1, RSRC = 0)**

Enable loop operation by setting the LOOPS bit and clearing the RSRC bit in SCI control register 1 (SCIxC1). Setting the LOOPS bit disables the path from the unsynchronized receiver input signal to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).

## 22.4 Reset

The reset values of registers and signals are described in [Chapter 4, “Memory”](#). All registers reset to a particular value are indicated in [Section 22.2](#).

## 22.5 Interrupts

This section describes interrupts originated by the SCI block. The MCU must service the interrupt requests. Interrupt sources can be grouped into categories by ORing them together at the SoC integration level; for example, receiver interrupt, transmitter interrupt and interrupt due to various error flags.

### 22.5.1 System Level Interrupt Sources

There are 3 interrupt signals that are sent from the SCI to the CPU. The interrupt sources are listed in [Table 22-18](#).

**Table 22-18. SCI Interrupt Sources**

Interrupt Source	Flag	Local Enable	DMA Select	Interrupt Signal
Transmitter	TDRE	TIE	TDMAS=0	SCI_tx
Transmitter	TC	TCIE		
Receiver	IDLE	ILIE		SCI_rx
Receiver	RDRF	RIE	RDMAS=0	
Receiver	LBKDIF	LBKDIE		
Receiver	RXEDGIF	RXEDGIE		

**Table 22-18. SCI Interrupt Sources (continued)**

Interrupt Source	Flag	Local Enable	DMA Select	Interrupt Signal
Receiver	OR	ORIE		SCI_err
Receiver	NF	NEIE		
Receiver	FE	FEIE		
Receiver	PF	PEIE		

### 22.5.1.1 Recovery from Wait Mode

Any of the SCI interrupt requests listed in [Table 22-18](#) can be used to bring the CPU out of wait mode.

## 22.5.2 Description of Interrupt Operation

The SCI only originates interrupt requests. The following is a description of how the SCI makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt number are chip dependent.

### 22.5.2.1 TDRE Description

The TDRE interrupt is set high by the SCI when the transmit shift register receives a byte from the SCI data register. A TDRE interrupt indicates that the transmit data register (SCIxC3[6]/SCIxD) is empty and that a new byte can be written to the SCIxC3[6]/SCIxD for transmission. Clear TDRE by reading SCI status register 1 with TDRE set and then writing to SCI data register (SCIxD).

### 22.5.2.2 TC Description

The TC interrupt is set by the SCI when a transmission has been completed. A TC interrupt indicates that there is no transmission in progress. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD pin becomes idle (logic 1). Clear TC by reading SCI status register 1 (SCIxS1) with TC set and then writing to SCI data register (SCIxD). TC is cleared automatically when data, preamble, or break is queued and ready to be sent.

### 22.5.2.3 RDRF Description

The RDRF interrupt is set when the data in the receive shift register transfers to the SCI data register but does not get set while LBKDE is set to one. A RDRF interrupt indicates that the received data has been transferred to the SCI data register and that the byte can now be read by the MCU. To clear the RDRF interrupt read the SCI status register 1 (SCIxS1) and then read SCI data register (SCIxD). If RIE and RDMAS are both set, then the RDRF flag will not be cleared by this mechanism.

### 22.5.2.4 OR Description

The OR interrupt is set when software fails to read the SCI data register before the receive shift register receives the next frame. The newly acquired data in the shift register will be lost in this case, but the data

already in the SCI data registers is not affected. The OR interrupt is cleared by reading the SCI status register one (SCIxS1) and then reading SCI data register (SCIxD).

### 22.5.2.5 IDLE Description

The IDLE interrupt is set when 10 consecutive logic 1s (if M=0) or 11 consecutive logic 1s (if M=1) appear on the receiver input. Once the IDLE is cleared, a valid frame must again set the RDRF flag or a LIN break character must set the LBKDIF flag before an idle condition can set the IDLE flag. Clear IDLE by reading SCI status register 1 (SCIxS1) with IDLE set and then reading SCI data register (SCIxD). If ILE and ILDMAS are both set, then the IDLE flag will not be cleared by this mechanism.

### 22.5.2.6 NF Description

The NF interrupt is set when SCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun or while the LBKDE bit is set to one. The NF interrupt is cleared by reading the SCI status register one (SCIxS1) and then reading SCI data register (SCIxD).

### 22.5.2.7 FE Description

The FE interrupt is set when logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun or while LBKDE bit is set to one. The FE interrupt is cleared by reading the SCI status register one (SCIxS1) and then reading SCI data register (SCIxD).

### 22.5.2.8 PF Description

The PF interrupt is set when the parity enable bit, PE, is set and the parity of the received data does not match its parity bit. PF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun or while LBKDE bit is set to one. To clear PF interrupt, read the SCI status register one (SCIxS1) and then read the SCI data register (SCIxD).

### 22.5.2.9 LBKDIF Description

The LBKDIF is set when LIN Break Detect circuitry is enabled (LBKDE=1) and a LIN break character is detected. LBKDIF is set right after receiving the last LIN break character bit. The LBKDIF interrupt is cleared by writing a 1 to it.

### 22.5.2.10 RXEDGIF Description

The RXEDGIF is set when an active edge is detected on the RxD pin.

#### NOTE

A RXEDGIF interrupt is generated only when RXEDGIE is set. If RXEDGIE is not enabled prior to RXEDGIF getting set, an interrupt is not generated until RXEDGIE bit gets written to 1.

### 22.5.2.10.1 RxD Edge Detect Sensitivity

Edge sensitive can be software programmed to be either falling or rising. The polarity of the edge sensitivity is selected using the RXINV bit in the SCI status register 2 (SCIxS2). To detect falling edge RXINV is programmed to zero and to detect rising edge RXINV is programmed to one.

Synchronous logic is used to detect edges. Prior to detecting an edge, RxD input must be at the de-asserted logic level. A falling edge is detected when RxD input signal is seen as a logic 1 (the deasserted level) during one bus cycle and then a logic 0 (the asserted level) during the next cycle. A rising edge is detected when the input is seen as a logic 0 during one bus cycle and then a logic 1 during the next cycle.

### 22.5.2.10.2 Clearing RXEDGIF Interrupt Request

Writing a logic 1 to the RXEDGIF bit in the SCIxS2 register immediately clears the RXEDGIF interrupt request even if the RxD pin remains asserted. RXEDGIF will remain set if another active edge is detected on RxD pin while attempting to clear the RXEDGIF flag by writing a 1 to it.

## 22.5.3 Exit from Low-Power Modes

The receive input active edge detect circuit is still active on low power modes (wait and stop). An active edge on the receive input brings the CPU out of low power mode if the interrupt is not masked (RXEDGIE=1).



## 22.6 DMA Operation

In the transmitter, flag TDRE can be configured to assert a DMA transfer request. In the receiver, flag RDRF can be configured to assert a DMA transfer request. [Table 22-19](#) shows the configuration bit settings required to configure each flag for DMA operation.

**Table 22-19. DMA Configuration**

Flag	Request Enable Bit	DMA Select Bit
TDRE	TIE=1	TDMAS=1
RDRF	RIE=1	RDMAS=1

When a flag is configured for a DMA request, its associated DMA request is asserted when the flag is set. When the RDRF flag is configured as a DMA request, the clearing mechanism of reading SCIxS1 followed by reading SCIxD does not clear the associated flag. The DMA request remains asserted until an indication is received that the DMA transactions are done. When this indication is received, the flag bit and the associated DMA request are cleared.

# Chapter 23

## Serial Peripheral Interface (S08SPIV6)

### 23.1 Introduction

The serial peripheral interface (SPI) module provides for full-duplex, synchronous, serial communication between the MCU and peripheral devices. These peripheral devices can include other microcontrollers, analog-to-digital converters, shift registers, sensors, memories, etc.

The SPI runs at a baud rate up to the bus clock divided by two in master mode and up to the bus clock divided by 4 in slave mode. Software can poll the status flags, or SPI operation can be interrupt driven.

The SPI also supports a data length of 8 or 16 bits and includes a hardware match feature for the receive data buffer.

The SPI includes DMA interface inside to support continuous SPI transmission through on-chip DMA controller instead of CPU so that CPU loading is decreased and CPU time saved can be utilized for other work.

#### 23.1.1 Features

The SPI includes these distinctive features:

- Master mode or slave mode operation
- Full-duplex or single-wire bidirectional mode
- Programmable transmit bit rate
- Double-buffered transmit and receive data register
- Serial clock phase and polarity options
- Slave select output
- Mode fault error flag with CPU interrupt capability
- Control of SPI operation during wait mode
- Selectable MSB-first or LSB-first shifting
- Programmable 8- or 16-bit data transmission length
- Receive data buffer hardware match feature
- Support transmission of both Transmit and Receive by DMA

#### 23.1.2 Modes of Operation

The SPI functions in three modes, run, wait, and stop.

- Run Mode

This is the basic mode of operation.

- Wait Mode

SPI operation in wait mode is a configurable low power mode, controlled by the SPISWAI bit located in the SPIxC2 register. In wait mode, if the SPISWAI bit is clear, the SPI operates like in Run Mode. If the SPISWAI bit is set, the SPI goes into a power conservative state, with the SPI clock generation turned off. If the SPI is configured as a master, any transmission in progress stops, but is resumed after CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a byte continues, so that the slave stays synchronized to the master.

- Stop Mode

The SPI is inactive in stop3/stop4 mode for reduced power consumption. If the SPI is configured as a master, any transmission in progress stops, but is resumed after the CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a data continues, so that the slave stays synchronized to the master.

The SPI is completely disabled in all other stop modes. When the CPU wakes from these stop modes, all SPI register content will be reset.

This is a high level description only, detailed descriptions of operating modes are contained in section [Section 23.4.10, “Low Power Mode Options.”](#)

### 23.1.3 Block Diagrams

This section includes block diagrams showing SPI system connections, the internal organization of the SPI module, and the SPI clock dividers that control the master mode bit rate.

#### 23.1.3.1 SPI System Block Diagram

[Figure 23-1](#) shows the SPI modules of two MCUs connected in a master-slave arrangement. The master device initiates all SPI data transfers. During a transfer, the master shifts data out (on the MOSI pin) to the slave while simultaneously shifting data in (on the MISO pin) from the slave. The transfer effectively exchanges the data that was in the SPI shift registers of the two SPI systems. The SPSCK signal is a clock output from the master and an input to the slave. The slave device must be selected by a low level on the slave select input ( $\overline{SS}$  pin). In this system, the master device has configured its  $\overline{SS}$  pin as an optional slave select output.

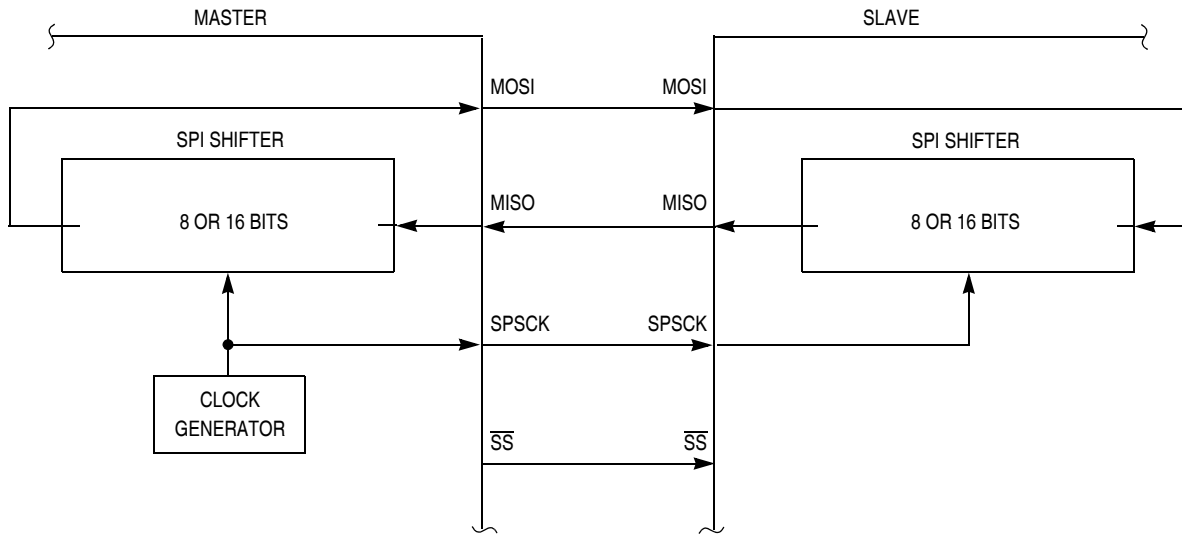


Figure 23-1. SPI System Connections

### 23.1.3.2 SPI Module Block Diagram

Figure 23-2 is a block diagram of the SPI module. The central element of the SPI is the SPI shift register. Data is written to the double-buffered transmitter (write to SPIxDH:SPIxDL) and gets transferred to the SPI shift register at the start of a data transfer. After shifting in 8 or 16 bits (as determined by SPI MODE bit) of data, the data is transferred into the double-buffered receiver where it can be read (read from SPIxDH:SPIxDL). Pin multiplexing logic controls connections between MCU pins and the SPI module.

When the SPI is configured as a master, the clock output is routed to the SPSCCK pin, the shifter output is routed to MOSI, and the shifter input is routed from the MISO pin.

When the SPI is configured as a slave, the SPSCCK pin is routed to the clock input of the SPI, the shifter output is routed to MISO, and the shifter input is routed from the MOSI pin.

In the external SPI system, simply connect all SPSCCK pins to each other, all MISO pins together, and all MOSI pins together. Peripheral devices often use slightly different names for these pins.

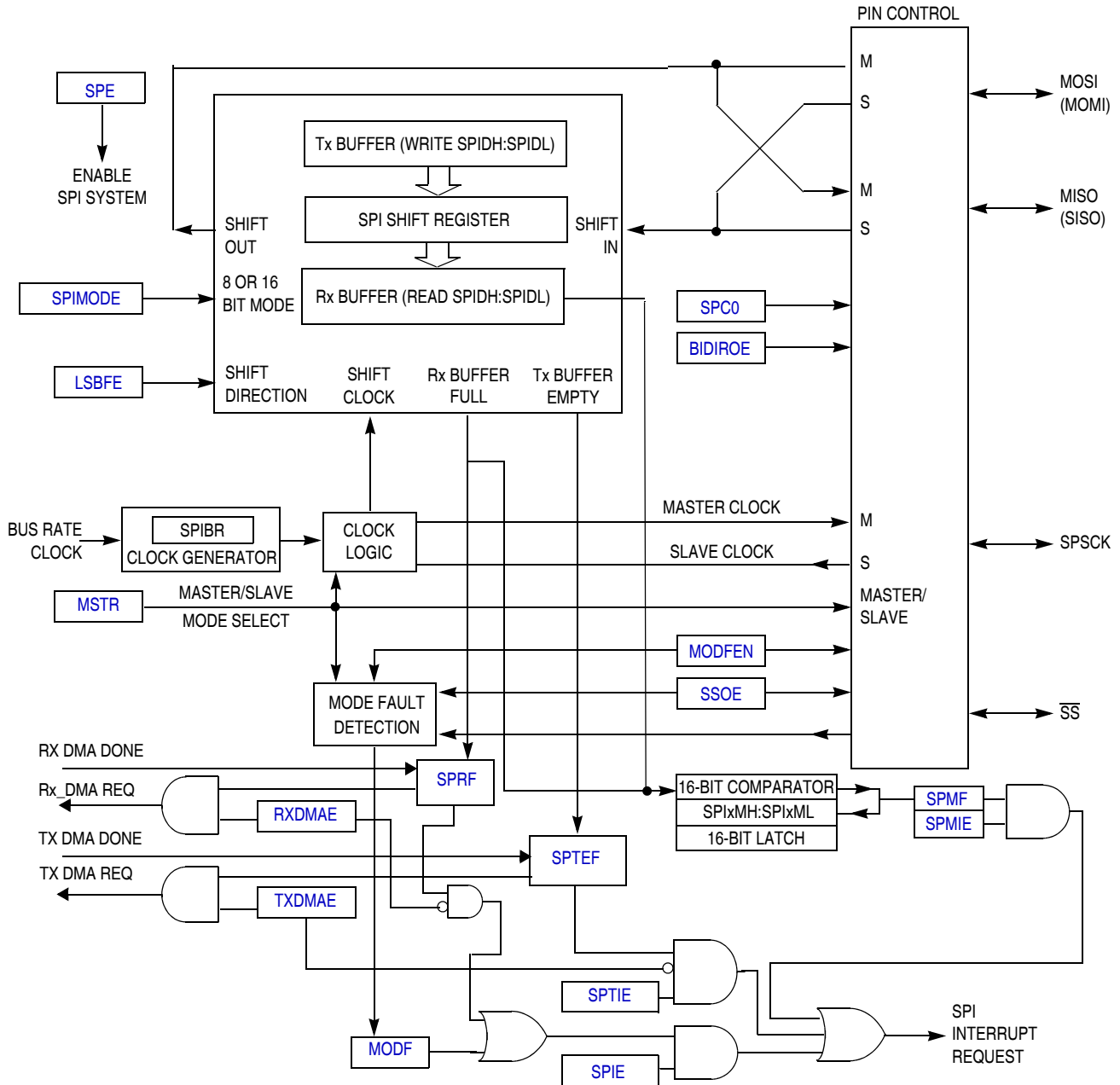


Figure 23-2. SPI Module Block Diagram

## 23.2 External Signal Description

The SPI optionally shares four port pins. The function of these pins depends on the settings of SPI control bits. When the SPI is disabled ( $SPE = 0$ ), these four pins revert to being general-purpose port I/O pins that are not controlled by the SPI.

### 23.2.1 SPCK — SPI Serial Clock

When the SPI is enabled as a slave, this pin is the serial clock input. When the SPI is enabled as a master, this pin is the serial clock output.

### 23.2.2 MOSI — Master Data Out, Slave Data In

When the SPI is enabled as a master and SPI pin control zero (SPC0) is 0 (not bidirectional mode), this pin is the serial data output. When the SPI is enabled as a slave and SPC0 = 0, this pin is the serial data input. If SPC0 = 1 to select single-wire bidirectional mode, and master mode is selected, this pin becomes the bidirectional data I/O pin (MOMI). Also, the bidirectional mode output enable bit determines whether the pin acts as an input (BIDIROE = 0) or an output (BIDIROE = 1). If SPC0 = 1 and slave mode is selected, this pin is not used by the SPI and reverts to being a general-purpose port I/O pin.

### 23.2.3 MISO — Master Data In, Slave Data Out

When the SPI is enabled as a master and SPI pin control zero (SPC0) is 0 (not bidirectional mode), this pin is the serial data input. When the SPI is enabled as a slave and SPC0 = 0, this pin is the serial data output. If SPC0 = 1 to select single-wire bidirectional mode, and slave mode is selected, this pin becomes the bidirectional data I/O pin (SISO) and the bidirectional mode output enable bit determines whether the pin acts as an input (BIDIROE = 0) or an output (BIDIROE = 1). If SPC0 = 1 and master mode is selected, this pin is not used by the SPI and reverts to being a general-purpose port I/O pin.

### 23.2.4 $\overline{SS}$ — Slave Select

When the SPI is enabled as a slave, this pin is the low-true slave select input. When the SPI is enabled as a master and mode fault enable is off (MODFEN = 0), this pin is not used by the SPI and reverts to being a general-purpose port I/O pin. When the SPI is enabled as a master and MODFEN = 1, the slave select output enable bit determines whether this pin acts as the mode fault input (SSOE = 0) or as the slave select output (SSOE = 1).

## 23.3 Register Definition

The SPI has above 8-bit registers to select SPI options, control baud rate, report SPI status, hold an SPI data match value, and for transmit/receive data.

Refer to the direct-page register summary in the Memory chapter of this data sheet for the absolute address assignments for all SPI registers. This section refers to registers and control bits only by their names, and a Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

### 23.3.1 SPI Control Register 1 (SPIxC1)

This read/write register includes the SPI enable control, interrupt enables, and configuration options.

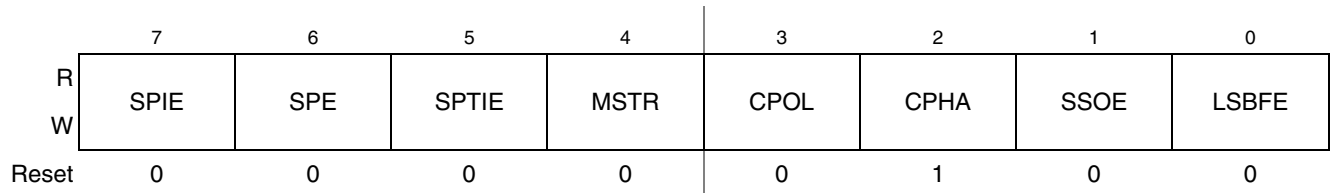


Figure 23-3. SPI Control Register 1 (SPIxC1)

Table 23-1. SPIxC1 Field Descriptions

Field	Description
7 SPIE	<b>SPI Interrupt Enable (for SPRF and MODF)</b> — This is the interrupt enable for SPI receive buffer full (SPRF) and mode fault (MODF) events. 0 Interrupts from SPRF and MODF inhibited (use polling) 1 When SPRF or MODF is 1, request a hardware interrupt
6 SPE	<b>SPI System Enable</b> — This bit enables the SPI system and dedicates the SPI port pins to SPI system functions. If SPE is cleared, SPI is disabled and forced into idle state, and all status bits in the SPIxS register are reset. 0 SPI system inactive 1 SPI system enabled
5 SPTIE	<b>SPI Transmit Interrupt Enable</b> — This is the interrupt enable bit for SPI transmit buffer empty (SPTEF). An interrupt occurs when the SPI transmit buffer is empty (SPTEF is set) 0 Interrupts from SPTEF inhibited (use polling) 1 When SPTEF is 1, hardware interrupt requested
4 MSTR	<b>Master/Slave Mode Select</b> — This bit selects master or slave mode operation. 0 SPI module configured as a slave SPI device 1 SPI module configured as a master SPI device
3 CPOL	<b>Clock Polarity</b> — This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values. This bit effectively places an inverter in series with the clock signal from a master SPI or to a slave SPI device. Refer to <a href="#">Section 23.4.6, “SPI Clock Formats”</a> for more details. 0 Active-high SPI clock (idles low) 1 Active-low SPI clock (idles high)
2 CPHA	<b>Clock Phase</b> — This bit selects one of two clock formats for different kinds of synchronous serial peripheral devices. Refer to <a href="#">Section 23.4.6, “SPI Clock Formats”</a> for more details. 0 First edge on SPSCCK occurs at the middle of the first cycle of a data transfer 1 First edge on SPSCCK occurs at the start of the first cycle of a data transfer
1 SSOE	<b>Slave Select Output Enable</b> — This bit is used in combination with the mode fault enable (MODFEN) bit in SPIxC2 and the master/slave (MSTR) control bit to determine the function of the $\overline{SS}$ pin as shown in <a href="#">Table 23-2</a> .
0 LSBFE	<b>LSB First (Shifter Direction)</b> — This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7 (or bit 15 in 16-bit mode). 0 SPI serial data transfers start with most significant bit 1 SPI serial data transfers start with least significant bit

Table 23-2.  $\overline{SS}$  Pin Function

MODFEN	SSOE	Master Mode	Slave Mode
0	0	General-purpose I/O (not SPI)	Slave select input
0	1	General-purpose I/O (not SPI)	Slave select input
1	0	$\overline{SS}$ input for mode fault	Slave select input
1	1	Automatic $\overline{SS}$ output	Slave select input

### 23.3.2 SPI Control Register 2 (SPIx2)

This read/write register is used to control optional features of the SPI system. Bits 5 and 2 are not implemented and always read 0.

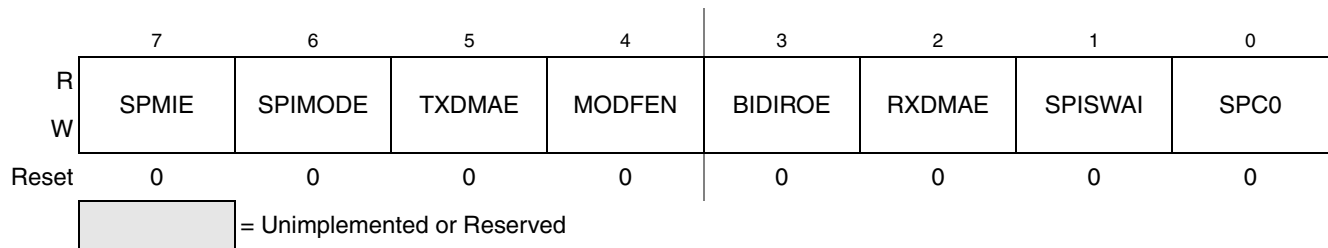


Figure 23-4. SPI Control Register 2 (SPIx2)

Table 23-3. SPIx2 Register Field Descriptions

Field	Description
7 SPMIE	<b>SPI Match Interrupt Enable</b> — This is the interrupt enable for the SPI receive data buffer hardware match (SPMF) function. 0 Interrupts from SPMF inhibited (use polling). 1 When SPMF = 1, requests a hardware interrupt.
6 SPIMODE	<b>SPI 8- or 16-bit Mode</b> — This bit allows the user to select either an 8-bit or 16-bit SPI data transmission length. In master mode, a change of this bit will abort a transmission in progress, force the SPI system into idle state, and reset all status bits in the SPIxS register. Refer to section <a href="#">Section 23.4.5, “Data Transmission Length,”</a> for details. 0 8-bit SPI shift register, match register, and buffers. 1 16-bit SPI shift register, match register, and buffers.
5 TXDMAE	<b>Transmit DMA Enable</b> — This is the enable for Transmit DMA request. If enabled, Transmit DMA request will be asserted when both SPTEF and SPE are set. Meanwhile interrupt from SPTEF will be disabled if this bit is set. 0 DMA request for Transmit is disabled and interrupt from SPTEF is allowed. 1 DMA request for Transmit is enabled and interrupt from SPTEF is disabled
4 MODFEN	<b>Master Mode-Fault Function Enable</b> — When the SPI is configured for slave mode, this bit has no meaning or effect. (The $\overline{SS}$ pin is the slave select input.) In master mode, this bit determines how the $\overline{SS}$ pin is used (refer to <a href="#">Table 23-2</a> for details) 0 Mode fault function disabled, master $\overline{SS}$ pin reverts to general-purpose I/O not controlled by SPI 1 Mode fault function enabled, master $\overline{SS}$ pin acts as the mode fault input or the slave select output



**Table 23-3. SPIx2 Register Field Descriptions (continued)**

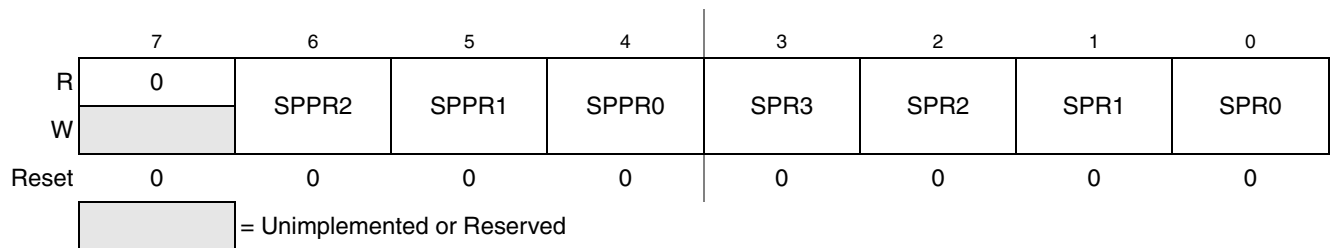
Field	Description
3 BIDIROE	<b>Bidirectional Mode Output Enable</b> — When bidirectional mode is enabled by SPI pin control 0 (SPC0) = 1, BIDIROE determines whether the SPI data output driver is enabled to the single bidirectional SPI I/O pin. Depending on whether the SPI is configured as a master or a slave, it uses either the MOSI (MOMI) or MISO (SISO) pin, respectively, as the single SPI data I/O pin. When SPC0 = 0, BIDIROE has no meaning or effect. 0 Output driver disabled so SPI data I/O pin acts as an input 1 SPI I/O pin enabled as an output
2 RXDMAE	<b>Receive DMA Enable</b> — This is the enable for Receive DMA request. If enabled, Receive DMA request will be asserted when both SPRF and SPE are set. Meanwhile interrupt from SPRF will be disabled if this bit is set. 0 DMA request for Receive is disabled and interrupt from SPRF is allowed. 1 DMA request for Receive is enabled and interrupt from SPRF is disabled
1 SPISWAI	<b>SPI Stop in Wait Mode</b> — This bit is used for power conservation while in wait. 0 SPI clocks continue to operate in wait mode 1 SPI clocks stop when the MCU enters wait mode
0 SPC0	<b>SPI Pin Control 0</b> — This bit enables bidirectional pin configurations as shown in <a href="#">Table 23-4</a> . 0 SPI uses separate pins for data input and data output. 1 SPI configured for single-wire bidirectional operation.

**Table 23-4. Bidirectional Pin Configurations**

Pin Mode	SPC0	BIDIROE	MISO	MOSI
<b>Master Mode of Operation</b>				
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
<b>Slave Mode of Operation</b>				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

### 23.3.3 SPI Baud Rate Register (SPIxBR)

This register is used to set the prescaler and bit rate divisor for an SPI master. This register may be read or written at any time.



**Figure 23-5. SPI Baud Rate Register (SPIxBR)**

**Table 23-5. SPIxBR Register Field Descriptions**

Field	Description
6:4 SPPR[2:0]	<b>SPI Baud Rate Prescaler Divisor</b> — This 3-bit field selects one of eight divisors for the SPI baud rate prescaler as shown in <a href="#">Table 23-6</a> . The input to this prescaler is the bus rate clock (BUSCLK). The output of this prescaler drives the input of the SPI baud rate divider (see <a href="#">Figure 23-15</a> ). See <a href="#">Section 23.4.7, “SPI Baud Rate Generation,”</a> for details.
3:0 SPR[3:0]	<b>SPI Baud Rate Divisor</b> — This 4-bit field selects one of nine divisors for the SPI baud rate divider as shown in <a href="#">Table 23-7</a> . The input to this divider comes from the SPI baud rate prescaler (see <a href="#">Figure 23-15</a> ). See <a href="#">Section 23.4.7, “SPI Baud Rate Generation,”</a> for details.

**Table 23-6. SPI Baud Rate Prescaler Divisor**

SPPR2:SPPR1:SPPR0	Prescaler Divisor
0:0:0	1
0:0:1	2
0:1:0	3
0:1:1	4
1:0:0	5
1:0:1	6
1:1:0	7
1:1:1	8

**Table 23-7. SPI Baud Rate Divisor**

SPR3:SPR2:SPR1:SPR0	Rate Divisor
0:0:0:0	2
0:0:0:1	4
0:0:1:0	8
0:0:1:1	16
0:1:0:0	32
0:1:0:1	64
0:1:1:0	128
0:1:1:1	256
1:0:0:0	512
All other combinations	Reserved

### 23.3.4 SPI Status Register (SPIxS)

This register has four read-only status bits. Bits 3 through 0 are not implemented and always read 0. Writes have no meaning or effect.

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	0	0	0	0
W								
Reset	0	0	1	0	0	0	0	0


 = Unimplemented or Reserved

Figure 23-6. SPI Status Register (SPIxS)

Table 23-8. SPIxS Register Field Descriptions

Field	Description
7 SPRF	<p><b>SPI Read Buffer Full Flag</b> — SPRF is set at the completion of an SPI transfer to indicate that received data may be read from the SPI data register (SPIxDH:SPIxDL). When Receive DMA request is disabled(RXDMAE=0), SPRF is cleared by reading SPRF while it is set, then reading the SPI data register. When Receive DMA request is enabled(RXDMAE=1), SPRF will be automatically cleared when the DMA transfer for Receive DMA request is completed(RX DMA Done is asserted).</p> <p>0 No data available in the receive data buffer. 1 Data available in the receive data buffer.</p>
6 SPMF	<p><b>SPI Match Flag</b> — SPMF is set after SPRF = 1 when the value in the receive data buffer matches the value in SPIxMH:SPIxML. To clear the flag, read SPMF when it is set, then write a 1 to it.</p> <p>0 Value in the receive data buffer does not match the value in SPIxMH:SPIxML registers. 1 Value in the receive data buffer matches the value in SPIxMH:SPIxML registers.</p>
5 SPTEF	<p><b>SPI Transmit Buffer Empty Flag</b> — This bit is set when the transmit data buffer is empty. When Transmit DMA request is disabled(TXDMAE=0), SPTEF is cleared by reading SPIxS with SPTEF set, followed by writing a data value to the transmit buffer at SPIxDH:SPIxDL. SPIxS must be read with SPTEF = 1 before writing data to SPIxDH:SPIxDL or the SPIxDH:SPIxDL write will be ignored. When Transmit DMA request is enabled(TXDMAE=1), SPTEF will be automatically cleared when the DMA transfer for Transmit DMA request is completed(TX DMA Done is asserted). SPTEF is automatically set when all data from the transmit buffer transfers into the transmit shift register. For an idle SPI, data written to SPIxDH:SPIxDL is transferred to the shifter almost immediately so SPTEF is set within two bus cycles allowing a second data to be queued into the transmit buffer. After completion of the transfer of the data in the shift register, the queued data from the transmit buffer will automatically move to the shifter and SPTEF will be set to indicate there is room for new data in the transmit buffer. If no new data is waiting in the transmit buffer, SPTEF simply remains set and no data moves from the buffer to the shifter. If the transfer does not stop, the last data was transmitted will be send out again.</p> <p>0 SPI transmit buffer not empty 1 SPI transmit buffer empty</p>
4 MODF	<p><b>Master Mode Fault Flag</b> — MODF is set if the SPI is configured as a master and the slave select input goes low, indicating some other SPI device is also configured as a master. The <math>\overline{SS}</math> pin acts as a mode fault error input only when MSTR = 1, MODFEN = 1, and SSOE = 0; otherwise, MODF will never be set. MODF is cleared by reading MODF while it is 1, then writing to SPI control register 1 (SPIxC1).</p> <p>0 No mode fault error 1 Mode fault error detected</p>

### 23.3.5 SPI Data Registers (SPIxDH:SPIxDL)

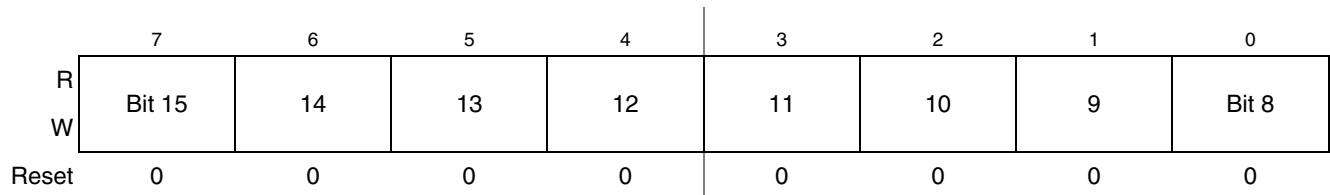


Figure 23-7. SPI Data Register High (SPIxDH)

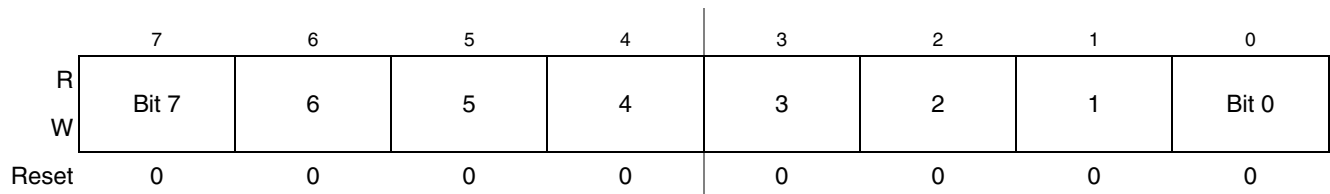


Figure 23-8. SPI Data Register Low (SPIxDL)

The SPI data registers (SPIxDH:SPIxDL) are both the input and output register for SPI data. A write to these registers writes to the transmit data buffer, allowing data to be queued and transmitted.

When the SPI is configured as a master, data queued in the transmit data buffer is transmitted immediately after the previous transmission has completed.

The SPI transmit buffer empty flag (SPTEF) in the SPIxS register indicates when the transmit data buffer is ready to accept new data. When Transmit DMA request is disabled (TXDMAE= 0), SPIxS must be read when SPTEF is set before writing to the SPI data registers, or the write will be ignored. When Transmit DMA request is enabled (TXDMAE=1), the SPI data registers can be written automatically by DMA without reading SPIxS first when SPTEF is set.

Data may be read from SPIxDH:SPIxDL any time after SPRF is set and before another transfer is finished. Failure to read the data out of the receive data buffer before a new transfer ends causes a receive overrun condition and the data from the new transfer is lost. In the case of a receive overrun, the new data is lost because the receive buffer still held the previous character and was not ready to accept the new data. There is no indication for such an overrun condition, so the application system designer must ensure that the previous data has been read from the receive buffer before a new transfer is initiated.

In 8-bit mode, only SPIxDL is available. Reads of SPIxDH will return all 0s. Writes to SPIxDH will be ignored.

In 16-bit mode, reading either byte (SPIxDH or SPIxDL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxDH or SPIxDL) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the transmit data buffer.

### 23.3.6 SPI Match Registers (SPIxMH:SPIxML)

These read/write registers contain the hardware compare value, which sets the SPI match flag (SPMF) when the value received in the SPI receive data buffer equals the value in the SPIxMH:SPIxML registers.

In 8-bit mode, only SPIxML is available. Reads of SPIxMH will return all 0s. Writes to SPIxMH will be ignored.

In 16-bit mode, reading either byte (SPIxMH or SPIxML) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxMH or SPIxML) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent value into the SPI match registers.

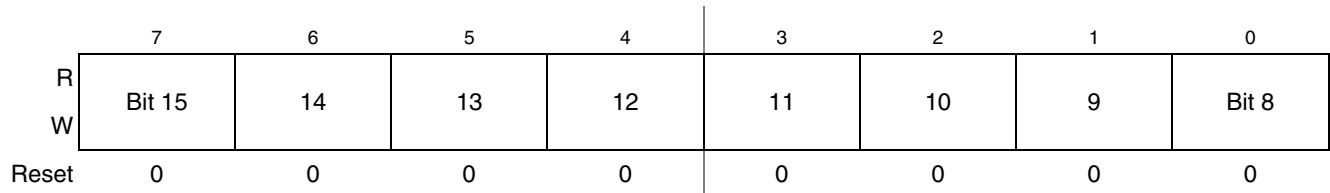


Figure 23-9. SPI Match Register High (SPIxMH)

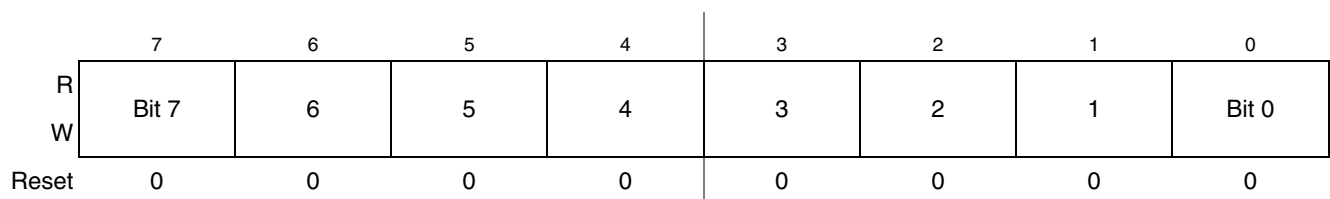


Figure 23-10. SPI Match Register Low (SPIxML)

## 23.4 Functional Description

### 23.4.1 General

The SPI system is enabled by setting the SPI enable (SPE) bit in SPI Control Register 1. While the SPE bit is set, the four associated SPI port pins are dedicated to the SPI function as:

- Slave select ( $\overline{SS}$ )
- Serial clock (SPSCK)
- Master out/slave in (MOSI)
- Master in/slave out (MISO)

An SPI transfer is initiated in the master SPI device by reading the SPI status register (SPIxS) when SPTEF = 1 and then writing data to the transmit data buffer (write to SPIxDH:SPIxDL). When a transfer is complete, received data is moved into the receive data buffer. The SPIxDH:SPIxDL registers act as the SPI receive data buffer for reads and as the SPI transmit data buffer for writes.

The clock phase control bit (CPHA) and a clock polarity control bit (CPOL) in the SPI Control Register 1 (SPIxC1) select one of four possible clock formats to be used by the SPI system. The CPOL bit simply selects a non-inverted or inverted clock. The CPHA bit is used to accommodate two fundamentally different protocols by sampling data on odd numbered SPSCK edges or on even numbered SPSCK edges.

The SPI can be configured to operate as a master or as a slave. When the MSTR bit in SPI control register 1 is set, master mode is selected, when the MSTR bit is clear, slave mode is selected.

## 23.4.2 Master Mode

The SPI operates in master mode when the MSTR bit is set. Only a master SPI module can initiate transmissions. A transmission begins by reading the SPIxS register while SPTEF = 1 and writing to the master SPI data registers. If the shift register is empty, the byte immediately transfers to the shift register. The data begins shifting out on the MOSI pin under the control of the serial clock.

- SPSCCK

The SPR3, SPR2, SPR1, and SPR0 baud rate selection bits in conjunction with the SPPR2, SPPR1, and SPPR0 baud rate preselection bits in the SPI Baud Rate register control the baud rate generator and determine the speed of the transmission. The SPSCCK pin is the SPI clock output. Through the SPSCCK pin, the baud rate generator of the master controls the shift register of the slave peripheral.

- MOSI, MISO pin

In master mode, the function of the serial data output pin (MOSI) and the serial data input pin (MISO) is determined by the SPC0 and BIDIROE control bits.

- $\overline{SS}$  pin

If MODFEN and SSOE bit are set, the  $\overline{SS}$  pin is configured as slave select output. The  $\overline{SS}$  output becomes low during each transmission and is high when the SPI is in idle state.

If MODFEN is set and SSOE is cleared, the  $\overline{SS}$  pin is configured as input for detecting mode fault error. If the  $\overline{SS}$  input becomes low this indicates a mode fault error where another master tries to drive the MOSI and SPSCCK lines. In this case, the SPI immediately switches to slave mode, by clearing the MSTR bit and also disables the slave output buffer MISO (or SISO in bidirectional mode). So the result is that all outputs are disabled and SPSCCK, MOSI and MISO are inputs. If a transmission is in progress when the mode fault occurs, the transmission is aborted and the SPI is forced into idle state.

This mode fault error also sets the mode fault (MODF) flag in the SPI Status Register (SPIxS). If the SPI interrupt enable bit (SPIE) is set when the MODF flag gets set, then an SPI interrupt sequence is also requested.

When a write to the SPI Data Register in the master occurs, there is a half SPSCCK-cycle delay. After the delay, SPSCCK is started within the master. The rest of the transfer operation differs slightly, depending on the clock format specified by the SPI clock phase bit, CPHA, in SPI Control Register 1 (see [Section 23.4.6, “SPI Clock Formats”](#)).

### NOTE

A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0, BIDIROE with SPC0 set, SPIMODE, SPPR2-SPPR0 and SPR3-SPR0 in master mode will abort a transmission in progress and force the SPI into idle state. The remote slave cannot detect this, therefore the master has to ensure that the remote slave is set back to idle state.

## 23.4.3 Slave Mode

The SPI operates in slave mode when the MSTR bit in SPI Control Register1 is clear.

- SPSCCK

In slave mode, SPSCCK is the SPI clock input from the master.

- MISO, MOSI pin

In slave mode, the function of the serial data output pin (MISO) and serial data input pin (MOSI) is determined by the SPC0 bit and BIDIROE bit in SPI Control Register 2.

- $\overline{SS}$  pin

The  $\overline{SS}$  pin is the slave select input. Before a data transmission occurs, the  $\overline{SS}$  pin of the slave SPI must be low.  $\overline{SS}$  must remain low until the transmission is complete. If  $\overline{SS}$  goes high, the SPI is forced into idle state.

The  $\overline{SS}$  input also controls the serial data output pin, if  $\overline{SS}$  is high (not selected), the serial data output pin is high impedance, and, if  $\overline{SS}$  is low the first bit in the SPI Data Register is driven out of the serial data output pin. Also, if the slave is not selected ( $\overline{SS}$  is high), then the SPSCCK input is ignored and no internal shifting of the SPI shift register takes place.

Although the SPI is capable of duplex operation, some SPI peripherals are capable of only receiving SPI data in a slave mode. For these simpler devices, there is no serial data out pin.

#### NOTE

When peripherals with duplex capability are used, take care not to simultaneously enable two receivers whose serial outputs drive the same system slave's serial data output line.

As long as no more than one slave device drives the system slave's serial data output line, it is possible for several slaves to receive the same transmission from a master, although the master would not receive return information from all of the receiving slaves.

If the CPHA bit in SPI Control Register 1 is clear, odd numbered edges on the SPSCCK input cause the data at the serial data input pin to be latched. Even numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

If the CPHA bit is set, even numbered edges on the SPSCCK input cause the data at the serial data input pin to be latched. Odd numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

When CPHA is set, the first edge is used to get the first data bit onto the serial data output pin. When CPHA is clear and the  $\overline{SS}$  input is low (slave selected), the first bit of the SPI data is driven out of the serial data output pin. After the eighth (SPIMODE = 0) or sixteenth (SPIMODE = 1) shift, the transfer is considered complete and the received data is transferred into the SPI data registers. To indicate transfer is complete, the SPRF flag in the SPI Status Register is set.

#### NOTE

A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0 and BIDIROE with SPC0 set and SPIMODE in slave mode will corrupt a transmission in progress and has to be avoided.

## 23.4.4 SPI Transmission by DMA

SPI supports both Transmit and Receive by DMA. The basic flow of SPI transmission by DMA is as below.

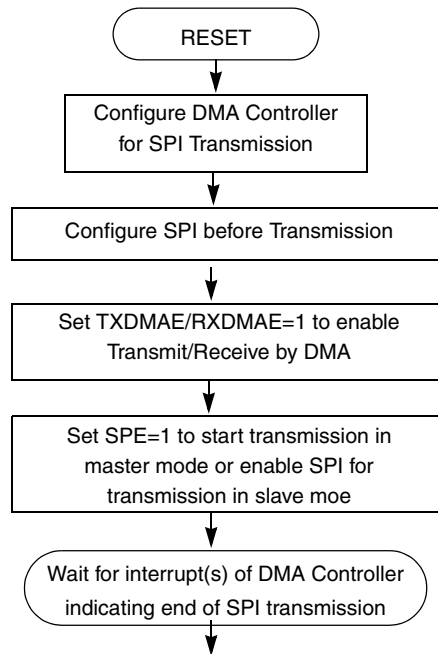


Figure 23-11. Basic Flow of SPI Transmission by DMA

### 23.4.4.1 Transmit by DMA

Transmit by DMA is support only when TXDMAE is set. At first, Transmit DMA request will be asserted when both SPE and SPTEF are set. Then on-chip DMA controller will detect this request and transfer data from memory into SPI data register. After that, TX DMA DONE will be asserted to clear SPTEF automatically. The process above will be repeated until all data for Transmit(number decided by configuration register(s) of DMA controller) has been sent out.

After DMA transfer the first byte to the SPI data register, the SPI would push this data into shifter thereby making the SPTEF high again. This would generate another DMA request immediately. The subsequent DMA request would be placed at the SPI transfer rate. Recommendation action is to configure DMA in cycle steal mode for transmitting multiple bytes from SPI using DMA.



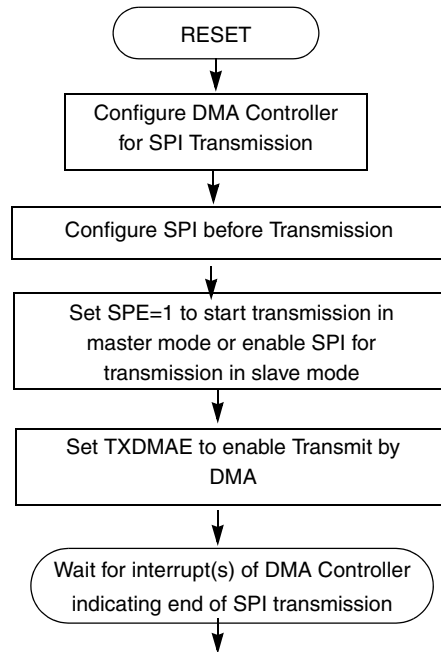


Figure 23-12. Recommendation startup of SPI Transmit by DMA

#### 23.4.4.2 Receive by DMA

Receive by DMA is support only when RXDMAE is set. At first, Receive DMA request will be asserted when both SPE and SPRF are set. Then on-chip DMA controller will detect this request and transfer data from SPI data register into memory. After that, RX DMA DONE will be asserted to clear SPRF automatically. The process above will be repeated until all data to be received (number decided by configuration register(s) of DMA controller) has been received or no Receive DMA request will be generated again since the SPI transmission is finished.

#### 23.4.5 Data Transmission Length

The SPI can support data lengths of 8 or 16 bits. The length can be configured with the SPIMODE bit in the SPIx C2 register.

In 8-bit mode (SPIMODE = 0), the SPI Data Register is comprised of one byte: SPIx DL. The SPI Match Register is also comprised of only one byte: SPIx ML. Reads of SPIx DH and SPIx MH will return zero. Writes to SPIx DH and SPIx MH will be ignored.

In 16-bit mode (SPIMODE = 1), the SPI Data Register is comprised of two bytes: SPIx DH and SPIx DL. Reading either byte (SPIx DH or SPIx DL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIx DH or SPIx DL) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the transmit data buffer.

In 16-bit mode, the SPI Match Register is also comprised of two bytes: SPIx MH and SPIx ML. There is no buffer mechanism for the reading of SPIx MH and SPIx ML since they can only be changed by writing at

CPU side. Writing to either byte (SPIxMH or SPIxML) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the SPI Match Register.

Any switching between 8- and 16-bit data transmission length (controlled by SPIMODE bit) in master mode will abort a transmission in progress, force the SPI system into idle state, and reset all status bits in the SPIxS register. To initiate a transfer after writing to SPIMODE, the SPIxS register must be read with SPTEF = 1, and data must be written to SPIxDH:SPIxDL in 16-bit mode (SPIMODE = 1) or SPIxDL in 8-bit mode (SPIMODE = 0).

In slave mode, user software should write to SPIMODE only once to prevent corrupting a transmission in progress.

#### NOTE

Data can be lost if the data length is not the same for both master and slave devices.

### 23.4.6 SPI Clock Formats

To accommodate a wide variety of synchronous serial peripherals from different manufacturers, the SPI system has a clock polarity (CPOL) bit and a clock phase (CPHA) control bit to select one of four clock formats for data transfers. CPOL selectively inserts an inverter in series with the clock. CPHA chooses between two different clock phase relationships between the clock and data.

Figure 23-13 shows the clock formats when SPIMODE = 0 (8-bit mode) and CPHA = 1. At the top of the figure, the eight bit times are shown for reference with bit 1 starting at the first SPSCCK edge and bit 8 ending one-half SPSCCK cycle after the eighth SPSCCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in CPOL. The SAMPLE IN waveform applies to the MOSI input of a slave or the MISO input of a master. The MOSI waveform applies to the MOSI output pin from a master and the MISO waveform applies to the MISO output from a slave. The  $\overline{SS}$  OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master  $\overline{SS}$  output goes to active low one-half SPSCCK cycle before the start of the transfer and goes back high at the end of the eighth bit time of the transfer. The  $\overline{SS}$  IN waveform applies to the slave select input of a slave.

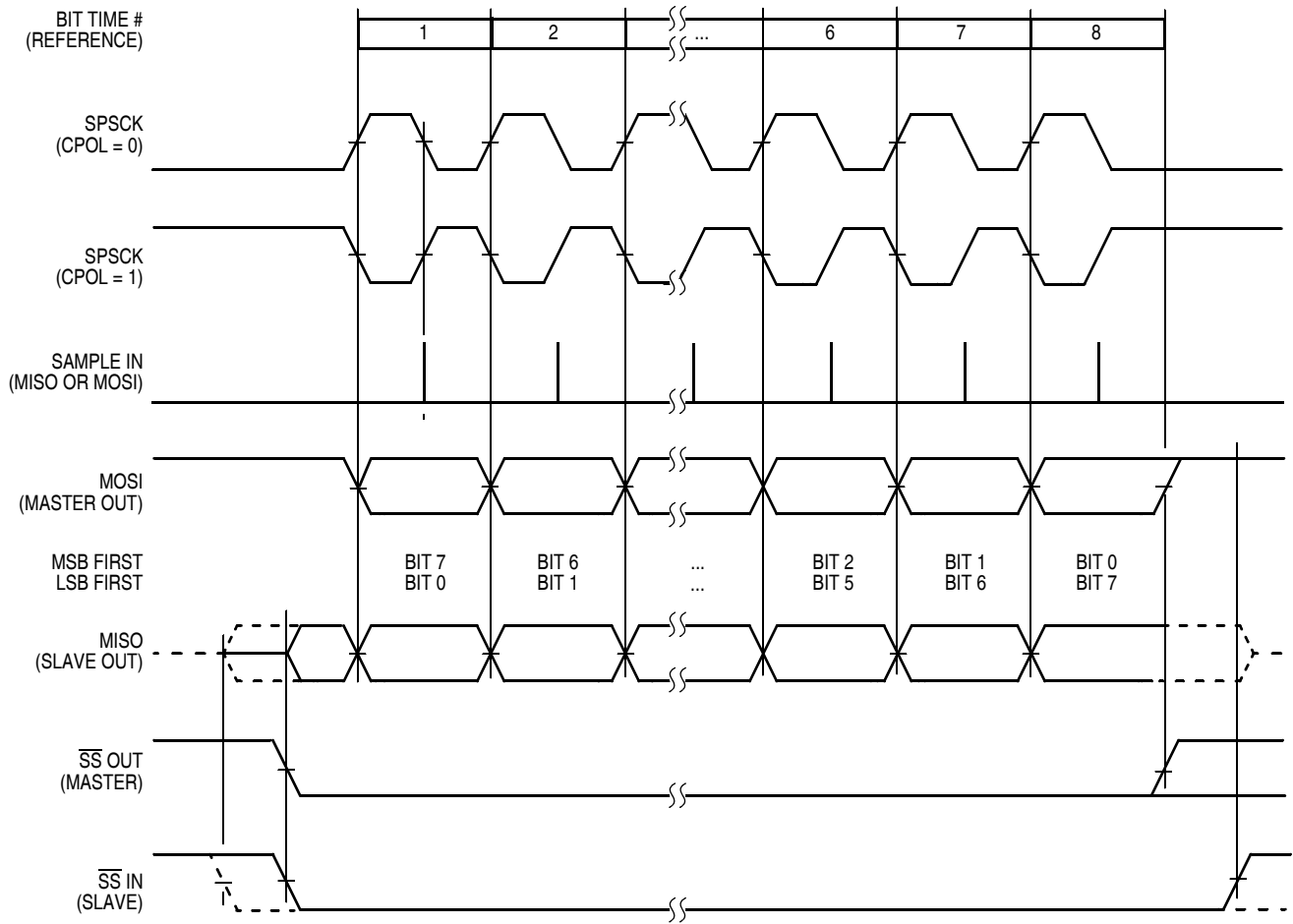


Figure 23-13. SPI Clock Formats (CPHA = 1)

When CPHA = 1, the slave begins to drive its MISO output when  $\overline{SS}$  goes to active low, but the data is not defined until the first SPSCCK edge. The first SPSCCK edge shifts the first bit of data from the shifter onto the MOSI output of the master and the MISO output of the slave. The next SPSCCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the third SPSCCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled, and shifts the second data bit value out the other end of the shifter to the MOSI and MISO outputs of the master and slave, respectively. When CPHA = 1, the slave's  $\overline{SS}$  input is not required to go to its inactive high level between transfers.

Figure 23-14 shows the clock formats when SPIMODE = 0 and CPHA = 0. At the top of the figure, the eight bit times are shown for reference with bit 1 starting as the slave is selected ( $\overline{SS}$  IN goes low), and bit 8 ends at the last SPSCCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in CPOL. The SAMPLE IN waveform applies to the MOSI input of a slave or the MISO input of a master. The MOSI waveform applies to the MOSI output pin from a master and the MISO waveform applies to the MISO output from a slave. The  $\overline{SS}$  OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master  $\overline{SS}$  output goes to active low at the start of the first bit time of the transfer and goes back high one-half

SPSCK cycle after the end of the eighth bit time of the transfer. The  $\overline{SS}$  IN waveform applies to the slave select input of a slave.

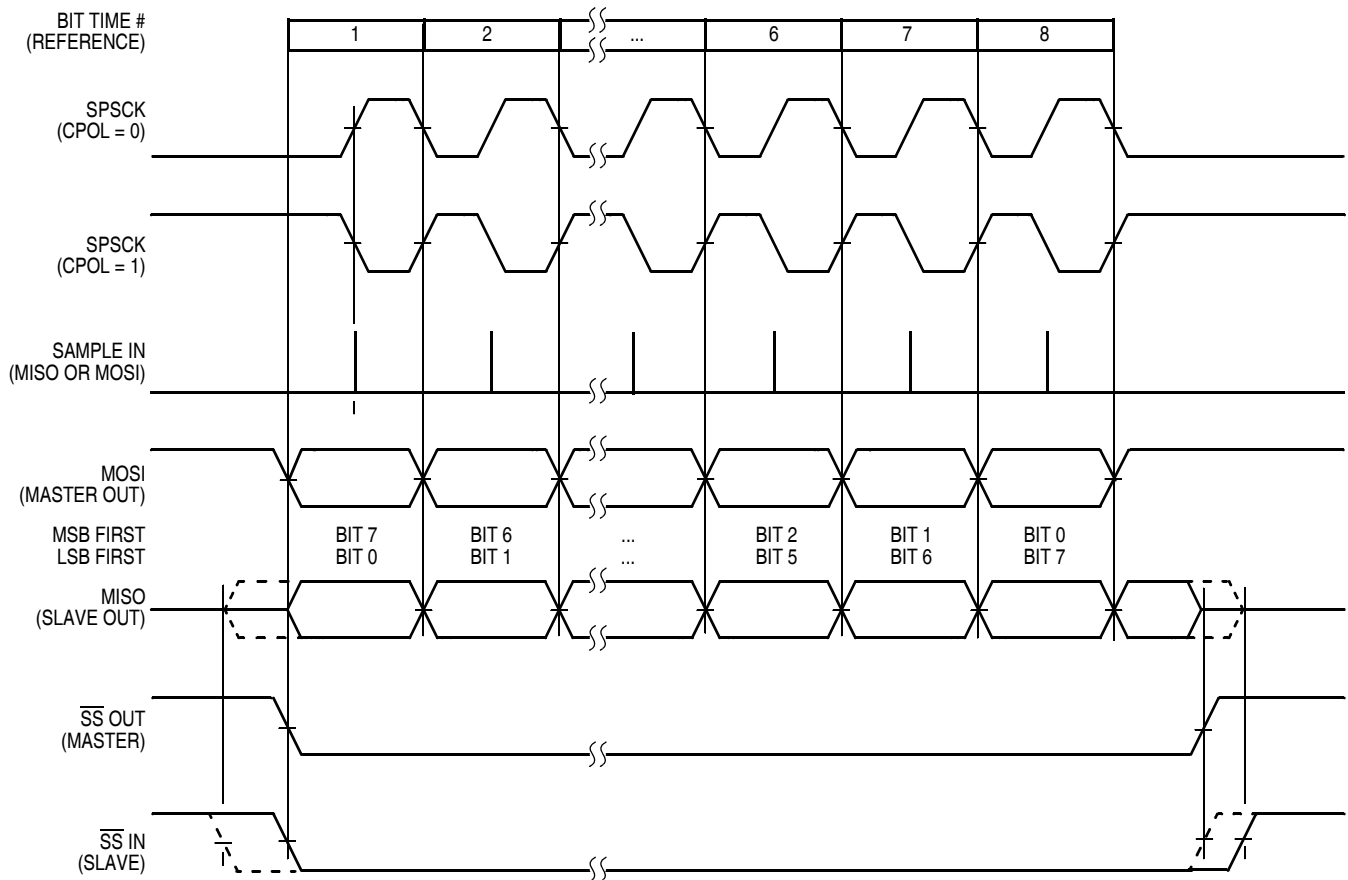


Figure 23-14. SPI Clock Formats (CPHA = 0)

When CPHA = 0, the slave begins to drive its MISO output with the first data bit value (MSB or LSB depending on LSBFE) when  $\overline{SS}$  goes to active low. The first SPSCCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the second SPSCCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled and shifts the second data bit value out the other end of the shifter to the MOSI and MISO outputs of the master and slave, respectively. When CPHA = 0, the slave's  $\overline{SS}$  input must go to its inactive high level between transfers.

### 23.4.7 SPI Baud Rate Generation

As shown in Figure 23-15, the clock source for the SPI baud rate generator is the bus clock. The three prescale bits (SPPR2:SPPR1:SPPR0) choose a prescale divisor of 1, 2, 3, 4, 5, 6, 7, or 8. The three rate select bits (SPR3:SPR2:SPR1:SPR0) divide the output of the prescaler stage by 2, 4, 8, 16, 32, 64, 128, 256 or 512 to get the internal SPI master mode bit-rate clock.

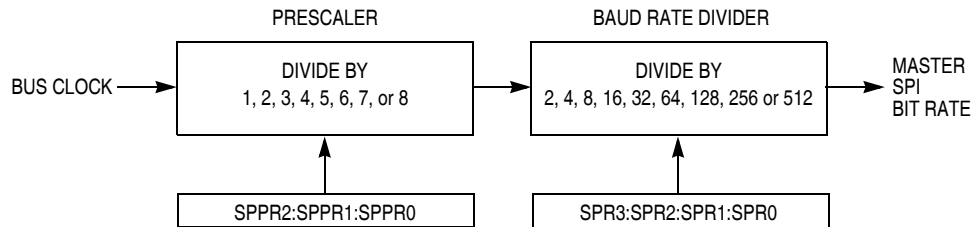
The baud rate generator is activated only when the SPI is in the master mode and a serial transfer is taking place. In the other cases, the divider is disabled to decrease  $I_{DD}$  current.

The baud rate divisor equation is as follows except those reserved combinations in [Table 23-7](#):

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) \cdot 2^{(\text{SPR} + 1)}$$

The baud rate can be calculated with the following equation:

$$\text{Baud Rate} = \text{BusClock} / \text{BaudRateDivisor}$$



**Figure 23-15. SPI Baud Rate Generation**

## 23.4.8 Special Features

### 23.4.8.1 $\overline{SS}$ Output

The  $\overline{SS}$  output feature automatically drives the  $\overline{SS}$  pin low during transmission to select external devices and drives it high during idle to deselect external devices. When  $\overline{SS}$  output is selected, the  $\overline{SS}$  output pin is connected to the  $\overline{SS}$  input pin of the external device.

The  $\overline{SS}$  output is available only in master mode during normal SPI operation by asserting the SSOE and MODFEN bits as shown in [Table 23-2](#).

The mode fault feature is disabled while  $\overline{SS}$  output is enabled.

#### NOTE

Care must be taken when using the  $\overline{SS}$  output feature in a multi-master system since the mode fault feature is not available for detecting system errors between masters.

### 23.4.8.2 Bidirectional Mode (MOMI or SISO)

The bidirectional mode is selected when the SPC0 bit is set in SPI Control Register 2 (see [Table 23-9](#)). In this mode, the SPI uses only one serial data pin for the interface with external device(s). The MSTR bit decides which pin to use. The MOSI pin becomes the serial data I/O (MOMI) pin for the master mode, and the MISO pin becomes serial data I/O (SISO) pin for the slave mode. The MISO pin in master mode and MOSI pin in slave mode are not used by the SPI.

Table 23-9. Normal Mode and Bidirectional Mode

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
<b>Normal Mode</b> SPC0 = 0		
<b>Bidirectional Mode</b> SPC0 = 1		

The direction of each serial I/O pin depends on the BIDIROE bit. If the pin is configured as an output, serial data from the shift register is driven out on the pin. The same pin is also the serial input to the shift register.

The SPSCK is output for the master mode and input for the slave mode.

The  $\overline{SS}$  is the input or output for the master mode, and it is always the input for the slave mode.

The bidirectional mode does not affect SPSCK and  $\overline{SS}$  functions.

#### NOTE

In bidirectional master mode, with mode fault enabled, both data pins MISO and MOSI can be occupied by the SPI, though MOSI is normally used for transmissions in bidirectional mode and MISO is not used by the SPI. If a mode fault occurs, the SPI is automatically switched to slave mode, in this case MISO becomes occupied by the SPI and MOSI is not used. This has to be considered, if the MISO pin is used for another purpose.

### 23.4.9 Error Conditions

The SPI has one error condition:

- Mode fault error

#### 23.4.9.1 Mode Fault Error

If the  $\overline{SS}$  input becomes low while the SPI is configured as a master, it indicates a system error where more than one master may be trying to drive the MOSI and SPSCK lines simultaneously. This condition is not permitted in normal operation, and the MODF bit in the SPI status register is set automatically provided the MODFEN bit is set.

In the special case where the SPI is in master mode and MODFEN bit is cleared, the  $\overline{SS}$  pin is not used by the SPI. In this special case, the mode fault error function is inhibited and MODF remains cleared. In case the SPI system is configured as a slave, the  $\overline{SS}$  pin is a dedicated input pin. Mode fault error doesn't occur in slave mode.

If a mode fault error occurs the SPI is switched to slave mode, with the exception that the slave output buffer is disabled. So SPSCCK, MISO and MOSI pins are forced to be high impedance inputs to avoid any possibility of conflict with another output driver. A transmission in progress is aborted and the SPI is forced into idle state.

If the mode fault error occurs in the bidirectional mode for a SPI system configured in master mode, output enable of the MOMI (MOSI in bidirectional mode) is cleared if it was set. No mode fault error occurs in the bidirectional mode for the SPI system configured in slave mode.

The mode fault flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to SPI Control Register 1. If the mode fault flag is cleared, the SPI becomes a normal master or slave again.

## 23.4.10 Low Power Mode Options

### 23.4.10.1 SPI in Run Mode

In run mode with the SPI system enable (SPE) bit in the SPI control register clear, the SPI system is in a low-power, disabled state. SPI registers can still be accessed, but clocks to the core of this module are disabled.

### 23.4.10.2 SPI in Wait Mode

SPI operation in wait mode depends upon the state of the SPISWAI bit in SPI Control Register 2.

- If SPISWAI is clear, the SPI operates normally when the CPU is in wait mode
- If SPISWAI is set, SPI clock generation ceases and the SPI module enters a power conservation state when the CPU is in wait mode.
  - If SPISWAI is set and the SPI is configured for master, any transmission and reception in progress stops at wait mode entry. The transmission and reception resumes when the SPI exits wait mode.
  - If SPISWAI is set and the SPI is configured as a slave, any transmission and reception in progress continues if the SPSCCK continues to be driven from the master. This keeps the slave synchronized to the master and the SPSCCK.

If the master transmits data while the slave is in wait mode, the slave will continue to send out data consistent with the operation mode at the start of wait mode (i.e., if the slave is currently sending its SPIxDH:SPIxDL to the master, it will continue to send the same byte. Otherwise, if the slave is currently sending the last data received byte from the master, it will continue to send each previously receive data from the master byte).

**NOTE**

Care must be taken when expecting data from a master while the slave is in wait or stop3 mode. Even though the shift register will continue to operate, the rest of the SPI is shut down (i.e. a SPRF interrupt will not be generated until exiting stop or wait mode). Also, the data from the shift register will not be copied into the SPIxDH:SPIxDL registers until after the slave SPI has exited wait or stop mode. A SPRF flag and SPIxDH:SPIxDL copy is only generated if wait mode is entered or exited during a transmission. If the slave enters wait mode in idle mode and exits wait mode in idle mode, neither a SPRF nor a SPIxDH:SPIxDL copy will occur.

**23.4.10.3 SPI in Stop Mode**

Stop3 mode is dependent on the SPI system. Upon entry to stop3 mode, the SPI module clock is disabled (held high or low). If the SPI is in master mode and exchanging data when the CPU enters stop mode, the transmission is frozen until the CPU exits stop mode. After stop, data to and from the external SPI is exchanged correctly. In slave mode, the SPI will stay synchronized with the master.

The stop mode is not dependent on the SPISWAI bit.

In all other stop modes, the SPI module is completely disabled. After stop, all registers are reset to their default values, and the SPI module must be re-initialized.

**23.4.10.4 Reset**

The reset values of registers and signals are described in [Section 23.3, “Register Definition.”](#) which details the registers and their bit-fields.

- If a data transmission occurs in slave mode after reset without a write to SPIxDH:SPIxDL, it will transmit garbage, or the data last received from the master before the reset.
- Reading from the SPIxDH:SPIxDL after reset will always read zeros.

**23.4.10.5 Interrupts**

The SPI only originates interrupt requests when the SPI is enabled (SPE bit in SPIxC1 set). The following is a description of how the SPI makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt priority are chip dependent.

**23.4.11 SPI Interrupts**

There are four flag bits, three interrupt mask bits, and one interrupt vector associated with the SPI system. The SPI interrupt enable mask (SPIE) enables interrupts from the SPI receiver full flag (SPRF) and mode fault flag (MODF). The SPI transmit interrupt enable mask (SPTIE) enables interrupts from the SPI transmit buffer empty flag (SPTEF). The SPI match interrupt enable mask bit (SPIMIE) enables interrupts from the SPI match flag (SPMF). When one of the flag bits is set, and the associated interrupt mask bit is set, a hardware interrupt request is sent to the CPU. If the interrupt mask bits are cleared, software can poll the associated flag bits instead of using interrupts. The SPI interrupt service routine (ISR) should check



the flag bits to determine what event caused the interrupt. The service routine should also clear the flag bit(s) before returning from the ISR (usually near the beginning of the ISR).

#### 23.4.11.1 MODF

MODF occurs when the master detects an error on the  $\overline{SS}$  pin. The master SPI must be configured for the MODF feature (see [Table 23-2](#)). Once MODF is set, the current transfer is aborted and the following bit is changed:

- MSTR=0, The master bit in SPIxCI1 resets.

The MODF interrupt is reflected in the status register MODF flag. Clearing the flag will also clear the interrupt. This interrupt will stay active while the MODF flag is set. MODF has an automatic clearing process which is described in [Section 23.3.4, “SPI Status Register \(SPIxS\).”](#)

#### 23.4.11.2 SPRF

SPRF occurs when new data has been received and copied to the SPI receive data buffer. In 8-bit mode, SPRF is set only after all 8 bits have been shifted out of the shift register and into SPIxDL. In 16-bit mode, SPRF is set only after all 16 bits have been shifted out of the shift register and into SPIxDH:SPIxDL.

Once SPRF is set, it does not clear until it is serviced. SPRF has an automatic clearing process which is described in [Section 23.3.4, “SPI Status Register \(SPIxS\).”](#) In the event that the SPRF is not serviced before the end of the next transfer (i.e. SPRF remains active throughout another transfer), the latter transfers will be ignored and no new data will be copied into the SPIxDH:SPIxDL.

#### 23.4.11.3 SPTEF

SPTEF occurs when the SPI transmit buffer is ready to accept new data. In 8-bit mode, SPTEF is set only after all 8 bits have been moved from SPIxDL into the shifter. In 16-bit mode, SPTEF is set only after all 16 bits have been moved from SPIxDH:SPIxDL into the shifter.

Once SPTEF is set, it does not clear until it is serviced. SPTEF has an automatic clearing process which is described in [Section 23.3.4, “SPI Status Register \(SPIxS\).”](#)

#### 23.4.11.4 SPMF

SPMF occurs when the data in the receive data buffer is equal to the data in the SPI match register. In 8-bit mode, SPMF is set only after bits 8–0 in the receive data buffer are determined to be equivalent to the value in SPIxML. In 16-bit mode, SPMF is set after bits 15–0 in the receive data buffer are determined to be equivalent to the value in SPIxMH:SPIxML.

## 23.5 Initialization/Application Information

### 23.5.1 SPI Module Initialization Example

#### 23.5.1.1 Initialization Sequence

Before the SPI module can be used for communication, an initialization procedure must be carried out, as follows:

1. Update control register 1 (SPIxC1) to enable the SPI and to control interrupt enables. This register also sets the SPI as master or slave, determines clock phase and polarity, and configures the main SPI options.
2. Update control register 2 (SPIxC2) to enable additional SPI functions such as the SPI match interrupt feature, the master mode-fault function, and bidirectional mode output. 8- or 16-bit mode select and other optional features are controlled here as well.
3. Update the baud rate register (SPIxBR) to set the prescaler and bit rate divisor for an SPI master.
4. Update the hardware match register (SPIxMH:SPIxML) with the value to be compared to the receive data register for triggering an interrupt if hardware match interrupts are enabled.
5. In the master, read SPIxS while SPTEF = 1, and then write to the transmit data register (SPIxDH:SPIxDL) to begin transfer.

#### 23.5.1.2 Pseudo—Code Example

In this example, the SPI module will be set up for master mode with only hardware match interrupts enabled. The SPI will run in 16-bit mode at a maximum baud rate of bus clock divided by 2. Clock phase and polarity will be set for an active-high SPI clock where the first edge on SPSCCK occurs at the start of the first cycle of a data transfer.

##### **SPIxC1=0x54(%01010100)**

Bit 7	SPIE	= 0	Disables receive and mode fault interrupts
Bit 6	SPE	= 1	Enables the SPI system
Bit 5	SPTIE	= 0	Disables SPI transmit interrupts
Bit 4	MSTR	= 1	Sets the SPI module as a master SPI device
Bit 3	CPOL	= 0	Configures SPI clock as active-high
Bit 2	CPHA	= 1	First edge on SPSCCK at start of first data transfer cycle
Bit 1	SSOE	= 0	Determines $\overline{SS}$ pin function when mode fault enabled
Bit 0	LSBFE	= 0	SPI serial data transfers start with most significant bit

**SPIxC2 = 0xC0(%11000000)**

Bit 7	SPMIE	= 1	SPI hardware match interrupt enabled
Bit 6	SPIMODE	= 1	Configures SPI for 16-bit mode
Bit 5		= 0	Unimplemented
Bit 4	MODFEN	= 0	Disables mode fault function
Bit 3	BIDIROE	= 0	SPI data I/O pin acts as input
Bit 2		= 0	Unimplemented
Bit 1	SPISWAI	= 0	SPI clocks operate in wait mode
Bit 0	SPC0	= 0	uses separate pins for data input and output

**SPIxBR = 0x00(%00000000)**

Bit 7		= 0	Unimplemented
Bit 6:4		= 000	Sets prescale divisor to 1
Bit 3:0		= 0000	Sets baud rate divisor to 2

**SPIxS = 0x00(%00000000)**

Bit 7	SPRF	= 0	Flag is set when receive data buffer is full
Bit 6	SPMF	= 0	Flag is set when SPIMH/L = receive data buffer
Bit 5	SPTEF	= 0	Flag is set when transmit data buffer is empty
Bit 4	MODF	= 0	Mode fault flag for master mode
Bit 3:0		= 0	Unimplemented

**SPIxMH = 0xXX**

In 16-bit mode, this register holds bits 8–15 of the hardware match buffer. In 8-bit mode, writes to this register will be ignored.

**SPIxML = 0xXX**

Holds bits 0–7 of the hardware match buffer.

**SPIxDH = 0xxx**

In 16-bit mode, this register holds bits 8–15 of the data to be transmitted by the transmit buffer and received by the receive buffer.

**SPIxDL = 0xxx**

Holds bits 0–7 of the data to be transmitted by the transmit buffer and received by the receive buffer.

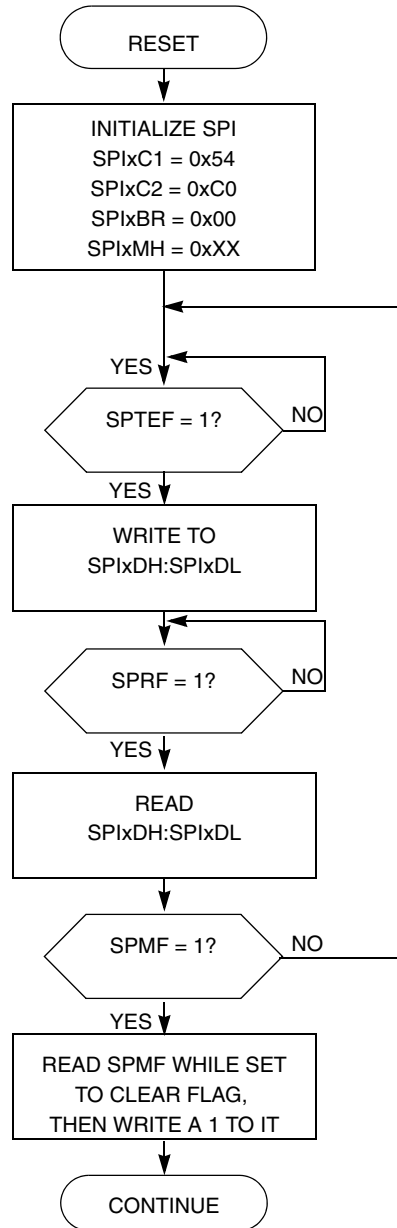


Figure 23-16. Initialization Flowchart Example for SPI Master Device in 16-bit Mode

# Chapter 24

## Programmable Delay Block (S08PDBV1)

### 24.1 Introduction

Motor control applications often need to synchronize the time at which ADC samples or Comparator sampling windows are taken with respect to the PWM period. Normally the PWM timer module has a SYNC output specifically for that purpose. The primary function of the programmable delay block is simply to provide a controllable delay from the PWM SYNC output to the sample trigger input of the programmable gain amplifiers and ADCs and a controllable window which is synchronized with PWM pulses for analog comparators to compare the analog signals in a defined window.

#### NOTE

The MCF51AG128 series of MCUs include one PDB module.

#### 24.1.1 Low Power Mode Operation

The PDB module is capable of functioning in run and wait modes of operation, and does not differentiate between run and wait modes. For details on low-power mode operation, refer to [Table 3-1](#) in [Chapter 3](#), “Modes of Operation.”

#### 24.1.2 PDB Clock Gating

The bus clock to the PDB module can be gated on and off using the SCGC2[PDB] bit. This bit is cleared after any reset that disables the bus clock to this module. The SCGC2[PDB] bit must be set before operation. See [Section 5.7](#), “Peripheral Clock Gating,” for details.

#### 24.1.3 PDB Input Trigger Assignments

The PDB module supports up to 7 external trigger sources selectable via the TRIGSEL bits in PDBCTRL2. [Table 24-1](#) summarizes the PDB input trigger sources.

**Table 24-1. PDB Input Trigger Sources**

TRIGSEL	Trigger Source	
	Module	Signal
000	RTC	Counter Overflow
001	HSCMP1	COUT
010	HSCMP2	COUT

Table 24-1. PDB Input Trigger Sources (continued)

TRIGSEL	Trigger Source	
	Module	Signal
011	FTM1	FTM1EXTTRIG
100	FTM2	FTM2EXTTRIG
101	iEvent_3	iEvent_ch3
110	—	—
111	Internal Software Trigger	

## 24.1.4 PDB Output Usage

### 24.1.4.1 PDB Interfacing to ADC Hardware Trigger

The PDB PreTriggerA and PreTriggerB are used as SSEL[0:1] signals of ADC module to support dual sampling.

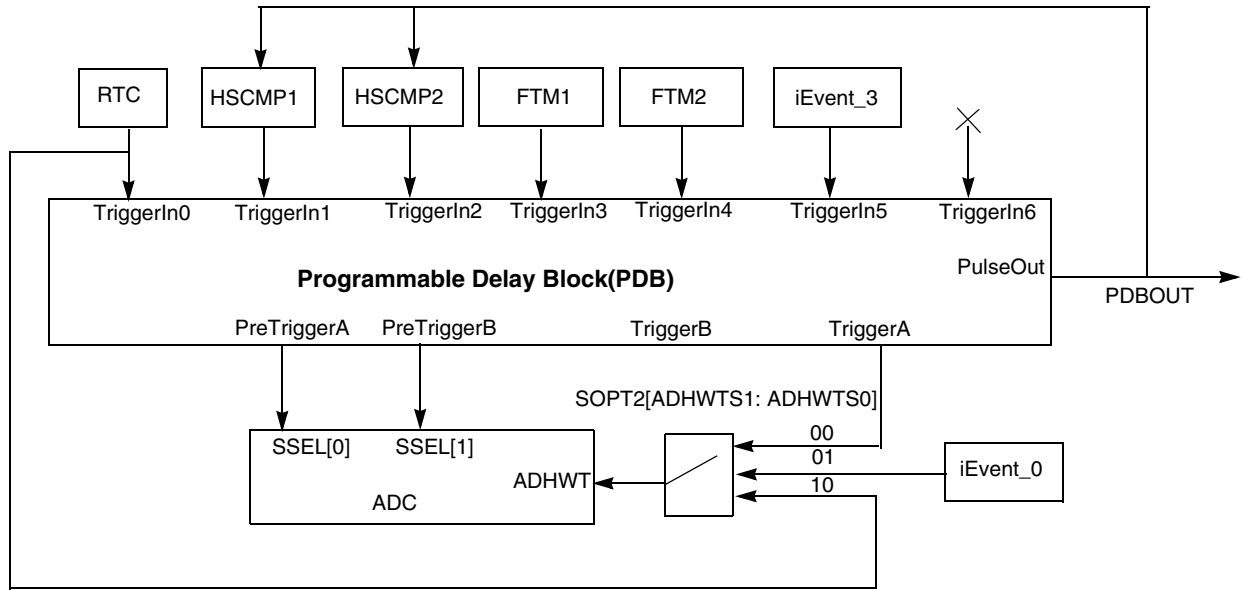
The ADC hardware trigger, ADHWT, is enabled by the ADCSC2[ADTRG] bit. The hardware trigger sources are TriggerA output, iEvent\_0 or RTC overflow output that could be selected with the SOPT2[ADHWTS1:ADHWTS0] bits.

### 24.1.4.2 PDB Interfacing to HSCMP Windowing Function

On MCF51AG128 series MCUs, each of the HSCMP modules can be configured for windowing or sampling modes of operation. The PDB PulseOut output is used as the windows/sample input to all two HSCMP modules.

[Figure 24-1](#) shows the configuration used for PDB.

**Programmable Delay Block (S08PDBV1)**



**Figure 24-1. PDB Interface to HSCMP, RTC IRQ, and ADC**

## 24.1.5 Features

- 16-bit of resolution with prescaler
- Positive transition of trigger event signal initiates the counter
- Supports two triggered delay outputs. Each has an independently controlled delay from trigger event
- PDB outputs can be ORed together to schedule two conversions from one input trigger event
- PDB outputs can schedule precise edge placement for a pulsed output. This feature is used to generate the control signal for the HSCMP windowing feature and output to a package pin if needed for applications, such as critical conductive mode power factor correction
- Continuous trigger or single shot mode supported
- Bypass mode supported
- Each PDB output is independently enabled
- Seven possible trigger events

## 24.1.6 Modes of Operation

Modes of operation include:

- Disabled — Counter is off and TriggerA and TriggerB are low.
- Enabled one-shot — Counter is enabled and restarted at count one upon receiving a positive edge on the trigger input. TriggerA and TriggerB see only one output trigger per input trigger.
- Enabled continuous — Counter is enabled & restarted at count one. The counter is rolled over to one again when the count reaches the value specified in the MOD register, and counting restarted. This enables a continuous stream of triggers out as a result of a single trigger input.
- Bypassed — The input trigger bypasses the PDB logic entirely. It is possible to bypass either of the two trigger outputs or both.
- In enabled one-shot and enabled continuous, the outputs of the delay A and delay B comparators can be combined so that two ADC events can be triggered from a single input event. These are referred to as two-shot and continuous two-shot modes.
- In enabled one-shot and enabled continuous, the outputs of the delay A and delay B comparators can be combined so that output pulses can be generated with precisely controlled rising and falling times. These are referred to as single pulse and continuous pulse modes.

## 24.1.7 Block Diagram

Figure 24-7. illustrates the basic structure of the PDB block. It contains a single counter whose output is compared against three different digital values. delayA and delayB determine the time between assertion of the trigger input to the point at which changes in the trigger output signals are initiated. These times are defined as:

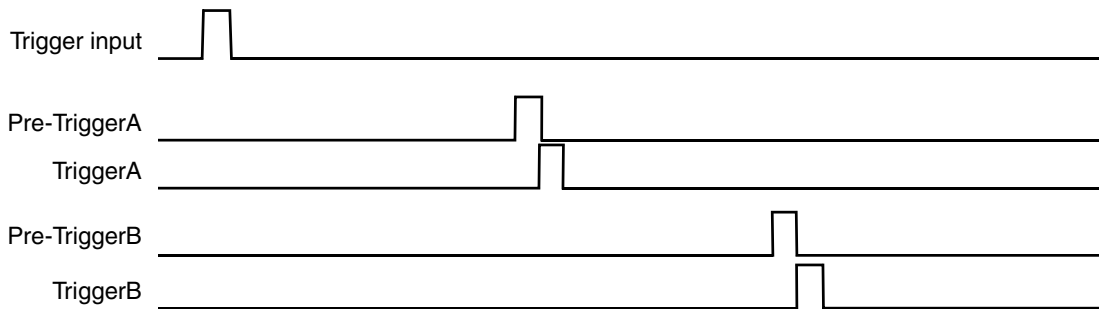
- Trigger input to Pre-TriggerA =  $(\text{prescaler} \times \text{delayA}) + 1$  peripheral bus clock cycle
- Trigger input to Pre-TriggerB =  $(\text{prescaler} \times \text{delayB}) + 1$  peripheral bus clock cycle



- Add one additional peripheral bus clock cycle to determine the time at which the trigger outputs change

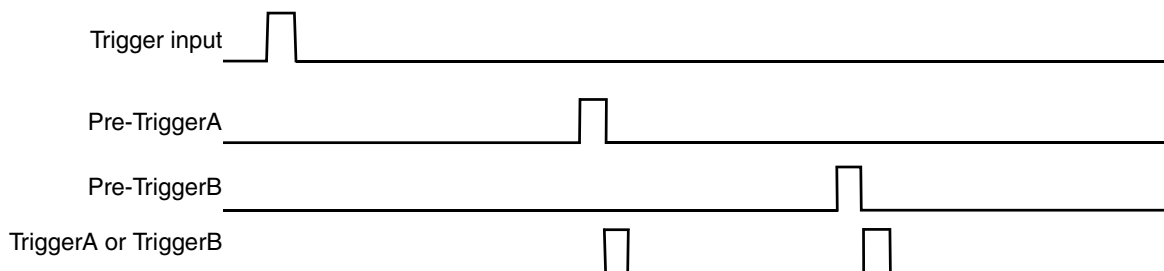
If the device includes the PGAs, Pre-TriggerA and Pre-TriggerB are used to pre-trigger the PGA in the PGA/ADC blocks with one peripheral bus clock period prior to the actual ADC measurement trigger.

If the ADC block contains duplicate control and result registers, TriggerA and TriggerB allow them to operate in a ping-pong fashion, alternating conversions between two different analog sources (per converter). The Pre-Trigger signals are used to specify which signal is sampled next. The signals shown in [Figure 24-2](#). would be used to operate the ADC to sample signal A and sample single B in one-shot mode. The trigger delays for the ADC are independently set via the DELAYA and DELAYB parameters. The one trigger signal from PWM timer module can start the ADC twice with two different delays. But the time between TriggerA and TriggerB must be longer than the ADC conversion time, then the second trigger signal can take effect.



**Figure 24-2. Decoupled A & B Trigger Generation**

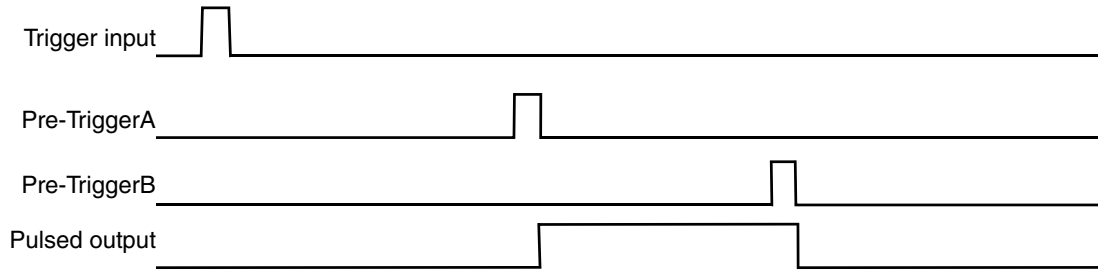
If device integrates two ADCs, two-shot mode is shown in [Figure 24-3](#). In this case, ADC A and ADC B are given the same trigger; resulting in a total of four ADC conversions (two on ADC A, and two on ADC B).



**Figure 24-3. Trigger Configured for Simultaneous Sampling / Ping-Pong Operation**

The third digital value, modulus, is used to reset the counter back to one at the end of the count. If PDBxCTRL2[CONT] is set, the counter then resumes a new count. Otherwise, the timer operation ceases until the next trigger input event occurs.

The pulsed modes are shown in [Figure 24-4](#). In this case, Pre-TriggerA and Pre-Trigger B are used to precisely schedule the rising and falling edges for the output waveform.

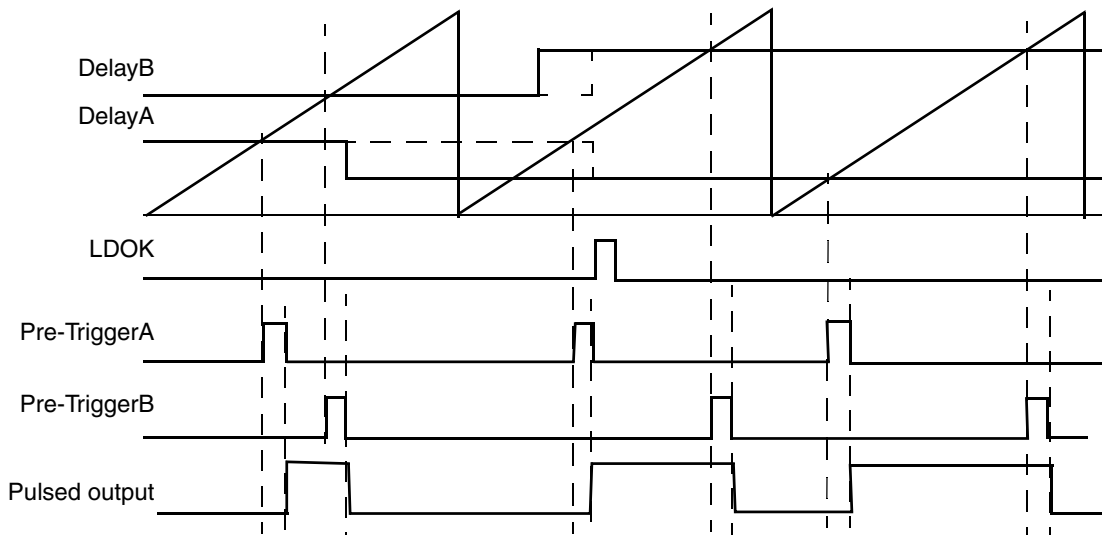


**Figure 24-4. Trigger Pulsed Output Operation**

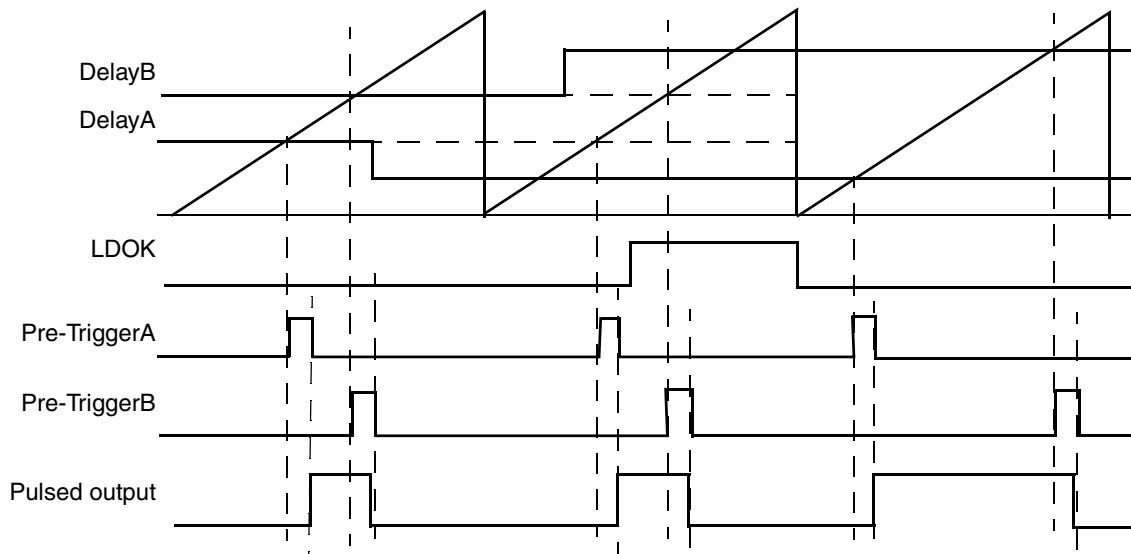
DelayA and DelayB and modulus registers are buffered. The values written into these registers does not take effect until:

- A logic 1 is written to PDBCTRL1[LDOK] if the PDBCTRL1[LDMOD] is cleared.
- Either counter rolls over in continuous mode or trigger signal is received in one-shot mode after a logic 1 is written to PDBxCTRL1[LDOK] if PDBxCTRL1[LDMOD] is set. In this case, if any value is written to any one of these registers, after a logic 1 being written to LDOK, it is ignored until the values in these registers are loaded into the buffers.

LDOK bit remains logic 1 after being set until the values in DelayA and DelayB and modulus registers are loaded into buffers. This bit is readable.



**Figure 24-5. Registers Update with LDMOD = 0**



**Figure 24-6. Registers Update with LDMOD = 1**

The 16-bit counter operates on up-count mode. The two read-only counter registers contain the high and low bytes of the value in the PDB counter. Reading either byte latches the contents of both bytes into a buffer where they remain latched until either byte is read. Odd numbered reads return new data from the counter. Even numbered reads return latched data.

Pulse output can be driven on external pin when PDBxSCR[PADEN] is set.

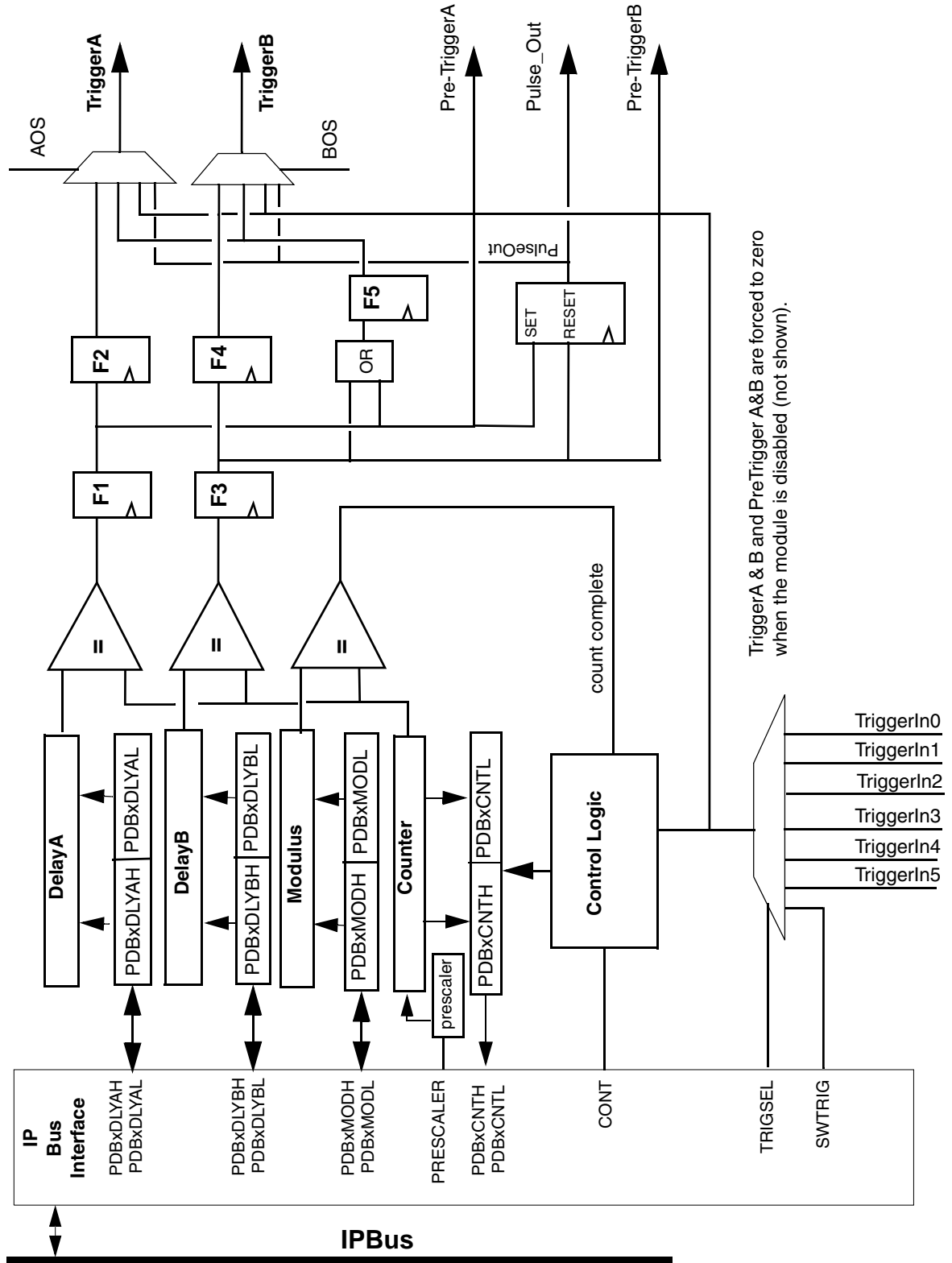


Figure 24-7. PDB Block Diagram

## 24.2 Memory Map/Register Definition

Table 24-2. PDB Memory Map

Offset	Register	Description
0x00	PDBxCTRL1	PDB control register 1
0x01	PDBxCTRL2	PDB control register 2
0x02	PDBxDLYAH	PDB delay A register high
0x03	PDBxDLYAL	PDB delay A register low
0x04	PDBxDLYBH	PDB delay B register high
0x05	PDBxDLYBL	PDB delay B register low
0x06	PDBxMODH	PDB counter modulus register high
0x07	PDBxMODL	PDB counter modulus register low
0x08	PDBxCNTH	PDB counter value register high
0x09	PDBxCNTL	PDB counter value register low
0x0A	PDBxSCR	PDB status & control register

### 24.2.1 PDB Control Register 1 (PDBxCTRL1)

This register contains control bits for the programmable delay block.

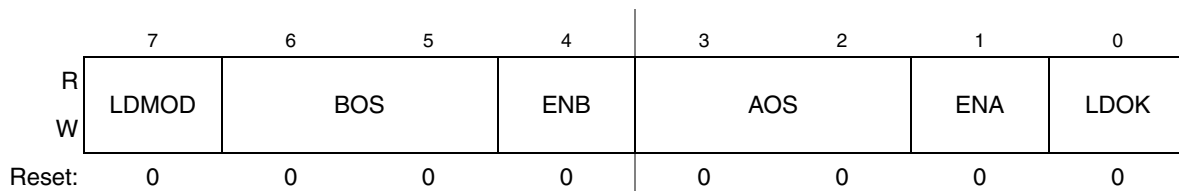


Figure 24-8. PDB Control Register 1 (PDBxCTRL1)

Table 24-3. PDBxCTRL1 Field Descriptions

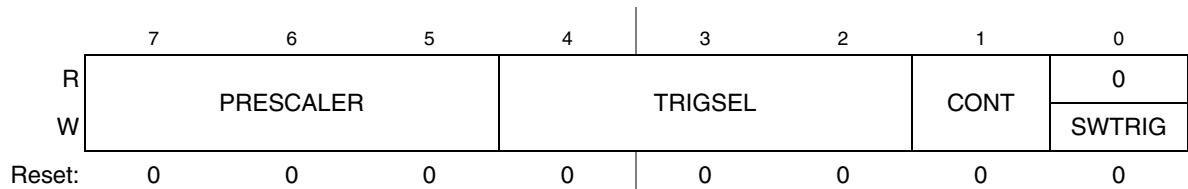
Field	Description
7 LDMOD	Load mode select 0 PDBxDLYAH, PDBxDLYAL, PDBxDLYBH, PDBxDLYBL, PDBxMODH, and PDBxMODL registers are loaded into a set of buffers and take effect immediately after logic 1 is written into LDOK bit 1 PDBxDLYAH, PDBxDLYAL, PDBxDLYBH, PDBxDLYBL, PDBxMODH, and PDBxMODL registers are loaded into a set of buffers and take effect when the counter rolls over in continuous mode or trigger signal is received in one shot mode after logic 1 is written into LDOK bit
6:5 BOS	Trigger B output select 00 Counter delay is bypassed 01 Trigger B is function of Delay B only 10 Trigger B is function of both Delay A and Delay B 11 Trigger B = PulseOut
4 ENB	Trigger B enable 0 Trigger B and PreTrigger B outputs are disabled (and forced to zero) 1 Trigger B and PreTrigger B outputs are enabled

**Table 24-3. PDBxCTRL1 Field Descriptions (continued)**

Field	Description
3–2 AOS	Trigger A output select 00 Counter delay is bypassed 01 Trigger A is function of Delay A only 10 Trigger A is function of both Delay A and Delay B 11 Trigger A = PulseOut
1 ENA	PreTrigger A enable 0 Trigger A and PreTrigger A outputs are disabled (and forced to zero) 1 Trigger A and PreTrigger A outputs are enabled
0 LDOK	Load OK- Writing logic 1 to this bit loads PDBxDLYAH, PDBxDLYAL, PDBxDLYBH, PDBxDLYBL, PDBxMODH, and PDBxMODL registers into a set of buffers. The buffered delayA, delayB, and modulus value take effect immediately if LDMOD = 0, or if the counter rolls over in continuous mode or trigger signal is received in one-shot mode when LDMOD = 1. Any value written to any one of these above registers, after a logic 1 is written to LDOK, is ignored until the values in these registers are loaded into the buffers. This bit is cleared when buffers are loaded. Writing logic 0 to this bit has no effect. Reading this bit can determine if the register values are loaded to the buffers and take effect.

### 24.2.2 PDB Control Register 2 (PDBxCTRL2)

This register contains PDB setting bits for the programmable delay block.



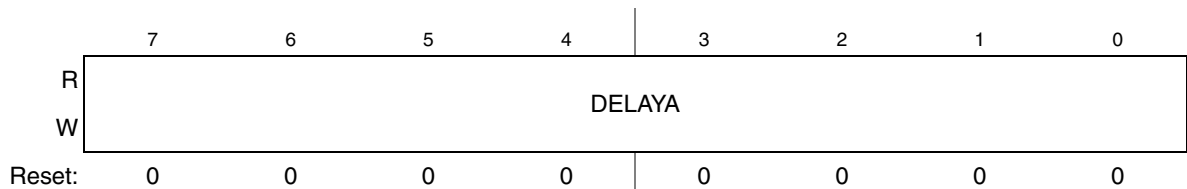
**Figure 24-9. PDB Control Register 2 (PDBxCTRL2)**

**Table 24-4. PDBxCTRL2 Field Descriptions**

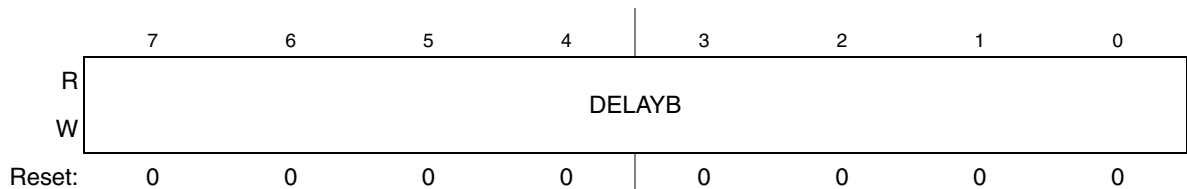
Field	Description
7–5 PRESCALER	Clock prescaler select 000 Peripheral clock 001 Peripheral clock / 2 010 Peripheral clock / 4 011 Peripheral clock / 8 100 Peripheral clock / 16 101 Peripheral clock / 32 110 Peripheral clock / 64 111 Peripheral clock / 128
4–2 TRIGSEL	Input trigger select See the <a href="#">Introduction</a> section for which device-specific signals are connected to the triggers. 000 TriggerIn0 001 TriggerIn1 010 TriggerIn2 011 TriggerIn3 100 TriggerIn4 101 TriggerIn5 110 TriggerIn6 111 SWTRIG
1 CONT	Continuous mode enable 0 Module is in one-shot mode 1 Module is in continuous mode
0 SWTRIG	Software trigger- When TRIGSEL = 111 and the module is enabled, writing a one to this field triggers a reset and restart of the counter.

### 24.2.3 PDB Delay Registers (PDBxDLYAL, PDBxDLYAH, PDBxDLYBL, & PDBxDLYBH)

These registers specify the delay from assertion of TriggerIn to assertion of TriggerA and TriggerB out. This delay is only applicable if the module is enabled and the corresponding output trigger has not been bypassed. In each case, the delay is in terms of peripheral clock cycles. A delay value of 0x0000 is never reached and no output trigger occurs since the counter goes from 0x0001 to 0xFFFF.



**Figure 24-10. PDB Delay A Registers (PDBxDLYAL & PDBxDLYAH)**



**Figure 24-12. PDB Delay B Registers (PDBxDLYBL & PDBxDLYBH)**

### 24.2.4 PDB Modulus Registers (PDBxMODH & PDBxMODL)

These registers specify the period of the counter in terms of peripheral bus cycles. When the counter reaches this value, it is reset back to one. If PDBxCTRL2[CONT] is set, the count begins a new round.

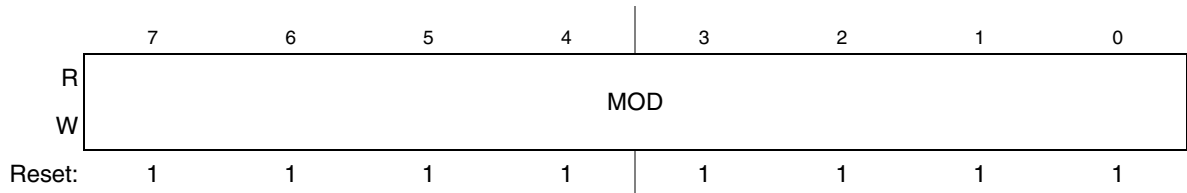


Figure 24-13. PDB Modulus Registers (PDBxMODL & PDBxMODH)

### 24.2.5 PDB Counter Registers (PDBxCNTH & PDBxCNTL)

This is a 16 bit counter which operates in up-count mode. The two read-only counter registers contain the high and low bytes of the value in the PDB counter. Reading either byte latches the contents of both bytes into a buffer where they remain latched until PDBxCNTH or PDBxCNTL is read.

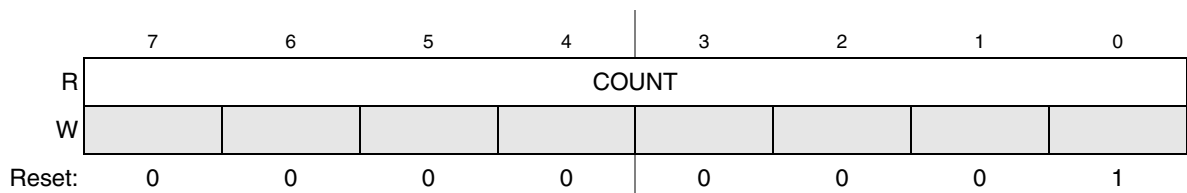


Figure 24-14. PDB Counter Registers (PDBxCNTL & PDBxCNTH)

### 24.2.6 PDB Status and Control Register (PDBxSCR)

This register contains status and control bits for the programmable delay block.

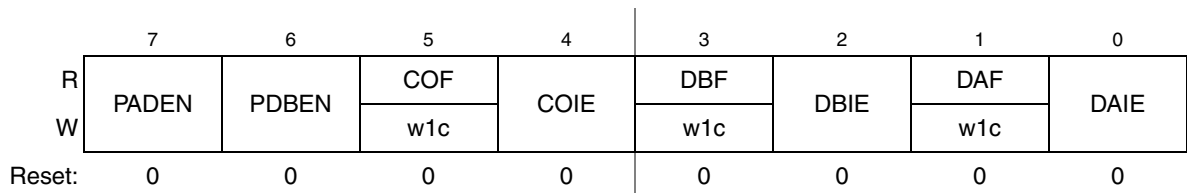


Figure 24-15. PDB Status and Control Register (PDBxSCR)



**Table 24-5. PDBxSCR Field Descriptions**

Field	Description
7 PADEN	Pulse output PAD enable 0 Pulse output does not control external pin 1 Pulse output is driven on external pin <b>Note:</b> Depending on clock speed, a single pulse may not propagate through the I/O pad.
6 PDBEN	PDB module enable 0 Counter is off and Pre-TriggerA, TriggerA, Pre-TriggerB, and TriggerB are low 1 Counter is enabled
5 COF	Counter overflow flag This bit is set when a successful compare of the counter value and modulus occurs then counter is rolled over. Clear this bit by writing logic one to it.
4 COIE	Counter overflow interrupt request enable 0 Disabled 1 Enabled
3 DBF	Delay B flag This bit is set when a successful compare of the counter value and delay B occurs. Clear this bit by writing logic one to it.
2 DBIE	Delay B successful compare interrupt request enable 0 Disabled 1 Enabled
1 DAF	Delay A flag This bit is set when a successful compare of the counter value and delay A occurs. Clear this bit by writing logic one to it.
0 DAIE	Delay A successful compare interrupt request enable 0 Disabled 1 Enabled

## 24.3 Functional Description

### 24.3.1 Miscellaneous Concerns and SoC Integration

- The purposes of this block are to
  - Manage the delay between an external event and the time at which comparator, ADC or PGA sample(s) are taken, and
  - Generate a variable width pulse which is synchronize with PWM timer module
- Trigger A and Trigger B are defined to be glitch free
- The PDB must correctly respond to an external trigger
- Additional trigger events, after the first, cause the counter to restart even if the counter is still counting from the previous trigger

## 24.3.2 Impacts of using the prescaler on timing resolution

Using a prescaler greater than one limits the count/delay accuracy in terms of peripheral clocks (to the modulus of the prescaler value). If the prescaler is set to divide-by-2, then the only values of total peripheral clocks that can be detected are even values. If using divide-by-4, then the only values of total peripheral clocks that can be decoded as detected are mod(4) and so forth. If you want to set a long delay value and use div 128, then you are limited to an resolution of 128 bus clocks. Therefore, use the lowest possible prescaler for a given application.

## 24.3.3 Resets

This module has a single reset input, corresponding to the chip-wide peripheral reset. After reset, all registers are set to their reset value.

## 24.3.4 Clocks

This module has a single clock input, the IP Bus peripheral clock.

## 24.3.5 Interrupts

This module has no interrupts.

# Chapter 25

## Timer/PWM Module (S08TPMV3)

### 25.1 Introduction

The TPM is a one-to-eight-channel timer system that supports traditional input capture, output compare, or edge-aligned PWM on each channel. A control bit configures the TPM so all channels are used for center-aligned PWM functions. Timing functions are based on a 16-bit counter with prescaler and modulo features to control frequency and range (period between overflows) of the time reference. This timing system is ideally suited for a wide range of control applications, and the center-aligned PWM capability extends the field of application to motor control in small appliances.

#### 25.1.1 TPM Clock Gating

The bus clock to the TPM module can be gated on and off using the SCGC1[TPM3] bit. This bit is cleared after any reset that disables the bus clock to this module. The SCGC1[TPM3] bit must be set before operation. See [Section 5.7, “Peripheral Clock Gating,”](#) for details.

#### 25.1.2 TPM3 Module Interconnects

##### 25.1.2.1 TPM3 Input Capture

The HSCMP2 module can be configured to connect the output of the analog comparator to TPM3 input capture channel 0 by setting the SOPT2[ACIC2] bit. With ACIC2 set, the TPM3CH0 pin is not available externally regardless of the configuration of the TPM3 module.

### 25.1.3 Features

The TPM includes these distinctive features:

- One to eight channels:
  - Each channel is input capture, output compare, or edge-aligned PWM
  - Rising-edge, falling-edge, or any-edge input capture trigger
  - Set, clear, or toggle output compare action
  - Selectable polarity on PWM outputs
- Module is configured for buffered, center-aligned pulse-width-modulation (CPWM) on all channels
- Timer clock source selectable as bus clock, fixed frequency clock, or an external clock
  - Prescale taps for divide-by 1, 2, 4, 8, 16, 32, 64, or 128 used for any clock input selection
  - Fixed frequency clock is an additional clock input to allow the selection of an on chip clock source other than bus clock
  - Selecting external clock connects TPM clock to a chip level input pin therefore allowing to synchronize the TPM counter with an off chip clock source
- 16-bit free-running or modulus count with up/down selection
- One interrupt per channel and one interrupt for TPM counter overflow

### 25.1.4 Modes of Operation

In general, TPM channels are independently configured to operate in input capture, output compare, or edge-aligned PWM modes. A control bit allows the whole TPM (all channels) to switch to center-aligned PWM mode. When center-aligned PWM mode is selected, input capture, output compare, and edge-aligned PWM functions are not available on any channels of this TPM module.

When the MCU is in active BDM background or BDM foreground mode, the TPM temporarily suspends all counting until the MCU returns to normal user operating mode. During stop mode, all TPM input clocks are stopped, so the TPM is effectively disabled until clocks resume. During wait mode, the TPM continues to operate normally. If the TPM does not need to produce a real time reference or provide the interrupt sources needed to wake the MCU from wait mode, the power can then be saved by disabling TPM functions before entering wait mode.

- Input capture mode
 

When a selected edge event occurs on the associated MCU pin, the current value of the 16-bit timer counter is captured into the channel value register and an interrupt flag bit is set. Rising edges, falling edges, any edge, or no edge (disable channel) are selected as the active edge that triggers the input capture.
- Output compare mode
 

When the value in the timer counter register matches the channel value register, an interrupt flag bit is set, and a selected output action is forced on the associated MCU pin. The output compare action is selected to force the pin to zero, force the pin to one, toggle the pin, or ignore the pin (used for software timing functions).
- Edge-aligned PWM mode

The value of a 16-bit modulo register plus 1 sets the period of the PWM output signal. The channel value register sets the duty cycle of the PWM output signal. You can also choose the polarity of the PWM output signal. Interrupts are available at the end of the period and at the duty-cycle transition point. This type of PWM signal is called edge-aligned because the leading edges of all PWM signals are aligned with the beginning of the period that is same for all channels within a TPM.

- Center-aligned PWM mode

Twice the value of a 16-bit modulo register sets the period of the PWM output, and the channel-value register sets the half-duty-cycle duration. The timer counter counts up until it reaches the modulo value and then counts down until it reaches zero. As the count matches the channel value register while counting down, the PWM output becomes active. When the count matches the channel value register while counting up, the PWM output becomes inactive. This type of PWM signal is called center-aligned because the centers of the active duty cycle periods for all channels are aligned with a count value of zero. This type of PWM is required for types of motors used in small appliances.

This is a high-level description only. Detailed descriptions of operating modes are in later sections.

### 25.1.5 Block Diagram

The TPM uses one input/output (I/O) pin per channel, TPMxCHn (timer channel n) where n is the channel number (1–8). The TPM shares its I/O pins with general purpose I/O port pins (refer to I/O pin descriptions in full-chip specification for the specific chip implementation).

[Figure 25-1](#) shows the TPM structure. The central component of the TPM is the 16-bit counter that can operate as a free-running counter or a modulo up/down counter. The TPM counter (when operating in normal up-counting mode) provides the timing reference for the input capture, output compare, and edge-aligned PWM functions. The timer counter modulo registers, TPMxMODH:TPMxMODL, control the modulo value of the counter (the values 0x0000 or 0xFFFF effectively make the counter free running). Software can read the counter value at any time without affecting the counting sequence. Any write to either half of the TPMxCNT counter resets the counter, regardless of the data value written.

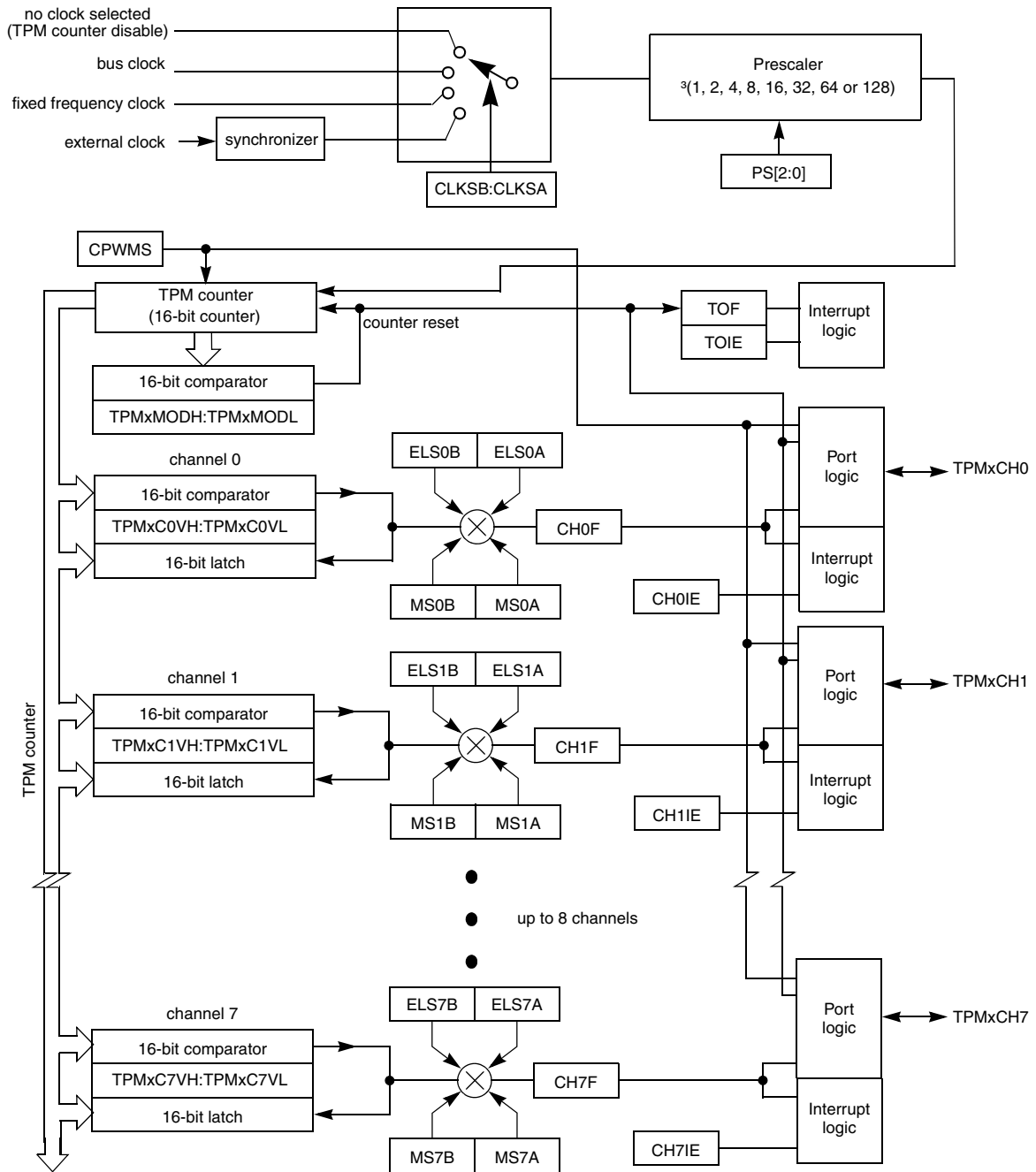


Figure 25-1. TPM Block Diagram

The TPM channels are programmable independently as input capture, output compare, or edge-aligned PWM channels. Alternately, the TPM can be configured to produce CPWM outputs on all channels. When the TPM is configured for CPWMs (the counter operates as an up/down counter) input capture, output compare, and EPWM functions are not practical.

## 25.2 Signal Description

Table 25-1 shows the user-accessible signals for the TPM. The number of channels are varied from one to eight. When an external clock is included, it can be shared with the same pin as any TPM channel; however, it could be connected to a separate input pin. Refer to the I/O pin descriptions in full-chip specification for the specific chip implementation.

**Table 25-1. Signal Properties**

Name	Function
EXTCLK <sup>1</sup>	External clock source that is selected to drive the TPM counter.
TPMxCHn <sup>2</sup>	I/O pin associated with TPM channel n.

<sup>1</sup> The external clock pin can be shared with any channel pin. However, depending upon full-chip implementation, this signal could be connected to a separate external pin.

<sup>2</sup> n = channel number (1–8)

### 25.2.1 Detailed Signal Descriptions

#### 25.2.1.1 EXTCLK — External Clock Source

The external clock signal can share the same pin as a channel pin, however the channel pin can not be used for channel I/O function when external clock is selected. If this pin is used as an external clock (CLKSB:CLKSA = 1:1), the channel can still be configured to output compare mode therefore allowing its use as a timer (ELSnB:ELSnA = 0:0).

For proper TPM operation, the external clock frequency must not exceed one-fourth of the bus clock frequency.

#### 25.2.1.2 TPMxCHn — TPM Channel n I/O Pins

The TPM channel does not control the I/O pin when ELSnB:ELSnA or CLKSb:CLKSA are cleared so it normally reverts to general purpose I/O control. When CPWMS is set and ELSnB:ELSnA are not cleared, all TPM channels are configured for center-aligned PWM and the TPMxCHn pins are all controlled by TPM. When CPWMS is cleared, the MSnB:MSnA control bits determine whether the channel is configured for input capture, output compare, or edge-aligned PWM.

When a channel is configured for input capture (CPWMS = 0, MSnB:MSnA = 0:0, and ELSnB:ELSnA ≠ 0:0), the TPMxCHn pin is forced to act as an edge-sensitive input to the TPM. ELSnB:ELSnA control bits determine what polarity edge or edges trigger input capture events. The channel input signal is synchronized on the bus clock. This implies the minimum pulse width—that can

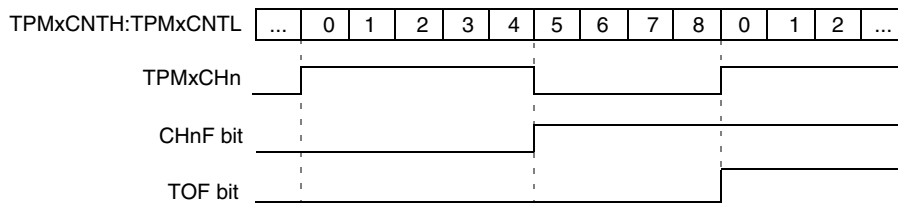
be reliably detected—on an input capture pin is four bus clock periods (with ideal clock pulses as near as two bus clocks can be detected).

When a channel is configured for output compare (CPWMS = 0, MSnB:MSnA = 0:1, and ELSnB:ELSnA ≠ 0:0), the TPMxCHn pin is an output controlled by the TPM. The ELSnB:ELSnA bits determine whether the TPMxCHn pin is toggled, cleared, or set each time the 16-bit channel value register matches the TPM counter.

When the output compare toggle mode is initially selected, the previous value on the pin is driven out until the next output compare event, the pin is then toggled.

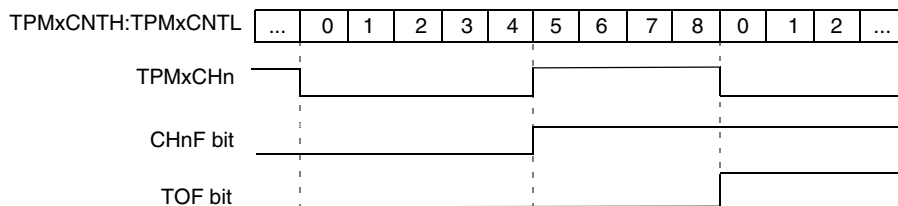
When a channel is configured for edge-aligned PWM (CPWMS = 0, MSnB = 1, and ELSnB:ELSnA ≠ 0:0), the TPMxCHn pin is an output controlled by the TPM, and ELSnB:ELSnA bits control the polarity of the PWM output signal. When ELSnB is set and ELSnA is cleared, the TPMxCHn pin is forced high at the start of each new period (TPMxCNT=0x0000), and it is forced low when the channel value register matches the TPM counter. When ELSnA is set, the TPMxCHn pin is forced low at the start of each new period (TPMxCNT=0x0000), and it is forced high when the channel value register matches the TPM counter.

TPMxMODH:TPMxMODL = 0x0008  
 TPMxCnVH:TPMxCnVL = 0x0005



**Figure 25-2. High-true pulse of an edge-aligned PWM**

TPMxMODH:TPMxMODL = 0x0008  
 TPMxCnVH:TPMxCnVL = 0x0005



**Figure 25-3. Low-true pulse of an edge-aligned PWM**

When the TPM is configured for center-aligned PWM (CPWMS = 1 and ELSnB:ELSnA ≠ 0:0), the TPMxCHn pins are outputs controlled by the TPM, and ELSnB:ELSnA bits control the polarity of the PWM output signal. If ELSnB is set and ELSnA is cleared, the corresponding TPMxCHn pin is cleared when the TPM counter is counting up, and the channel value register matches the TPM counter; and it is



set when the TPM counter is counting down, and the channel value register matches the TPM counter. If ELSnA is set, the corresponding TPMxCHn pin is set when the TPM counter is counting up and the channel value register matches the TPM counter; and it is cleared when the TPM counter is counting down and the channel value register matches the TPM counter.

TPMxMODH:TPMxMODL = 0x0008  
 TPMxCnVH:TPMxCnVL = 0x0005

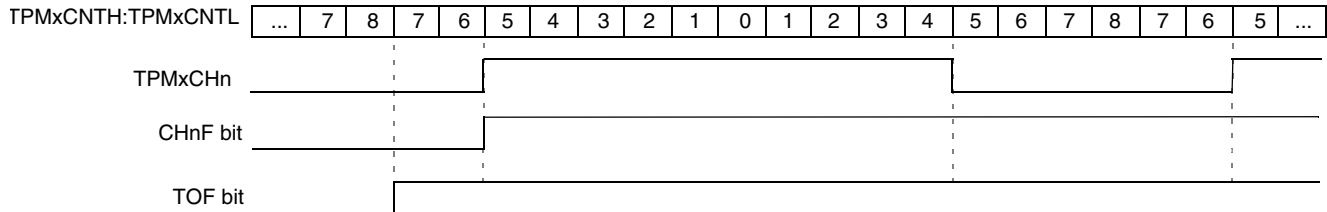


Figure 25-4. High-true pulse of a center-aligned PWM

TPMxMODH:TPMxMODL = 0x0008  
 TPMxCnVH:TPMxCnVL = 0x0005

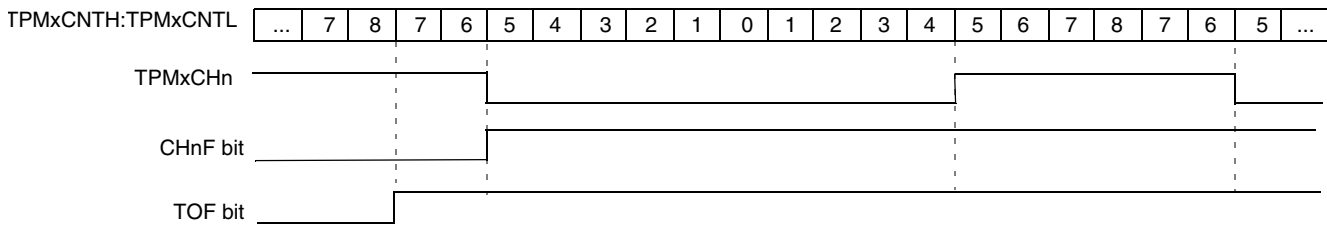


Figure 25-5. Low-true pulse of a center-aligned PWM

## 25.3 Register Definition

### 25.3.1 TPM Status and Control Register (TPMxSC)

TPMxSC contains the overflow status flag and control bits used to configure the interrupt enable, TPM configuration, clock source, and prescale factor. These controls relate to all channels within this timer module.

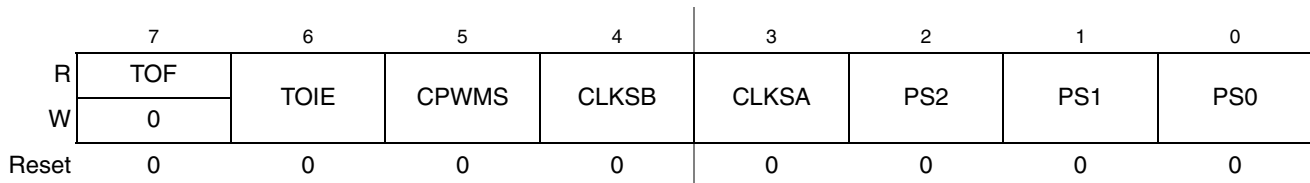


Figure 25-6. TPM Status and Control Register (TPMxSC)

Table 25-2. TPMxSC Field Descriptions

Field	Description
7 TOF	Timer overflow flag. This read/write flag is set when the TPM counter resets to 0x0000 after reaching the modulo value programmed in the TPM counter modulo registers. Clear TOF by reading the TPM status and control register when TOF is set and then writing a logic 0 to TOF. If another TPM overflow occurs before the clearing sequence is completed, the sequence is reset so TOF remains set after the clear sequence was completed for the earlier TOF. This is done so a TOF interrupt request cannot be lost during the clearing sequence for a previous TOF. Reset clears TOF. Writing a logic 1 to TOF has no effect. 0 TPM counter has not reached modulo value or overflow. 1 TPM counter has overflowed.
6 TOIE	Timer overflow interrupt enable. This read/write bit enables TPM overflow interrupts. If TOIE is set, an interrupt is generated when TOF equals one. Reset clears TOIE. 0 TOF interrupts inhibited (use for software polling). 1 TOF interrupts enabled.
5 CPWMS	Center-aligned PWM select. This read/write bit selects CPWM operating mode. By default, the TPM operates in up-counting mode for input capture, output compare, and edge-aligned PWM functions. Setting CPWMS reconfigures the TPM to operate in up/down counting mode for CPWM functions. Reset clears CPWMS. 0 All channels operate as input capture, output compare, or edge-aligned PWM mode as selected by the MSnB:MSnA control bits in each channel's status and control register. 1 All channels operate in center-aligned PWM mode.
4–3 CLKS[B:A]	Clock source selection bits. As shown in <a href="#">Table 25-3</a> , this 2-bit field is used to disable the TPM counter or select one of three clock sources to TPM counter and counter prescaler.
2–0 PS[2:0]	Prescale factor select. This 3-bit field selects one of eight division factors for the TPM clock as shown in <a href="#">Table 25-4</a> . This prescaler is located after any clock synchronization or clock selection so it affects the clock selected to drive the TPM counter. The new prescale factor affects the selected clock on the next bus clock cycle after the new value is updated into the register bits.

Table 25-3. TPM Clock Selection

CLKSB:CLKSA	TPM Clock to Prescaler Input
00	No clock selected (TPM counter disable)

**Table 25-3. TPM Clock Selection**

CLKSB:CLKSA	TPM Clock to Prescaler Input
01	Bus clock
10	Fixed frequency clock
11	External clock

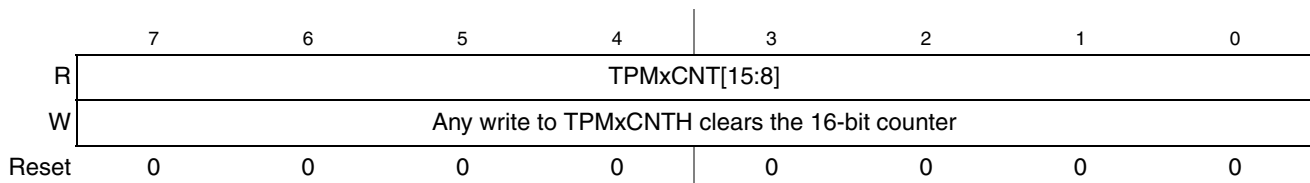
**Table 25-4. Prescale Factor Selection**

PS[2:0]	TPM Clock Divided-by
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

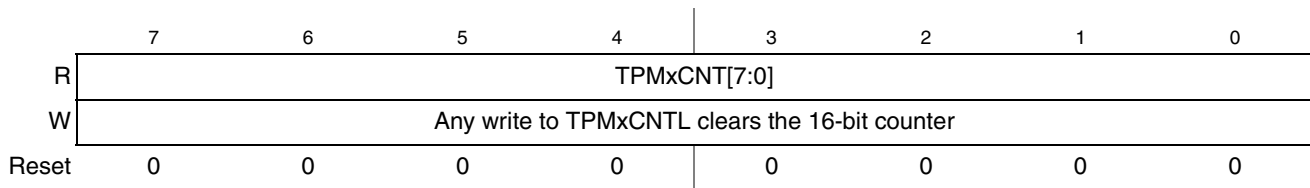
### 25.3.2 TPM-Counter Registers (TPMxCNTH:TPMxCNTL)

The two read-only TPM counter registers contain the high and low bytes of the value in the TPM counter. Reading either byte (TPMxCNTH or TPMxCNTL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This allows coherent 16-bit reads in big-endian or little-endian order that makes this more friendly to various compiler implementations. The coherency mechanism is automatically restarted by an MCU reset or any write to the timer status/control register (TPMxSC).

Reset clears the TPM counter registers. Writing any value to TPMxCNTH or TPMxCNTL also clears the TPM counter (TPMxCNTH:TPMxCNTL) and resets the coherency mechanism, regardless of the data involved in the write.



**Figure 25-7. TPM Counter Register High (TPMxCNTH)**



**Figure 25-8. TPM Counter Register Low (TPMxCNTL)**

When BDM is active, the timer counter is frozen (this is the value you read). The coherency mechanism is frozen so the buffer latches remain in the state they were in when the BDM became active, even if one or both counter halves are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

In BDM mode, writing any value to TPMxSC, TPMxCNTH, or TPMxCNTL registers resets the read coherency mechanism of the TPMxCNTH:TPMxCNTL registers, regardless of the data involved in the write.

### 25.3.3 TPM Counter Modulo Registers (TPMxMODH:TPMxMODL)

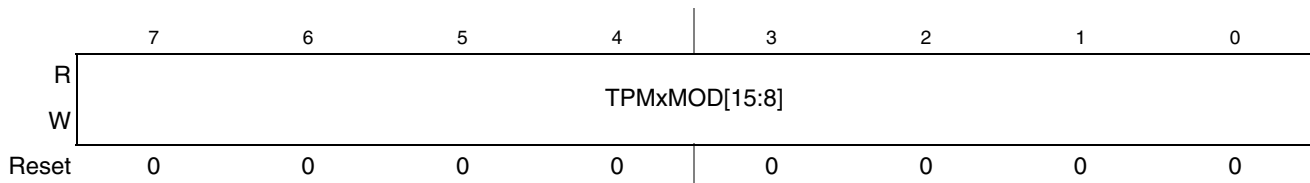
The read/write TPM modulo registers contain the modulo value for the TPM counter. After the TPM counter reaches the modulo value, the TPM counter resumes counting from 0x0000 at the next clock, and the overflow flag (TOF) becomes set. Writing to TPMxMODH or TPMxMODL inhibits the TOF bit and overflow interrupts until the other byte is written. Reset sets the TPM counter modulo registers to 0x0000 that results in a free running timer counter (modulo disabled).

Writes to any of the registers TPMxMODH and TPMxMODL actually writes to buffer registers and the registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

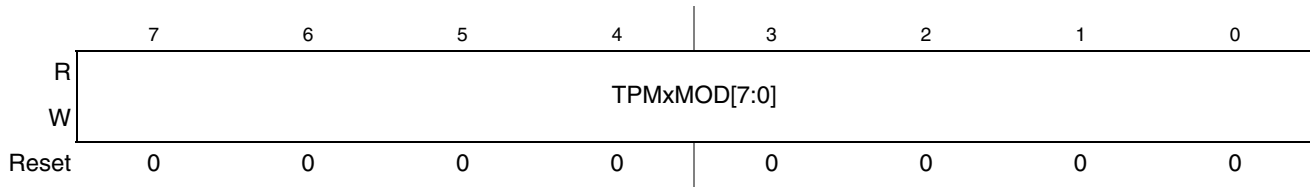
- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFFE to 0xFFFF

The latching mechanism is manually reset by writing to the TPMxSC address (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any write to the modulo registers bypasses the buffer latches and directly writes to the modulo register while BDM is active.



**Figure 25-9. TPM Counter Modulo Register High (TPMxMODH)**



Reset the TPM counter before writing to the TPM modulo registers to avoid confusion about when the first counter overflow occurs.

### 25.3.4 TPM Channel n Status and Control Register (TPMxCnSC)

TPMxCnSC contains the channel-interrupt-status flag and control bits that configure the interrupt enable, channel configuration, and pin function.

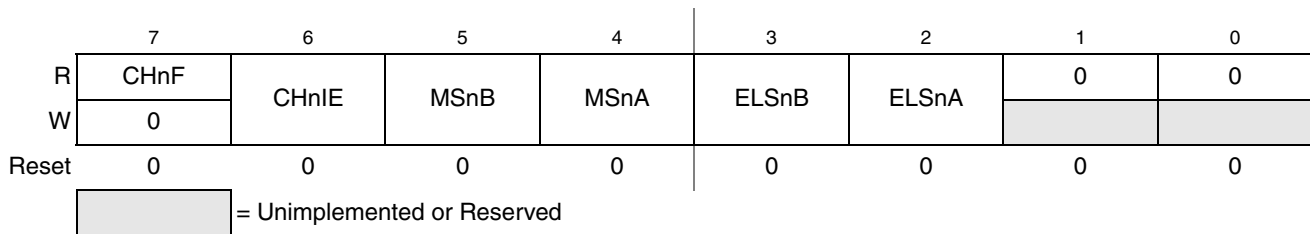


Figure 25-11. TPM Channel n Status and Control Register (TPMxCnSC)

Table 25-5. TPMxCnSC Field Descriptions

Field	Description
7 CHnF	<p>Channel n flag. When channel n is an input capture channel, this read/write bit is set when an active edge occurs on the channel n input. When channel n is an output compare or edge-aligned/center-aligned PWM channel, CHnF is set when the value in the TPM counter registers matches the value in the TPM channel n value registers. When channel n is an edge-aligned/center-aligned PWM channel and the duty cycle is set to 0% or 100%, CHnF is not set even when the value in the TPM counter registers matches the value in the TPM channel n value registers.</p> <p>A corresponding interrupt is requested when this bit is set and channel n interrupt is enabled (CHnIE = 1). Clear CHnF by reading TPMxCnSC while this bit is set and then writing a logic 0 to it. If another interrupt request occurs before the clearing sequence is completed CHnF remains set. This is done so a CHnF interrupt request is not lost due to clearing a previous CHnF.</p> <p>Reset clears this bit. Writing a logic 1 to CHnF has no effect.</p> <p>0 No input capture or output compare event occurred on channel n. 1 Input capture or output compare event on channel n.</p>
6 CHnIE	<p>Channel n interrupt enable. This read/write bit enables interrupts from channel n. Reset clears this bit.</p> <p>0 Channel n interrupt requests disabled (use for software polling). 1 Channel n interrupt requests enabled.</p>
5 MSnB	<p>Mode select B for TPM channel n. When CPWMS is cleared, setting the MSnB bit configures TPM channel n for edge-aligned PWM mode. Refer to the summary of channel mode and setup controls in <a href="#">Table 25-6</a>.</p>

Table 25-5. TPMxCnSC Field Descriptions (continued)

Field	Description
4 MSnA	Mode select A for TPM channel n. When CPWMS and MSnB are cleared, the MSnA bit configures TPM channel n for input capture mode or output compare mode. Refer to <a href="#">Table 25-6</a> for a summary of channel mode and setup controls. <b>Note:</b> If the associated port pin is not stable for at least two bus clock cycles before changing to input capture mode, it is possible to get an unexpected indication of an edge trigger.
3–2 ELSnB ELSnA	Edge/level select bits. Depending upon the operating mode for the timer channel as set by CPWMS:MSnB:MSnA and shown in <a href="#">Table 25-6</a> , these bits select the polarity of the input edge that triggers an input capture event, select the level that is driven in response to an output compare match, or select the polarity of the PWM output. If ELSnB and ELSnA bits are cleared, the channel pin is not controlled by TPM. This configuration can be used by software compare only, because it does not require the use of a pin for the channel.

Table 25-6. Mode, Edge, and Level Selection

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	XX	00	Pin is not controlled by TPM. It is reverted to general purpose I/O or other peripheral control	
0	00	01	Input capture	Capture on rising edge only
		10		Capture on falling edge only
		11		Capture on rising or falling edge
	01	00	Output compare	Software compare only
		01		Toggle output on channel match
		10		Clear output on channel match
		11		Set output on channel match
	1X	10	Edge-aligned PWM	High-true pulses (clear output on channel match)
X1		Low-true pulses (set output on channel match)		
1	XX	10	Center-aligned PWM	High-true pulses (clear output on channel match when TPM counter is counting up)
		X1		Low-true pulses (set output on channel match when TPM counter is counting up)

### 25.3.5 TPM Channel Value Registers (TPMxCnVH:TPMxCnVL)

These read/write registers contain the captured TPM counter value of the input capture function or the output compare value for the output compare or PWM functions. The channel registers are cleared by reset.

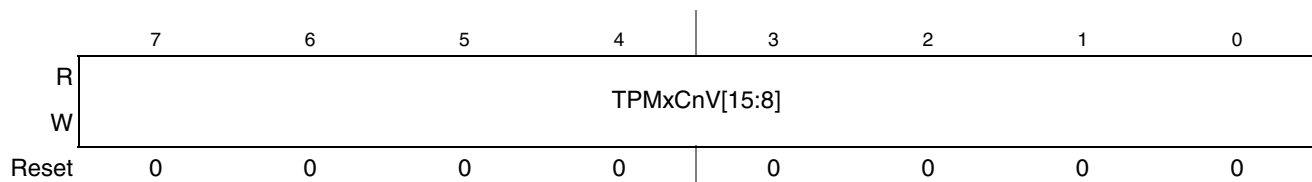


Figure 25-12. TPM Channel Value Register High (TPMxCnVH)



**Figure 25-13. TPM Channel Value Register Low (TPMxCnVL)**

In input capture mode, reading either byte (TPMxCnVH or TPMxCnVL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This latching mechanism also resets (becomes unlatched) when the TPMxCnSC register is written (whether BDM mode is active or not). Any write to the channel registers is ignored during the input capture mode.

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxCnSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the channel register are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. The value read from the TPMxCnVH and TPMxCnVL registers in BDM mode is the value of these registers and not the value of their read buffer.

In output compare or PWM modes, writing to either byte (TPMxCnVH or TPMxCnVL) latches the value into a buffer. After both bytes were written, they are transferred as a coherent 16-bit value into the timer-channel registers according to the value of CLKSB:CLKSA bits and the selected mode:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared and in output compare mode, the registers are updated after the second byte is written and on the next change of the TPM counter (end of the prescaler counting).
- If CLKSB and CLKSA are not cleared and in EPWM or CPWM modes, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFFE to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxCnSC register (whether BDM mode is active or not). This latching mechanism allows coherent 16-bit writes in either big-endian or little-endian order that is friendly to various compiler implementations.

When BDM is active, the coherency mechanism is frozen so the buffer latches remain in the state they were in when the BDM became active even if one or both halves of the channel register are written while BDM is active. Any write to the channel registers bypasses the buffer latches and directly write to the channel register while BDM is active. The values written to the channel register while BDM is active are used for PWM and output compare operation after normal execution resumes. Writes to the channel registers while BDM is active do not interfere with partial completion of a coherency sequence. After the coherency mechanism is fully exercised, the channel registers are updated using the buffered values (while BDM was not active).

## 25.4 Functional Description

All TPM functions are associated with a central 16-bit counter that allows flexible selection of the clock and prescale factor. There is also a 16-bit modulo register associated with this counter.

The CPWMS control bit chooses between center-aligned PWM operation for all channels in the TPM (CPWMS=1) or general purpose timing functions (CPWMS=0) where each channel can independently be configured to operate in input capture, output compare, or edge-aligned PWM mode. The CPWMS control bit is located in the TPM status and control register because it affects all channels within the TPM and influences the way the main counter operates. (In CPWM mode, the counter changes to an up/down mode rather than the up-counting mode used for general purpose timer functions.)

The following sections describe TPM counter and each of the timer operating modes (input capture, output compare, edge-aligned PWM, and center-aligned PWM). Because details of pin operation and interrupt activity depend upon the operating mode, these topics are covered in the associated mode explanation sections.

### 25.4.1 Counter

All timer functions are based on the main 16-bit counter (TPMxCNTH:TPMxCNTL). This section discusses selection of the clock, end-of-count overflow, up-counting vs. up/down counting, and manual counter reset.

#### 25.4.1.1 Counter Clock Source

The 2-bit field, CLKSB:CLKSA, in the timer status and control register (TPMxSC) disables the TPM counter or selects one of three clock sources to TPM counter (Table 25-3). After any MCU reset, CLKSB and CLKSA are cleared so no clock is selected and the TPM counter is disabled (TPM is in a very low power state). You can read or write these control bits at any time. Disabling the TPM counter by writing 00 to CLKSB:CLKSA bits, does not affect the values in the TPM counter or other registers.

The fixed frequency clock is an alternative clock source for the TPM counter that allows the selection of a clock other than the bus clock or external clock. This clock input is defined by chip integration. You can refer chip specific documentation for further information. Due to TPM hardware implementation limitations, the frequency of the fixed frequency clock must not exceed the bus clock frequency. The fixed frequency clock has no limitations for low frequency operation.

The external clock passes through a synchronizer clocked by the bus clock to assure that counter transitions are properly aligned to bus clock transitions. Therefore, in order to meet Nyquist criteria considering also jitter, the frequency of the external clock source must not exceed 1/4 of the bus clock frequency.

When the external clock source is shared with a TPM channel pin, this pin must not be used in input capture mode. However, this channel can be used in output compare mode with ELSnB:ELSnA = 0:0 for software timing functions. In this case, the channel output is disabled, but the channel match events continue to set the appropriate flag.



### 25.4.1.2 Counter Overflow and Modulo Reset

An interrupt flag and enable are associated with the 16-bit main counter. The flag (TOF) is a software-accessible indication that the timer counter has overflowed. The enable signal selects between software polling (TOIE = 0) where no interrupt is generated, or interrupt-driven operation (TOIE = 1) where the interrupt is generated whenever the TOF is set.

The conditions causing TOF to become set depend on whether the TPM is configured for center-aligned PWM (CPWMS = 1). If CPWMS is cleared and there is no modulus limit, the 16-bit timer counter counts from 0x0000 through 0xFFFF and overflows to 0x0000 on the next counting clock. TOF is set at the transition from 0xFFFF to 0x0000. When a modulus limit is set, TOF is set at the transition from the value set in the modulus register to 0x0000. When the TPM is in center-aligned PWM mode (CPWMS = 1), the TOF flag is set as the counter changes direction at the end of the count value set in the modulus register (at the transition from the value set in the modulus register to the next lower count value). This corresponds to the end of a PWM period (the 0x0000 count value corresponds to the center of a period).

### 25.4.1.3 Counting Modes

The main timer counter has two counting modes. When center-aligned PWM is selected (CPWMS = 1), the counter operates in up/down counting mode. Otherwise, the counter operates as a simple up counter. As an up counter, the timer counter counts from 0x0000 through its terminal count and continues with 0x0000. The terminal count is 0xFFFF or a modulus value in TPMxMODH:TPMxMODL.

When center-aligned PWM operation is specified, the counter counts up from 0x0000 through its terminal count and then down to 0x0000 where it changes back to up counting. The terminal count value and 0x0000 are normal length counts (one timer clock period long). In this mode, the timer overflow flag (TOF) is set at the end of the terminal-count period (as the count changes to the next lower count value).

### 25.4.1.4 Manual Counter Reset

The main timer counter can be manually reset at any time by writing any value to TPMxCNTH or TPMxCNTL. Resetting the counter in this manner also resets the coherency mechanism in case only half of the counter was read before resetting the count.

## 25.4.2 Channel Mode Selection

If CPWMS is cleared, MSnB and MSnA bits determine the basic mode of operation for the corresponding channel. Choices include input capture, output compare, and edge-aligned PWM.

### 25.4.2.1 Input Capture Mode

With the input capture function, the TPM can capture the time at which an external event occurs. When an active edge occurs on the pin of an input capture channel, the TPM latches the contents of the TPM counter into the channel-value registers (TPMxCnVH:TPMxCnVL). Rising edges, falling edges, or any edge is chosen as the active edge that triggers an input capture.

In input capture mode, the TPMxCnVH and TPMxCnVL registers are read only.

When either half of the 16-bit capture register is read, the other half is latched into a buffer to support coherent 16-bit accesses in big-endian or little-endian order. The coherency sequence can be manually reset by writing to TPMxCnSC.

An input capture event sets a flag bit (CHnF) that optionally generates a CPU interrupt request.

While in BDM, the input capture function works as configured. When an external event occurs, the TPM latches the contents of the TPM counter (frozen because of the BDM mode) into the channel value registers and sets the flag bit.

### 25.4.2.2 Output Compare Mode

With the output compare function, the TPM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter reaches the value in TPMxCnVH:TPMxCnVL registers of an output compare channel, the TPM can set, clear, or toggle the channel pin.

Writes to any of TPMxCnVH and TPMxCnVL registers actually write to buffer registers. In output compare mode, the TPMxCnVH:TPMxCnVL registers are updated with the value of their write buffer only after both bytes were written and according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated at the next change of the TPM counter (end of the prescaler counting) after the second byte is written.

The coherency sequence can be manually reset by writing to the channel status/control register (TPMxCnSC).

An output compare event sets a flag bit (CHnF) that optionally generates a CPU interrupt request.

### 25.4.2.3 Edge-Aligned PWM Mode

This type of PWM output uses the normal up-counting mode of the timer counter (CPWMS=0) and can be used when other channels in the same TPM are configured for input capture or output compare functions. The period of this PWM signal is determined by the value of the modulus register (TPMxMODH:TPMxMODL) plus 1. The duty cycle is determined by the value of the timer channel register (TPMxCnVH:TPMxCnVL). The polarity of this PWM signal is determined by ELSnA bit. 0% and 100% duty cycle cases are possible.

The time between the modulus overflow and the channel match value (TPMxCnVH:TPMxCnVL) is the pulse width or duty cycle (Figure 25-14). If ELSnA is cleared, the counter overflow forces the PWM signal high, and the channel match forces the PWM signal low. If ELSnA is set, the counter overflow forces the PWM signal low, and the channel match forces the PWM signal high.

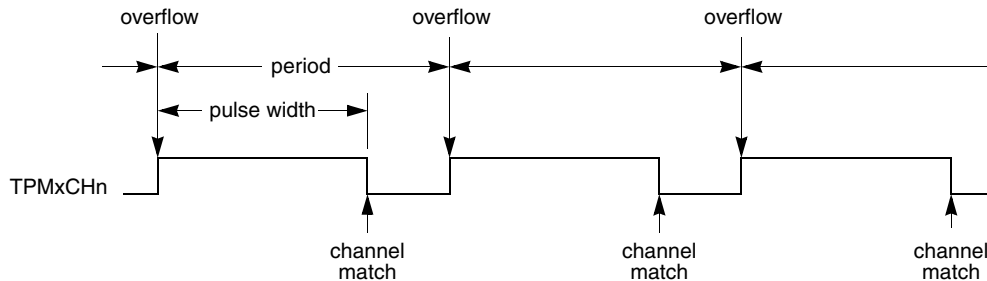


Figure 25-14. EPWM period and pulse width (ELSnA=0)

When the channel value register is set to 0x0000, the duty cycle is 0%. A 100% duty cycle is achieved by setting the timer-channel register (TPMxCnVH:TPMxCnVL) to a value greater than the modulus setting. This implies that the modulus setting must be less than 0xFFFF in order to get 100% duty cycle.

The timer channel registers are buffered to ensure coherent 16-bit updates and to avoid unexpected PWM pulse widths. Writes to any of the registers TPMxCnVH and TPMxCnVL actually write to buffer registers. In edge-aligned PWM mode, the TPMxCnVH:TPMxCnVL registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from (TPMxMODH:TPMxMODL – 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFE to 0xFFFF.

#### 25.4.2.4 Center-Aligned PWM Mode

This type of PWM output uses the up/down counting mode of the timer counter (CPWMS=1). The channel match value in TPMxCnVH:TPMxCnVL determines the pulse width (duty cycle) of the PWM signal while the period is determined by the value in TPMxMODH:TPMxMODL. TPMxMODH:TPMxMODL must be kept in the range of 0x0001 to 0x7FFF because values outside this range can produce ambiguous results. ELSnA determines the polarity of the CPWM signal.

$$\text{pulse width} = 2 \times (\text{TPMxCnVH:TPMxCnVL})$$

$$\text{period} = 2 \times (\text{TPMxMODH:TPMxMODL}); \text{TPMxMODH:TPMxMODL} = 0x0001\text{--}0x7FFF$$

If TPMxCnVH:TPMxCnVL is zero or negative (bit 15 set), the duty cycle is 0%. If TPMxCnVH:TPMxCnVL is a positive value (bit 15 clear) and is greater than the non-zero modulus setting, the duty cycle is 100% because the channel match never occurs. This implies the usable range of periods set by the modulus register is 0x0001 through 0x7FFE (0x7FFF if you do not need to generate 100% duty cycle). This is not a significant limitation. The resulting period is much longer than required for normal applications.

All zeros in TPMxMODH:TPMxMODL is a special case that must not be used with center-aligned PWM mode. When CPWMS is cleared, this case corresponds to the counter running free from 0x0000 through 0xFFFF. When CPWMS is set, the counter needs a valid match to the modulus register somewhere other than at 0x0000 in order to change directions from up-counting to down-counting.

The channel match value in the TPM channel registers (times two) determines the pulse width (duty cycle) of the CPWM signal (Figure 25-15). If ELSnA is cleared, a channel match occurring while counting up clears the CPWM output signal and a channel match occurring while counting down sets the output. The counter counts up until it reaches the modulo setting in TPMxMODH:TPMxMODL, then counts down until it reaches zero. This sets the period equal to two times TPMxMODH:TPMxMODL.

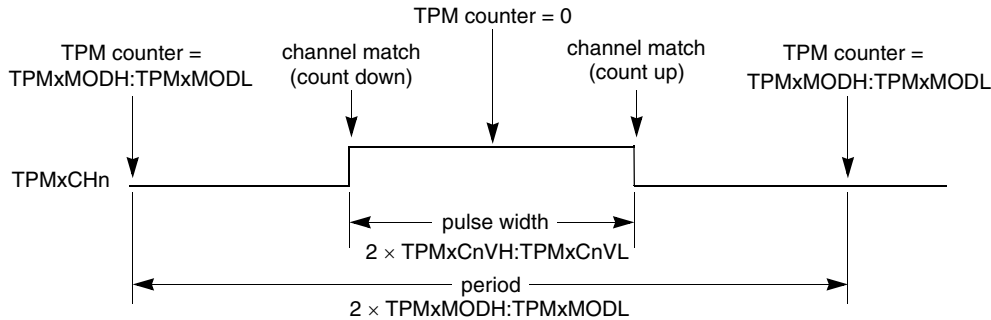


Figure 25-15. CPWM period and pulse width (ELSnA=0)

Center-aligned PWM outputs typically produce less noise than edge-aligned PWMs because fewer I/O pin transitions are lined up at the same system clock edge. This type of PWM is also required for some types of motor drives.

Input capture, output compare, and edge-aligned PWM functions do not make sense when the counter is operating in up/down counting mode so this implies that all active channels within a TPM must be used in CPWM mode when CPWMS is set.

The timer channel registers are buffered to ensure coherent 16-bit updates and to avoid unexpected PWM pulse widths. Writes to any of the registers TPMxCnVH and TPMxCnVL actually write to buffer registers. In center-aligned PWM mode, the TPMxCnVH:TPMxCnVL registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written, and the TPM counter changes from  $(\text{TPMxMODH:TPMxMODL} - 1)$  to  $(\text{TPMxMODH:TPMxMODL})$ . If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFE to 0xFFFF.

When TPMxCNTH:TPMxCNTL equals TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

## 25.5 Reset Overview

### 25.5.1 General

The TPM is reset whenever any MCU reset occurs.

## 25.5.2 Description of Reset Operation

Reset clears TPMxSC that disables TPM counter clock and overflow interrupt (TOIE=0). CPWMS, MSnB, MSnA, ELSnB, and ELSnA are all cleared. This configures all TPM channels for input capture operation and the associated pins are not controlled by TPM.

## 25.6 Interrupts

### 25.6.1 General

The TPM generates an optional interrupt for the main counter overflow and an interrupt for each channel. The meaning of channel interrupts depends on each channel's mode of operation. If the channel is configured for input capture, the interrupt flag is set each time the selected input capture edge is recognized. If the channel is configured for output compare or PWM modes, the interrupt flag is set each time the main timer counter matches the value in the 16-bit channel value register.

All TPM interrupts are listed in [Table 25-7](#).

**Table 25-7. Interrupt Summary**

Interrupt	Local Enable	Source	Description
TOF	TOIE	Counter overflow	Set each time the TPM counter reaches its terminal count (at transition to its next count value)
CHnF	CHnIE	Channel event	An input capture event or channel match took place on channel n

The TPM module provides high-true interrupt signals.

### 25.6.2 Description of Interrupt Operation

For each interrupt source in the TPM, a flag bit is set upon recognition of the interrupt condition such as timer overflow, channel input capture, or output compare events. This flag is read (polled) by software to determine that the action has occurred, or an associated enable bit (TOIE or CHnIE) can be set to enable the interrupt generation. While the interrupt enable bit is set, the interrupt is generated whenever the associated interrupt flag is set. Software must perform a sequence of steps to clear the interrupt flag before returning from the interrupt-service routine.

TPM interrupt flags are cleared by a two-step process including a read of the flag bit while it is set followed by a write of zero to the bit. If a new event is detected between these two steps, the sequence is reset and the interrupt flag remains set after the second step to avoid the possibility of missing the new event.

#### 25.6.2.1 Timer Overflow Interrupt (TOF) Description

The meaning and details of operation for TOF interrupts varies slightly depending upon the mode of operation of the TPM system (general purpose timing functions versus center-aligned PWM operation). The flag is cleared by the two step sequence described above.

### 25.6.2.1.1 Normal Case

When CPWMS is cleared, TOF is set when the timer counter changes from the terminal count (the value in the modulo register) to 0x0000. If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0x0000.

### 25.6.2.1.2 Center-Aligned PWM Case

When CPWMS is set, TOF is set when the timer counter changes direction from up-counting to down-counting at the end of the terminal count (the value in the modulo register).

## 25.6.2.2 Channel Event Interrupt Description

The meaning of channel interrupts depends on the channel's current mode (input capture, output compare, edge-aligned PWM, or center-aligned PWM).

### 25.6.2.2.1 Input Capture Events

When a channel is configured as an input capture channel, the ELSnB:ELSnA bits select if channel pin is not controlled by TPM, rising edges, falling edges, or any edge as the edge that triggers an input capture event. When the selected edge is detected, the interrupt flag is set. The flag is cleared by the two-step sequence described in [Section 25.6.2, "Description of Interrupt Operation."](#)

### 25.6.2.2.2 Output Compare Events

When a channel is configured as an output compare channel, the interrupt flag is set each time the main timer counter matches the 16-bit value in the channel value register. The flag is cleared by the two-step sequence described in [Section 25.6.2, "Description of Interrupt Operation."](#)

### 25.6.2.2.3 PWM End-of-Duty-Cycle Events

When the channel is configured for edge-aligned PWM, the channel flag is set when the timer counter matches the channel value register that marks the end of the active duty cycle period. When the channel is configured for center-aligned PWM, the timer count matches the channel value register twice during each PWM cycle. In this CPWM case, the channel flag is set at the start and at the end of the active duty cycle period when the timer counter matches the channel value register. The flag is cleared by the two-step sequence described in [Section 25.6.2, "Description of Interrupt Operation."](#)

---

## Chapter 26

# Version 1 ColdFire Debug (CF1\_DEBUG)

### 26.1 Introduction

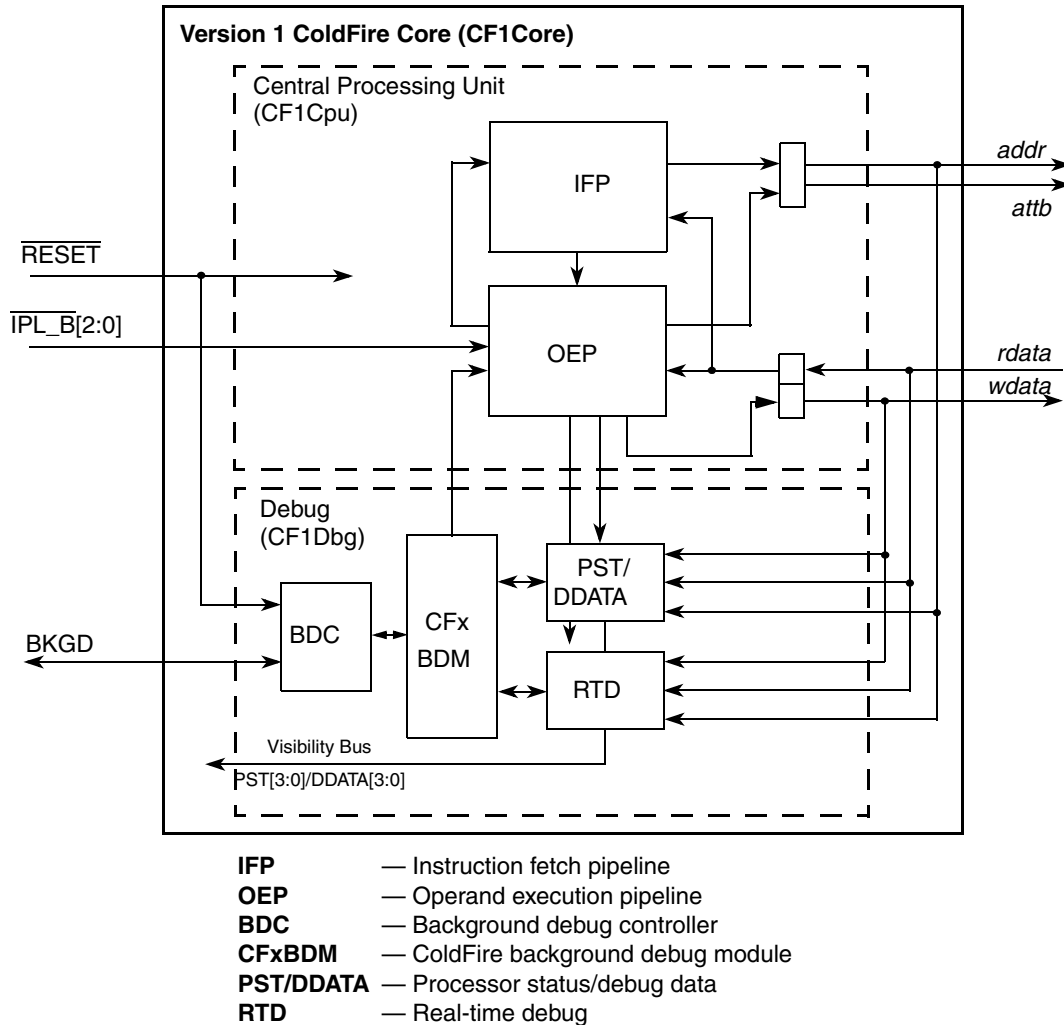
This chapter describes the capabilities defined by the Version 1 ColdFire debug architecture. The Version 1 ColdFire core supports BDM functionality using the HCS08's single-pin interface. The traditional 3-pin full-duplex ColdFire BDM serial communication protocol based on 17-bit data packets is replaced with the HCS08 debug protocol where all communication is based on an 8-bit data packet using a single package pin (BKGD).

An on-chip trace buffer allows a stream of compressed processor execution status packets to be recorded for subsequent retrieval to provide program (and partial data) trace capabilities.

For this device, the Version 1 ColdFire debug architecture also supports optional visibility bus (VBus) signals. The VBus signals support real-time program trace capabilities of arbitrarily-long instruction streams.

The following sections in this chapter provide details on the BKGD pin, the background debug serial interface controller (BDC), a standard 6-pin BDM connector, the BDM command set as well as real-time debug and trace capabilities. The V1 definition supports revision B+ (DEBUG\_B+) of the ColdFire debug architecture.

A simplified block diagram of the V1 core including the processor and debug module is shown in [Figure 26-1](#).



**Figure 26-1. Simplified Version 1 ColdFire Core Block Diagram**

## 26.1.1 Overview

Debug support is divided into three areas:

- **Background debug mode (BDM)**—Provides low-level debugging in the ColdFire processor core. In BDM, the processor core is halted and a variety of commands can be sent to the processor to access memory, registers, and peripherals. The external emulator uses a one-pin serial communication protocol. See [Section 26.4.1, “Background Debug Mode \(BDM\)”](#).
- **Real-time debug support**—Use of the full BDM command set requires the processor to be halted, which many real-time embedded applications cannot support. The core includes a variety of internal breakpoint registers which can be configured to trigger and generate a special interrupt. The resulting debug interrupt lets 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. The external development system can then access the saved data, because the hardware supports



concurrent operation of the processor and BDM-initiated memory commands. In addition, the option is provided to allow interrupts to occur. See [Section 26.4.2, “Real-Time Debug Support”](#).

- Program trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The V1 solution implements a trace buffer that records processor execution status and data, which can be subsequently accessed by the external emulator system to provide program (and optional partial data) trace information. See [Section 26.4.4, “Trace Support With the Visibility Bus Disabled \(CSR\[VBD\] = 1\)”](#).
- Additionally, this device includes the VBus interface signals which support real-time program trace by outputting the processor execution status and debug data to an external emulator system. See [Section 26.4.3, “Real-Time Trace Support with the Visibility Bus Enabled \(CSR\[VBD\] = 0\)”](#).

There are two fields in debug registers which provide revision information: the hardware revision level in CSR and the 1-pin debug hardware revision level in CSR2. [Table 26-1](#) summarizes the various debug revisions.

**Table 26-1. Debug Revision Summary**

Revision	CSR[HRL]	CSR2[D1HRL]	Enhancements
A	0000	N/A	Initial ColdFire debug definition
B	0001	N/A	BDM command execution does not affect hardware breakpoint logic Added BDM address attribute register (BAAR) 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	N/A	Added 3 PC breakpoint registers PBR1–3
CF1_B+	1001	0001	Converted to HCS08 1-pin BDM serial interface Added PST compression and on-chip PST/DDATA buffer for program trace
CF1_B+ with VBus	1001	1001	CF1 debug plus visibility bus support

## 26.1.2 Features

The Version 1 ColdFire debug definition supports the following features:

- Classic ColdFire DEBUG\_B+ functionality mapped into the single-pin BDM interface
- Real time debug support, with 6 hardware breakpoints (4 PC, 1 address pair and 1 data) that can be configured into a 1- or 2-level trigger with a programmable response (processor halt or interrupt)
- Capture of compressed processor status and debug data into on-chip trace buffer provides program (and optional slave bus data) trace capabilities
- On-chip trace buffer provides programmable start/stop recording conditions plus support for obtrusive or PC-profiling modes
- Debug resources are accessible via single-pin BDM interface or the privileged WDEBUG instruction from the core
- Support for real-time program (and optional partial data) trace using the visibility bus

### 26.1.3 Modes of Operations

V1 ColdFire devices typically implement a number of modes of operation, including run, wait, and stop modes. Additionally, the operation of the core's debug module is highly dependent on a number of chip configurations which determine its operating state.

When operating in secure mode, as defined by a 2-bit field in the flash memory examined at reset, BDM access to debug resources is extremely restricted. It is possible to tell that the device has been secured, and to clear security, which involves mass erasing the on-chip flash memory. No other debug access is allowed. Secure mode can be used in conjunction with each of the wait and stop low-power modes.

If the BDM interface is not enabled, access to the debug resources is limited in the same manner as a secure device.

If the device is not secure and the BDM interface is enabled (XCSR[ENBDM] is set), the device is operating in debug mode and additional resources are available via the BDM interface. In this mode, the status of the processor (running, stopped, or halted) determines which BDM commands may be used.

Debug mode functions are managed through the background debug controller (BDC) in the Version 1 ColdFire core. The BDC provides the means for analyzing MCU operation during software development.

BDM commands can be classified into three types as shown in [Table 26-2](#).

**Table 26-2. BDM Command Types**

Command Type	Flash Secure?	BDM?	Core Status	Command Set
Always-available	Secure or Unsecure	Enabled or Disabled	—	<ul style="list-style-type: none"> <li>Read/write access to XCSR[31–24], CSR2[31–24], CSR3[31–24]</li> </ul>
Non-intrusive	Unsecure	Enabled	Run, Halt	<ul style="list-style-type: none"> <li>Memory access</li> <li>Memory access with status</li> <li>Debug register access</li> <li>BACKGROUND</li> </ul>
Active background	Unsecure	Enabled	Halt	<ul style="list-style-type: none"> <li>Read or write CPU registers (also available in stop mode)</li> <li>Single-step the application</li> <li>Exit halt mode to return to the application program (GO)</li> </ul>

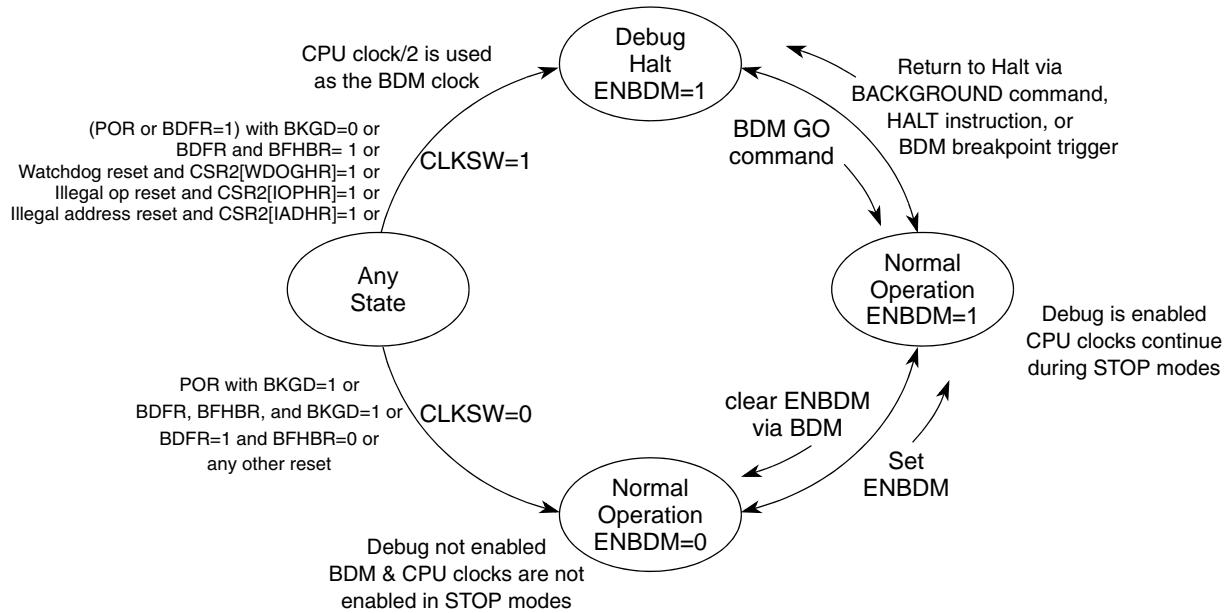
For more information on these three BDM command classifications, see [Section 26.4.1.5, “BDM Command Set Summary.”](#)

The core's halt mode is entered in a number of ways:

- The BKGD pin is low during POR
- The BKGD pin is low immediately after a BDM-initiated force reset (see CSR2[BDFR] in [Section 26.3.3, “Configuration/Status Register 2 \(CSR2\),”](#) for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- A watchdog reset occurs and CSR2[WDOGHR] is set
- An illegal operand reset occurs and CSR2[IOPHR] is set
- An illegal address reset occurs and CSR2[IADHR] is set

- A BACKGROUND command is received through the BKGD pin. If necessary, this wakes the device from STOP/WAIT modes.
- A properly-enabled (XCSR[ENBDM] is set) HALT instruction is executed
- Encountering a BDM breakpoint and the trigger response is programmed to generate a halt
- Reaching a PSTB trace buffer full condition when operating in an obtrusive recording mode (CSR2[PSTBRM] is set to 01 or 11)

While in halt mode, the core waits for serial background commands rather than executing instructions from the application program.



**Figure 26-2. Debug Modes State Transition Diagram**

Figure 26-2 contains a simplified view of the V1 ColdFire debug mode states. The XCSR[CLKSW] bit controls the BDC clock source. When CLKS is set, the BDC serial clock frequency is half the CPU clock. When CLKS is cleared, the BDC serial clock is supplied from an alternate clock source.

The ENBDM bit determines if the device can be placed in halt mode, if the core and BDC serial clocks continue to run in STOP modes, and if the regulator can be placed into standby mode. Again, if booting to halt mode, XCSR[ENBDM, CLKS] are automatically set.

If ENBDM is cleared, the ColdFire core treats the HALT instruction as an illegal instruction and generates a reset (if CPUCR[IRD] is cleared) or an exception (if CPUCR[IRD] is set) if execution is attempted.

If XCSR[ENBDM] is set, the device can be restarted from STOP/WAIT via the BDM interface.

## 26.2 External Signal Descriptions

Table 26-3 describes the debug module's 1-pin external signal (BKGD) and the signals associated with the visibility bus. A standard 6-pin debug connector is shown in Section 26.4.5, "Freescale-Recommended BDM Pinout".

**Table 26-3. Debug Module Signals**

Signal	Description
Background Debug (BKGD)	Single-wire background debug interface pin. The primary function of this pin is for bidirectional serial communication of background debug mode commands and data. During reset, this pin selects between starting in active background (halt) mode or starting the application program. This pin also requests a timed sync response pulse to allow a host development tool to determine the correct clock frequency for background debug serial communications.
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.
Processor Status Clock 0 & 1 (PSTCLK $n$ )	Two separate half-speed output signals to externally derive the processor status clock. The PSTCLK0 and PSTCLK1 signals operate at half speed compared to the processor clock with PSTCLK1 delayed by half a processor clock period relative to PSTCLK0. Externally, these two clock signals should be exclusive-OR'd together and then complemented (an exclusive-NOR boolean function) to generate a derived processor status clock suitable for sampling the PST[3:0] and DDATA[3:0] outputs. Specifically, the rising-edge of the derived processor status clock defines the sample point for capturing the PST/DDATA outputs. The state of CSR[VBD] controls the package pin muxing: <ul style="list-style-type: none"> <li>• If CSR[VBD] = 0, the VBus is enabled, the PSTCLK<math>n</math>, PST[3:0] and DDATA[3:0] outputs are functional and enabled in the appropriate GPIO logic and <math>\overline{\text{BKPT}}</math> is routed into the debug module.</li> <li>• If CSR[VBD] = 1, the VBus is disabled, the PSTCLK<math>n</math>, PST and DDATA outputs are all quiescent and the appropriate GPIO logic is disabled. The <math>\overline{\text{BKPT}}</math> pin is logically disconnected from the debug module.</li> </ul> See Figure 26-3 for details on the PSTCLK $n$ behavior and the derived PSTCLK and Table 26-26 for definition of the 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 which are displayed on DDATA. These signals are updated each processor cycle.
Processor Status (PST[3:0])	These output signals report the processor status. Table 26-26 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.

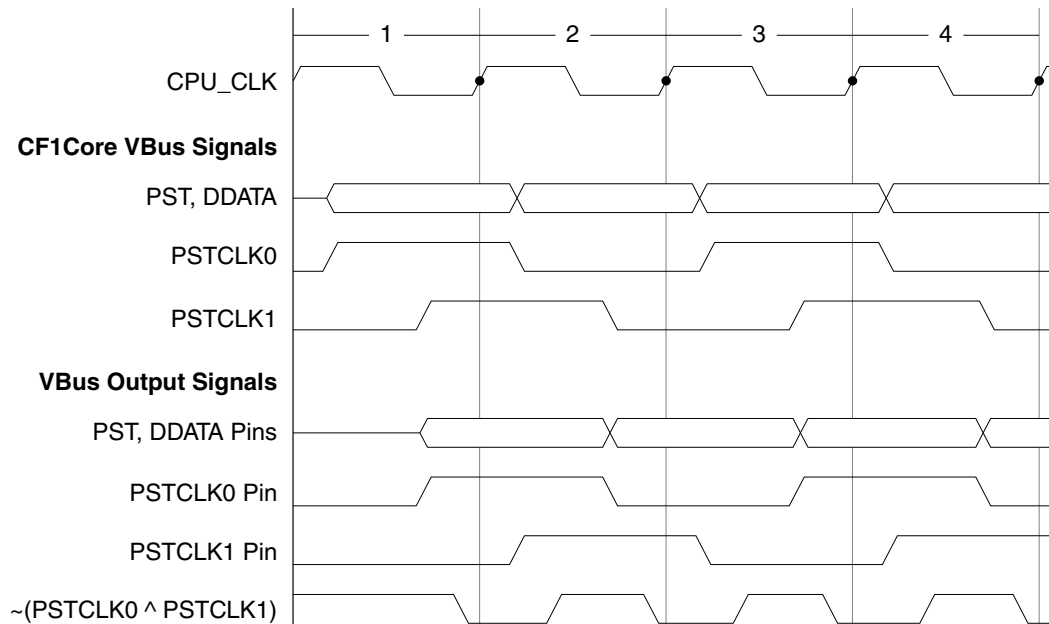


Figure 26-3. Deriving PSTCLK

## 26.3 Memory Map/Register Definition

In addition to the BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains a number of registers. Most of these registers (all except the PST/DDATA trace buffer) are also accessible (write-only) from the processor's supervisor programming model by executing the WDEBUG instruction. Thus, the breakpoint hardware in the debug module can be read (certain registers) or written by the external development system using the serial debug interface or written by the operating system running on the processor core. Software is responsible for guaranteeing that accesses to these resources are serialized and logically consistent. The hardware provides a locking mechanism in the CSR to allow the external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued during the processor's execution of the WDEBUG instruction to configure debug module registers or the resulting behavior is undefined.

These registers, shown in [Table 26-4](#), are treated as 32-bit quantities regardless of the number of implemented bits and unimplemented bits are reserved and must be cleared. These registers are also accessed through the BDM port by the commands, WRITE\_DREG and READ\_DREG, described in [Section 26.4.1.5, "BDM Command Set Summary."](#) These commands contain a 5-bit field, DRc, that specifies the register, as shown in [Table 26-4](#).

Table 26-4. Debug Module Memory Map

DRc	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W (BDM), W (CPU)	0x0090_0000	<a href="#">26.3.1/26-9</a>
0x01	Extended Configuration/Status Register (XCSR)	32	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">26.3.2/26-12</a>
0x02	Configuration/Status Register 2 (CSR2)	32	R/W <sup>1</sup> (BDM), W (CPU)	See Section	<a href="#">26.3.3/26-15</a>
0x03	Configuration/Status Register 3 (CSR3)	32 <sup>2</sup>	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">26.3.4/26-18</a>
0x05	BDM address attribute register (BAAR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">26.3.5/26-19</a>
0x06	Address attribute trigger register (AATR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">26.3.6/26-20</a>
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	<a href="#">26.3.7/26-21</a>
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined, Unaffected	<a href="#">26.3.8/26-24</a>
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined, Unaffected	<a href="#">26.3.8/26-24</a>
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined, Unaffected	<a href="#">26.3.9/26-26</a>
0x0D	Address breakpoint low register (ABLR)	32	W	0x0000_0000	<a href="#">26.3.9/26-26</a>
0x0E	Data breakpoint register (DBR)	32	W	0x0000_0000	<a href="#">26.3.10/26-27</a>
0x0F	Data breakpoint mask register (DBMR)	32	W	0x0000_0000	<a href="#">26.3.10/26-27</a>
0x18	PC breakpoint register 1 (PBR1)	32	W	PBR1[0] = 0	<a href="#">26.3.8/26-24</a>
0x1A	PC breakpoint register 2 (PBR2)	32	W	PBR2[0] = 0	<a href="#">26.3.8/26-24</a>
0x1B	PC breakpoint register 3 (PBR3)	32	W	PBR3[0] = 0	<a href="#">26.3.8/26-24</a>
—	PST Trace Buffer <i>n</i> (PSTB <sub><i>n</i></sub> ); <i>n</i> = 0–11 (0xB)	32	R (BDM) <sup>3</sup>	Undefined, Unaffected	<a href="#">26.4.1.5.12/26-48</a>

<sup>1</sup> The most significant bytes of the XCSR, CSR2, and CSR3 registers support special control functions and are writeable via BDM using the WRITE\_XCSR\_BYTE, WRITE\_CSR2\_BYTE, and WRITE\_CSR3\_BYTE commands. They can be read from BDM using the READ\_XCSR\_BYTE, READ\_CSR2\_BYTE, and READ\_CSR3\_BYTE commands. These 3 registers, along with the CSR, can also be referenced as 32-bit quantities using the BDM READ\_DREG and WRITE\_DREG commands, but the WRITE\_DREG command only writes bits 23–0 of these three registers.

<sup>2</sup> Each debug register is accessed as a 32-bit value; undefined fields are reserved and must be cleared.

<sup>3</sup> The contents of the PST trace buffer is only read from BDM (32 bits per access) using READ\_PSTB commands.

**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 WRITE\_DREG command. In addition, the four configuration/status registers (CSR, XCSR, CSR2, CSR3) can be read through the BDM port using the READ\_DREG 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. The triggers can be configured to halt the processor or generate a debug interrupt exception. Additionally, these same breakpoint registers can be used to specify start/stop conditions for recording in the PST trace buffer.

The core includes four PC breakpoint triggers and a set of operand address breakpoint triggers with two independent address registers (to allow specification of a range) and an optional data breakpoint with masking capabilities. Core breakpoint triggers are accessible through the serial BDM interface or written through the supervisor programming model using the WDEBUG instruction.

**26.3.1 Configuration/Status Register (CSR)**

CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is accessible from the programming model using the WDEBUG instruction and through the BDM port using the READ\_DREG and WRITE\_DREG 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	VBD	IPW
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	TRC	0	DDC	UHE	BTB		0	NPL	IPI	SSM	0	0	FID	DDH	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 26-4. Configuration/Status Register (CSR)**

Table 26-5. 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, 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. The PSTB value that follows the PSTB entry of 0x1B is $0x20 + (2 \times \text{BSTAT})$ and the visibility bus PST value is $2 \times \text{BSTAT}$ . 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
27 FOF	Fault-on-fault. Indicates a catastrophic halt occurred and forced entry into BDM. FOF is cleared by reset or when CSR is read (from the BDM port only).
26 TRG	Hardware breakpoint trigger. Indicates 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) clears TRG.
25 HALT	Processor halt. Indicates the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears HALT.
24 BKPT	Breakpoint assert. Indicates when either: <ul style="list-style-type: none"> <li>• The <math>\overline{\text{BKPT}}</math> input was asserted,</li> <li>• BDM BACKGROUND command received, or</li> <li>• The PSTB halt on full condition, CSR2[PSTBH], sets.</li> </ul> This forces the processor into a BDM halt. Reset, the debug GO command, or reading CSR (from the BDM port only) clears BKPT.
23–20 HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator can 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+ (The value used for this device) 1011 Revision D+
19	Reserved, must be cleared.
18 BKD	Breakpoint disable. Disables the normal $\overline{\text{BKPT}}$ input signal and BACKGROUND command functionality, and allows the assertion of this pin (or execution of the BACKGROUND command) to generate a debug interrupt. 0 Normal operation 1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ or the receipt of a BDM BACKGROUND command 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 VBD	Visibility bus disable. Disables the Vbus in the debug module and the GPIO logic paths. 0 VBus enabled. PSTCLK <sub>n</sub> , PST[3:0], DDATA[3:0] and $\overline{\text{BKPT}}$ pins are enabled and operational. 1 VBus disabled. PSTCLK <sub>n</sub> , PST[3:0], and DDATA[3:0] pins are quiescent and disabled in the GPIO logic. $\overline{\text{BKPT}}$ is logically disconnected from the debug module and disabled in the GPIO logic.
16 IPW	Inhibit processor writes. Inhibits processor-initiated writes to the debug module's programming model registers. IPW can be modified only by commands from the BDM interface.
15	Reserved, must be cleared.



Table 26-5. CSR Field Descriptions (continued)

Field	Description
14 TRC	Force emulation mode on trace exception. 0 Processor enters supervisor mode. 1 Processor enters emulator mode when a trace exception occurs.
13	Reserved, must be cleared.
12–11 DDC	Debug data control. Controls peripheral bus operand data capture for DDATA, which displays the number of bytes defined by the operand reference size (a marker) 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 <a href="#">Table 26-27</a> . A non-zero value enables partial data trace capabilities. 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. The core must be operating with XCSR[ENBDM] set to execute any HALT instruction, else the instruction is treated as an illegal opcode. 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. See <a href="#">Section 26.4.4.1</a> , “ <a href="#">Begin Execution of Taken Branch (PST = 0x05)</a> .” 00 No target address capture 01 Lower 2 bytes of the target address 1x Lower 3 bytes of the target address
7	Reserved, must be cleared.
6 NPL	Non-pipelined mode. Determines if the core operates in pipelined mode. 0 Pipelined mode 1 Non-pipelined mode. The processor effectively executes one instruction at a time with no overlap. This typically adds five cycles to the execution time of each instruction. Given an average execution latency of ~2 cycles per instruction, throughput in non-pipeline mode would be ~7 cycles per instruction, approximately 25% - 33% of pipelined performance.  Regardless of the NPL state, a triggered PC breakpoint is always reported before the triggering instruction executes. In normal pipeline operation, the occurrence of an address and/or data breakpoint trigger is imprecise. In non-pipeline mode, these triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise.
5 IPI	Ignore pending interrupts when in single-step mode. 0 Core services any pending interrupt requests signalled while in single-step mode. 1 Core ignores any pending interrupt requests signalled while in single-step mode.
4 SSM	Single-step mode enable. 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.
3–2	Reserved, must be cleared.

Table 26-5. CSR Field Descriptions (continued)

Field	Description
1 FID	Force <i>ipg_debug</i> . The core generates this output to the device, signaling it is in debug mode. 0 Do not force the assertion of <i>ipg_debug</i> 1 Force the assertion of <i>ipg_debug</i>
0 DDH	Disable <i>ipg_debug</i> due to a halt condition. The core generates an output to the other modules in the device, signaling it is in debug mode. By default, this output signal is asserted when the core halts. 0 Assert <i>ipg_debug</i> if the core is halted 1 Negate <i>ipg_debug</i> due to the core being halted

### 26.3.2 Extended Configuration/Status Register (XCSR)

The 32-bit XCSR is partitioned into two sections: the upper byte contains status and command bits always accessible to the BDM interface, even if debug mode is disabled. This status byte is also known as XCSR\_SB. The lower 24 bits contain fields related to the generation of automatic SYNC\_PC commands, which can be used to periodically capture and display the current program counter (PC) in the PST trace buffer (if properly configured).

There are multiple ways to reference the XCSR. They are summarized in [Table 26-6](#).

Table 26-6. XCSR Reference Summary

Method	Reference Details
READ_XCSR_BYTE	Reads XCSR[31–24] from the BDM interface. Available in all modes.
WRITE_XCSR_BYTE	Writes XCSR[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads XCSR[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes XCSR[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG instruction	Writes XCSR[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x01 (XCSR)

Access: Supervisor write-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CPU HALT	CPU STOP	CSTAT		CLK SW	SEC ERASE	EN BDM	0	0	0	0	0	0	0	0	0
W			ESEQC													
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	APCSC		APC ENB
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. Extended Configuration/Status Register (XCSR)

Table 26-7. XCSR Field Descriptions

Field	Description												
31 CPUHALT	<p>Indicates that the CPU is in the halt state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown below.</p> <table border="1"> <thead> <tr> <th>XCSR [CPUHALT]</th> <th>XCSR [CPUSTOP]</th> <th>CPU State</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Running</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stopped</td> </tr> <tr> <td>1</td> <td>0</td> <td>Halted</td> </tr> </tbody> </table>	XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State	0	0	Running	0	1	Stopped	1	0	Halted
XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State											
0	0	Running											
0	1	Stopped											
1	0	Halted											
30 CPUSTOP	<p>Indicates that the CPU is in the stop state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown in the CPUHALT bit description.</p>												
29–27 CSTAT (R) ESEQC (W)	<p>During reads, indicates the BDM command status.</p> <p>000 Command done, no errors 001 Command done, data invalid 01x Command done, illegal 1xx Command busy, overrun</p> <p>If an overrun is detected (CSTAT = 1xx), the following sequence is suggested to clear the source of the error:</p> <ol style="list-style-type: none"> <li>1. Issue a SYNC command to reset the BDC channel.</li> <li>2. The host issues a BDM NOP command.</li> <li>3. The host checks the channel status using a READ_XCSR_BYTE command.</li> <li>4. If XCSR[CSTAT] = 000  then status is okay; proceed  else  Halt the CPU with a BDM BACKGROUND command  Repeat steps 1,2,3  If XCSR[CSTAT] ≠ 000, then reset device</li> </ol> <p>During writes, the ESEQC field is used for the erase sequence control during flash programming. ERASE must also be set for this bit to have an effect.</p> <p>000 User mass erase Else Reserved</p> <p><b>Note:</b> See the Memory chapter for a detailed description of the algorithm for clearing security.</p>												
26 CLKSW	<p>Select source for serial BDC communication clock.</p> <p>0 Alternate, asynchronous BDC clock, typically 10 MHz 1 Synchronous bus clock (CPU clock divided by 2)</p> <p>The initial state of the XCSR[CLKSW] bit is loaded by the hardware in response to certain reset events and the state of the BKGD pin as described in <a href="#">Figure 26-2</a>.</p>												

Table 26-7. XCSR Field Descriptions (continued)

Field	Description
25 SEC (R) ERASE (W)	<p>The read value of this bit typically defines the status of the flash security field.</p> <p>0 Flash security is disabled            1 Flash security is enabled</p> <p>In addition, the SEC bit is context-sensitive during reads. After a mass-erase sequence has been initiated by BDM, it acts as a flash busy flag. When the erase operation is complete and the bit is cleared, it returns to reflect the status of the chip security.</p> <p>0 Flash is not busy performing a BDM mass-erase sequence            1 Flash is busy performing a BDM mass-erase sequence</p> <p>During writes, this bit qualifies XCSR[ESEQC] for the write modes shown in the ESEQC field description.</p> <p>0 Do not perform a mass-erase of the flash.            1 Perform a mass-erase of the flash, using the sequence specified in the XCSR[ESEQC] field.</p>
24 ENBDM	<p>Enable BDM.</p> <p>0 BDM mode is disabled            1 Active background mode is enabled (assuming the flash is not secure)</p>
23–3	<p>Reserved for future use by the debug module, must be cleared.</p>

**Table 26-7. XCSR Field Descriptions (continued)**

Field	Description																																				
2–1 APCSC	<p>Automatic PC synchronization control. Determines the periodic interval of PC address captures, if XCSR[APCENB] is set. When the selected interval is reached, a SYNC_PC command is sent to the ColdFire CPU. For more information on the SYNC_PC operation, see the APCENB description.</p> <p>The chosen frequency depends on CSR2[APCDIV16] as shown in the equation and table below:</p> $\text{PC address capture period} = \frac{2^{(\text{APCSC} + 1)} \times 1024}{16^{\text{APCDIV16}}}$ <p style="text-align: right;"><b>Eqn. 26-1</b></p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>XCSR [APCENB]</th> <th>CSR2 [APCDIV16]</th> <th>XCSR [APCSC]</th> <th>SYNC_PC Interval</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>00</td> <td>2048 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>01</td> <td>4096 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>10</td> <td>8192 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>11</td> <td>16384 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>00</td> <td>128 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>01</td> <td>256 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>10</td> <td>512 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>11</td> <td>1024 cycles</td> </tr> </tbody> </table>	XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval	1	0	00	2048 cycles	1	0	01	4096 cycles	1	0	10	8192 cycles	1	0	11	16384 cycles	1	1	00	128 cycles	1	1	01	256 cycles	1	1	10	512 cycles	1	1	11	1024 cycles
XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval																																		
1	0	00	2048 cycles																																		
1	0	01	4096 cycles																																		
1	0	10	8192 cycles																																		
1	0	11	16384 cycles																																		
1	1	00	128 cycles																																		
1	1	01	256 cycles																																		
1	1	10	512 cycles																																		
1	1	11	1024 cycles																																		
0 APCENB	<p>Automatic PC synchronization enable. Enables the periodic output of the PC which can be used for PST/DDATA trace synchronization and code profiling.</p> <p>As described in XCSR[APCSC], when the enabled periodic timer expires, a SYNC_PC command is sent to the ColdFire CPU which generates a forced instruction fetch of the next instruction. The PST/DDATA module captures the target address as defined by CSR[9] (two bytes if CSR[9] is cleared, three bytes if CSR[9] is set). This produces a PST sequence of the PST marker indicating a 2- or 3-byte address, followed by the captured instruction address.</p> <p>0 Automatic PC synchronization disabled 1 Automatic PC synchronization enabled</p>																																				

### 26.3.3 Configuration/Status Register 2 (CSR2)

The 32-bit CSR2 is partitioned into two sections. The upper byte contains status and configuration bits always accessible to the BDM interface, even if debug mode is disabled. The lower 24 bits contain fields related to the configuration of the PST trace buffer (PSTB).

There are multiple ways to reference CSR2. They are summarized in [Table 26-8](#).

**Table 26-8. CSR2 Reference Summary**

Method	Reference Details
READ_CSR2_BYTE	Reads CSR2[31–24] from the BDM interface. Available in all modes.
WRITE_CSR2_BYTE	Writes CSR2[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads CSR2[31–0] from the BDM interface. Classified as a non-intrusive BDM command.

Table 26-8. CSR2 Reference Summary (continued)

Method	Reference Details
WRITE_DREG	Writes CSR2[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	Writes CSR2[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x02 (CSR2)

Access: Supervisor read-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PSTBP	0	WDOG	IOP	IAD	0	BFHBR	0	PSTBH	PSTBST	0	D1HRL				
W			HR	HR	HR			BDFR								
Power-on Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Other Reset	0	0	u	u	u	0	u	0	0	0	0	0	1	0	0	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PSTBWA								0	APC	0	PSTBRM		PSTBSS		
W									PSTBR	DIV16						
Reset	Unaffected and Undefined								0	0	0	0	0	0	0	0

Figure 26-6. Configuration/Status Register 2 (CSR2)

Table 26-9. CSR2 Field Descriptions

Field	Description
31 PSTBP	PST buffer stop. Signals if a PST buffer stop condition has been reached. 0 A PST trace buffer stop condition has not been reached 1 A PST trace buffer stop condition has been reached
30	Reserved, must be cleared.
29 WDOGHR	Watchdog halt after reset. Determines operation of the device after a watchdog reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After a watchdog reset, the device immediately enters normal operation mode. 1 A watchdog reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.
28 IOPHR	Illegal operation halt after reset. Determines operation of the device after an illegal operation reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After the device has an illegal operation reset, the device immediately enters normal operation mode. 1 An illegal operation reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.
27 IADHR	Illegal address halt after reset. Determines operation of the device after an illegal address reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After the device has an illegal address reset, the device immediately enters normal operation mode. 1 An illegal address reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). <b>Note:</b> This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.
26	Reserved, must be cleared.

Table 26-9. CSR2 Field Descriptions (continued)

Field	Description						
25 BFHBR	<p>BDM force halt on BDM reset. Determines operation of the device after a BDM reset. This bit is cleared after a power-on reset and is unaffected by any other reset.</p> <p>0 The device enters normal operation mode following a BDM reset. 1 The device enters in halt mode following a BDM reset, as if the BKGD pin was held low after a power-on-reset or standard BDM-initiated reset.</p> <p><b>Note:</b> This bit can only change state if XCSR[ENBDM] = 1 and the flash is unsecure.</p>						
24 BDFR	<p>Background debug force reset. Forces a BDM reset to the device. This bit always reads as 0 after the reset has been initiated.</p> <p>0 No reset initiated. 1 Force a BDM reset.</p>						
23 PSTBH	<p>PST trace buffer halt. Indicates if the processor is halted due to the PST trace buffer being full when recording in a obtrusive mode.</p> <p>0 PST trace buffer not full 1 CPU halted due to PST trace buffer being full in obtrusive mode</p>						
22–21 PSTBST	<p>PST trace buffer state. Indicates the current state of the PST trace buffer recording.</p> <p>00 PSTB disabled 01 PSTB enabled and waiting for the start condition 10 PSTB enabled, recording and waiting for the stop condition 11 PSTB enabled, completed recording after the stop condition was reached</p>						
20	Reserved, must be cleared.						
19–16 D1HRL	Debug 1-pin hardware revision level. Indicates the hardware revision level of the 1-pin debug module implemented in the ColdFire core. For this device, this field is 0x9.						
15–8 PSTBWA	<p>PST trace buffer write address. Indicates the current write address of the PST trace buffer. The most significant bit of this field is sticky; if set, it remains set until a PST/DDATA reset event occurs. As the ColdFire core inserts PST and DDATA packets into the trace buffer, this field is incremented. The value of the write address defines the next location in the PST trace buffer to be loaded. In other words, the contents of PSTB[PSTBWA-1] is the last valid entry in the trace buffer.</p> <p>The msb of this field can be used to determine if the entire PST trace buffer has been loaded with valid data.</p> <table border="1" data-bbox="518 1266 1224 1451"> <thead> <tr> <th>PSTBWA[7]</th> <th>PSTB Valid Data Locations (Oldest to Newest)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0, 1, ... PSTBWA-1</td> </tr> <tr> <td>1</td> <td>PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1</td> </tr> </tbody> </table> <p>The PSTBWA is unaffected when a buffer stop condition has been reached, the buffer is disabled, or a system reset occurs. This allows the contents of the PST trace buffer to be retrieved after these events to assist in debug.</p> <p><b>Note:</b> Since this device contains a 64-entry trace buffer, PSTBWA[6] is always zero.</p>	PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)	0	0, 1, ... PSTBWA-1	1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1
PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)						
0	0, 1, ... PSTBWA-1						
1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1						
7 PSTBR	<p>PST trace buffer reset. Generates a reset of the PST trace buffer logic, which clears PSTBWA and PSTBST. The same resources are reset when a disabled trace buffer becomes enabled and upon the receipt of a BDM GO command when operating in obtrusive trace mode. These reset events also clear any accumulation of PSTs. This bit always reads as a zero.</p> <p>0 Do not force a PST trace buffer reset 1 Force a PST trace buffer reset</p>						
6 APCDIV16	Automatic PC synchronization divide cycle counts by 16. This bit divides the cycle counts for automatic SYNC_PC command insertion by 16. See the APCSC and APCENB field descriptions.						

**Table 26-9. CSR2 Field Descriptions (continued)**

Field	Description																								
5	Reserved, must be cleared.																								
4–3 PSTBRM	<p>PST trace buffer recording mode. Defines the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field.</p> <p>00 Non-obtrusive, normal recording mode  01 Obtrusive, normal recording  10 Non-obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]).  11 Obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]).</p> <p>The terms obtrusive and non-obtrusive are defined as:</p> <ul style="list-style-type: none"> <li>• Non-obtrusive—The core is not halted. The PST trace buffer is overwritten unless a PSTB start/stop combination results in less than or equal to 64 PSTB captures.</li> <li>• Obtrusive—The core is halted when the PSTB trace buffer reaches its full level (full before overwriting). The PSTB trace buffer contents are available by the BDM PSTB_READ commands. The PSTB trace buffer write address resets and the CPU resumes upon a BDM GO command.</li> </ul>																								
2–0 PSTBSS	<p>PST trace buffer start/stop definition. Specifies the start and stop conditions for PST trace buffer recording. In certain cases, the start and stop conditions are defined by the breakpoint registers. The remaining breakpoint registers are available for trigger configurations.</p> <table border="1" data-bbox="516 865 1235 1306"> <thead> <tr> <th>PSTBSS</th> <th>Start Condition</th> <th>Stop Condition</th> </tr> </thead> <tbody> <tr> <td>000</td> <td colspan="2">Trace buffer disabled, no recording</td> </tr> <tr> <td>001</td> <td colspan="2">Unconditional recording</td> </tr> <tr> <td>010</td> <td rowspan="2">ABxR{&amp; DBR/DBMR}</td> <td>PBR0/PBMR</td> </tr> <tr> <td>011</td> <td>PBR1</td> </tr> <tr> <td>100</td> <td rowspan="2">PBR0/PBMR</td> <td>ABxR{&amp; DBR/DBMR}</td> </tr> <tr> <td>101</td> <td>PBR1</td> </tr> <tr> <td>110</td> <td rowspan="2">PBR1</td> <td>ABxR{&amp; DBR/DBMR}</td> </tr> <tr> <td>111</td> <td>PBR0/PBMR</td> </tr> </tbody> </table>	PSTBSS	Start Condition	Stop Condition	000	Trace buffer disabled, no recording		001	Unconditional recording		010	ABxR{& DBR/DBMR}	PBR0/PBMR	011	PBR1	100	PBR0/PBMR	ABxR{& DBR/DBMR}	101	PBR1	110	PBR1	ABxR{& DBR/DBMR}	111	PBR0/PBMR
PSTBSS	Start Condition	Stop Condition																							
000	Trace buffer disabled, no recording																								
001	Unconditional recording																								
010	ABxR{& DBR/DBMR}	PBR0/PBMR																							
011		PBR1																							
100	PBR0/PBMR	ABxR{& DBR/DBMR}																							
101		PBR1																							
110	PBR1	ABxR{& DBR/DBMR}																							
111		PBR0/PBMR																							

### 26.3.4 Configuration/Status Register 3 (CSR3)

CSR3 contains the BDM flash clock divider (BFCDIV) value in a format similar to HCS08 devices.

There are multiple ways to reference CSR3. They are summarized in [Table 26-10](#).

**Table 26-10. CSR3 Reference Summary**

Method	Reference Details
READ_CSR3_BYTE	Reads CSR3[31–24] from the BDM interface. Available in all modes.
WRITE_CSR3_BYTE	Writes CSR3[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.



**Table 26-10. CSR3 Reference Summary (continued)**

Method	Reference Details
WRITE_DREG	Writes CSR3[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	No operation during the core's execution of a WDEBUG instruction

DRc: 0x03 (CSR3)

Access: Supervisor write-only  
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	0	BFC DIV8	BFCDIV						0	0	0	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 26-7. Configuration/Status Register 3 (CSR3)****Table 26-11. CSR3 Field Descriptions**

Field	Description
31	Reserved, must be cleared.
30 BFCDIV8	BDM flash clock divide by 8. 0 Input to the flash clock divider is the bus clock 1 Input to the flash clock divider is the bus clock divided by 8
29–24 BFCDIV	BDM flash clock divider. The BFCDIV8 and BFCDIV fields specify the frequency of the internal flash clock when performing a mass erase operation initiated by setting XCSR[ERASE]. These fields must be loaded with the appropriate values prior to the setting of XCSR[ERASE] to initiate a mass erase operation in the flash memory.  This field divides the bus clock (or the bus clock divided by 8 if BFCDIV8 is set) by the value defined by the BFCDIV plus one. The resulting frequency of the internal flash clock must fall within the range of 150–200 kHz for proper flash operations. Program/erase timing pulses are one cycle of this internal flash clock, which corresponds to a range of 5–6.7 $\mu$ s. The automated programming logic uses an integer number of these pulses to complete an erase or program operation.  if BFCDIV8 = 0, then $f_{\text{FLK}} = f_{\text{BUS}} \div (\text{BFCDIV} + 1)$ if BFCDIV8 = 1, then $f_{\text{FLK}} = f_{\text{BUS}} \div (8 \times (\text{BFCDIV} + 1))$  where $f_{\text{FLK}}$ is the frequency of the flash clock and $f_{\text{BUS}}$ is the frequency of the bus clock.
23–0	Reserved for future use by the debug module, must be cleared.

### 26.3.5 BDM Address Attribute Register (BAAR)

BAAR defines the address space for memory-referencing BDM commands. BAAR[R, SZ] are loaded directly from the BDM command, while the lower five bits can be programmed from the external development system. BAAR is loaded any time AATR is written and is initialized to a value of 0x05,

setting supervisor data as the default address space. The upper 24 bits of this register are reserved for future use and any attempted write of these bits is ignored.

DRc: 0x05 (BAAR)

Access: Supervisor write-only  
BDM 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R	SZ	TT	TM					
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	1	0	1	

Figure 26-8. BDM Address Attribute Register (BAAR)

Table 26-12. BAAR Field Descriptions

Field	Description
31–8	Reserved for future use by the debug module, must be cleared.
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, <a href="#">Section 26.3.6, “Address Attribute Trigger Register (AATR)”</a> .
2–0 TM	Transfer modifier. See the TM definition in the AATR description, <a href="#">Section 26.3.6, “Address Attribute Trigger Register (AATR)”</a> .

### 26.3.6 Address Attribute Trigger Register (AATR)

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 high-speed local 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 WRITE\_DREG command.

DRc: 0x06 (AATR)

Access: Supervisor write-only  
BDM 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RM	SZM	TTM	TMM					R	SZ	TT	TM					
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	1	0	1	

Figure 26-9. Address Attribute Trigger Register (AATR)

Table 26-13. AATR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15 RM	Read/write mask. Masks the R bit in address comparisons.
14–13 SZM	Size mask. Masks the corresponding SZ bit in address comparisons.
12–11 TTM	Transfer type mask. Masks the corresponding TT bit in address comparisons.
10–8 TMM	Transfer modifier mask. 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
4–3 TT	Transfer type. Compared with the local bus transfer type signals. These bits also define the TT encoding for BDM memory commands. 00 Normal processor access Else Reserved
2–0 TM	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). 000 Reserved 001 User-mode data access 010 User-mode code access 011 Reserved 100 Reserved 101 Supervisor-mode data access 110 Supervisor-mode code access 111 Reserved

### 26.3.7 Trigger Definition Register (TDR)

TDR configures the operation of the hardware breakpoint logic that corresponds 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 one- or two-level trigger. TDR[31–16] defines the second-level trigger, and TDR[15–0] defines the 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 (clear TDR[L2EBL,L1EBL]) 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 WRITE\_DREG command.

DRc: 0x07 (TDR)

Access: Supervisor write-only  
BDM write-only

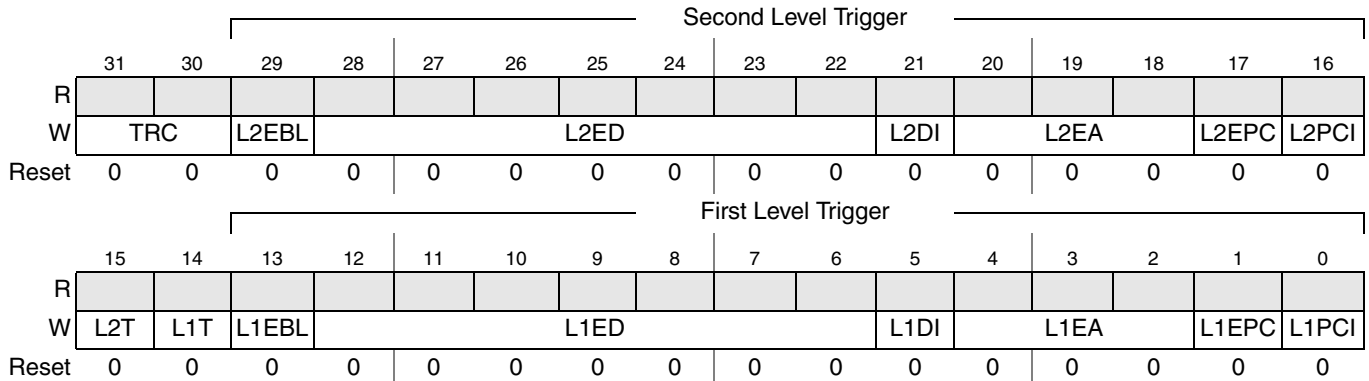


Figure 26-10. Trigger Definition Register (TDR)

Table 26-14. 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 displayed on PST. 00 Display on PST 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. <table border="1" style="margin-left: 40px;"> <thead> <tr> <th>TDR Bit</th><th>Description</th></tr> </thead> <tbody> <tr> <td>28</td><td>Data longword. Entire processor's local data bus.</td></tr> <tr> <td>27</td><td>Lower data word.</td></tr> <tr> <td>26</td><td>Upper data word.</td></tr> <tr> <td>25</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr> <tr> <td>24</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr> <tr> <td>23</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr> <tr> <td>22</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr> </tbody> </table>	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.
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 26-14. TDR Field Descriptions (continued)

Field	Description								
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.								
20–18 L2EA	Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. <table border="1" data-bbox="472 499 1252 789"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	TDR Bit	Description	20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	18	Address breakpoint low. The breakpoint is based on the address in the ABLR.
TDR Bit	Description								
20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.								
19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.								
18	Address breakpoint low. The breakpoint is based on the address in the ABLR.								
17 L2EPC	Enable level 2 PC breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint								
16 L2PCI	Level 2 PC breakpoint invert. 0 The PC breakpoint is defined within the region defined by PBR <sub>n</sub> and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR <sub>n</sub> and PBMR.								
15 L2T	Level 2 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and 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)								
14 L1T	Level 1 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and 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)								
13 L1EBL	Enable level 1 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 1 breakpoints 1 Enables all level 1 breakpoint triggers								

Table 26-14. TDR Field Descriptions (continued)

Field	Description																
12–6 L1ED	<p>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.</p> <table border="1"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	TDR Bit	Description	12	Data longword. Entire processor's local data bus.	11	Lower data word.	10	Upper data word.	9	Lower lower data byte. Low-order byte of the low-order word.	8	Lower middle data byte. High-order byte of the low-order word.	7	Upper middle data byte. Low-order byte of the high-order word.	6	Upper upper data byte. High-order byte of the high-order word.
TDR Bit	Description																
12	Data longword. Entire processor's local data bus.																
11	Lower data word.																
10	Upper data word.																
9	Lower lower data byte. Low-order byte of the low-order word.																
8	Lower middle data byte. High-order byte of the low-order word.																
7	Upper middle data byte. Low-order byte of the high-order word.																
6	Upper upper data byte. High-order byte of the high-order word.																
5 L1DI	<p>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.</p> <p>0 No inversion 1 Invert data breakpoint comparators.</p>																
4–2 L1EA	<p>Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.</p> <table border="1"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	TDR Bit	Description	4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.								
TDR Bit	Description																
4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.																
3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.																
2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.																
1 L1EPC	<p>Enable level 1 PC breakpoint.</p> <p>0 Disable PC breakpoint 1 Enable PC breakpoint</p>																
0 L1PCI	<p>Level 1 PC breakpoint invert.</p> <p>0 The PC breakpoint is defined within the region defined by PBR<math>n</math> and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR<math>n</math> and PBMR.</p>																

### 26.3.8 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

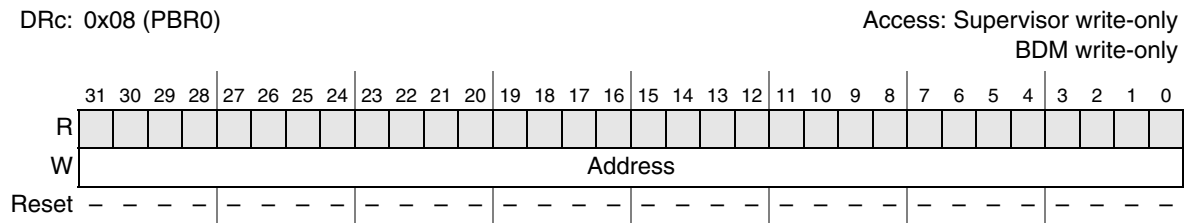
The PBR $n$  registers define instruction addresses for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR is configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR. The PC breakpoint registers, PBR1–3, have no masking associated with them, but do include a

valid bit. These registers' contents 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 WRITE\_DREG command using values shown in [Section 26.4.1.4, "BDM Command Set Descriptions"](#).

### NOTE

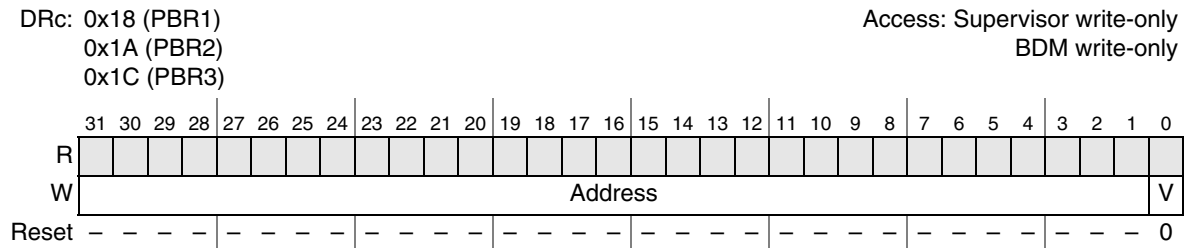
Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.



**Figure 26-11. Program Counter Breakpoint Register 0 (PBR0)**

**Table 26-15. PBR0 Field Descriptions**

Field	Description
31–0 Address	PC breakpoint address. The address to be compared with the PC as a breakpoint trigger. Because all instruction sizes are multiples of 2 bytes, bit 0 of the address should always be zero.



**Figure 26-12. Program Counter Breakpoint Register  $n$  (PBR $n$ ,  $n = 1,2,3$ )**

**Table 26-16. 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 26-13](#) shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WRITE\_DREG command. PBMR only masks PBR0.

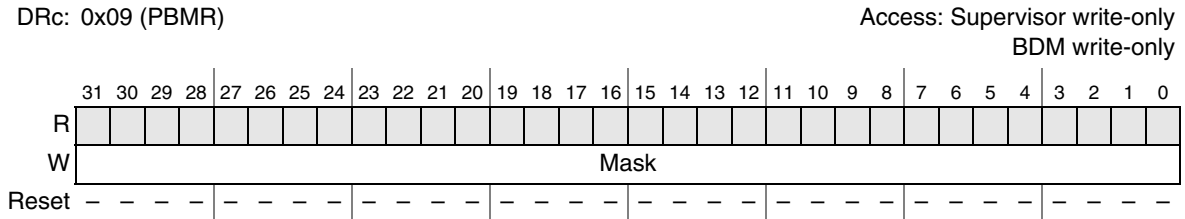


Figure 26-13. Program Counter Breakpoint Mask Register (PBMR)

Table 26-17. PBMR Field Descriptions

Field	Description
31–0 Mask	PC breakpoint mask. If using PBR0, this register must be initialized since it is undefined after reset. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

### 26.3.9 Address Breakpoint Registers (ABLR, ABHR)

The ABLR and ABHR define regions in the processor’s data address space that can be used 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:

- Identical to the value in ABLR
- Inside the range bound by ABLR and ABHR inclusive
- Outside that same range

The address breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE\_DREG command using values shown in [Section 26.4.1.4, “BDM Command Set Descriptions.”](#)

#### NOTE

Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.

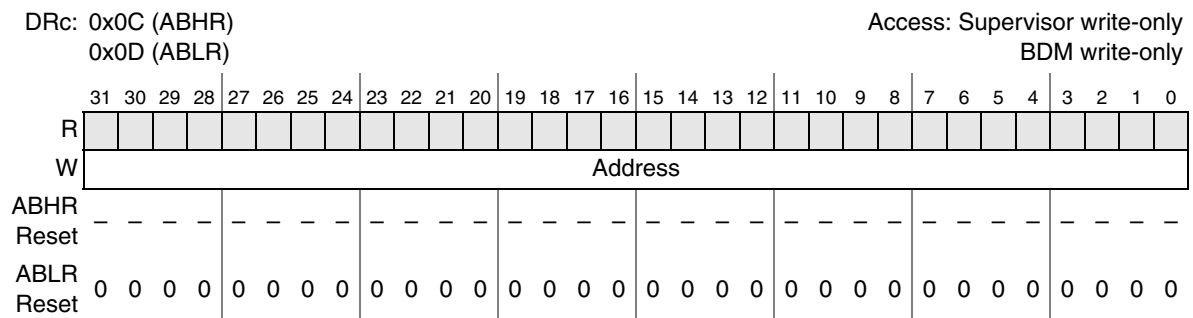


Figure 26-14. Address Breakpoint Registers (ABLR, ABHR)





**Table 26-22. 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]

### 26.3.10.1 Resulting Set of Possible Trigger Combinations

The resulting set of possible breakpoint trigger combinations consists of the following options where `||` denotes logical OR, `&&` denotes logical AND, and `{ }` denotes an optional additional trigger term:

One-level triggers of the form:

```
if (PC_breakpoint)
if (PC_breakpoint || Address_breakpoint{&& Data_breakpoint})
if (Address_breakpoint {&& Data_breakpoint})
```

Two-level triggers of the form:

```
if (PC_breakpoint)
then if (Address_breakpoint{&& Data_breakpoint})

if (Address_breakpoint {&& Data_breakpoint})
then if (PC_breakpoint)
```

In these examples, `PC_breakpoint` is the logical summation of the PBR0/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; `Address_breakpoint` is a function of ABHR, ABLR, and AATR; `Data_breakpoint` is a function of DBR and DBMR. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

The breakpoint registers can also be used to define the start and stop recording conditions for the PST trace buffer. For information on this functionality, see [Section 26.3.3, “Configuration/Status Register 2 \(CSR2\)”](#).

### 26.3.11 PST Buffer (PSTB)

The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. See [Figure 26-16](#) for an illustration of how the buffer entries are packed.

The write pointer for the trace buffer is available as `CSR2[PSTBWA]`. Using this pointer, it is possible to determine the oldest-to-newest entries in the trace buffer.

Core register number (CRN)	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			
	0x10	TB #00				TB #01				TB #02				TB #03				TB #04				05[5:4]										
0x11	TB #05[3:0]				TB #06				TB #07				TB #08				TB #09				TB #10[5:2]											
0x12	10[1:0]				TB #11				TB #12				TB #13				TB #14				TB #15											
0x13	TB #16				TB #17				TB #18				TB #19				TB #20				21[5:4]											
0x14	TB #21[3:0]				TB #22				TB #23				TB #24				TB #25				TB #26[5:2]											
0x15	26[1:0]				TB #27				TB #28				TB #29				TB #30				TB #31											
0x16	TB #32				TB #33				TB #34				TB #35				TB #36				37[5:4]											
0x17	TB #37[3:0]				TB #38				TB #39				TB #40				TB #41				TB #42[5:2]											
0x18	42[1:0]				TB #43				TB #44				TB #45				TB #46				TB #47											
0x19	TB #48				TB #49				TB #50				TB #51				TB #52				53[5:4]											
0x1A	TB #53[3:0]				TB #54				TB #55				TB #56				TB #57				TB #58[5:2]											
0x1B	58[1:0]				TB #59				TB #60				TB #61				TB #62				TB #63											

Figure 26-16. PST Trace Buffer Entries and Locations

## 26.4 Functional Description

### 26.4.1 Background Debug Mode (BDM)

This section provides details on the background debug serial interface controller (BDC) and the BDM command set.

The BDC provides a single-wire debug interface to the target MCU. As shown in the Version 1 ColdFire core block diagram of [Figure 26-1](#), the BDC module interfaces between the single-pin (BKGD) interface and the remaining debug modules, including the ColdFire background debug logic, the real-time debug hardware, and the PST/DDATA trace logic. This interface provides a convenient means for programming the on-chip flash and other non-volatile memories. The BDC is the primary debug interface for development and allows non-intrusive access to memory data and traditional debug features such as run/halt control, read/write of core registers, breakpoints, and single instruction step.

Features of the background debug controller (BDC) include:

- Single dedicated pin for mode selection and background communications
- Special BDC registers not located in system memory map
- SYNC command to determine target communications rate
- Non-intrusive commands for memory access
- Active background (halt) mode commands for core register access
- GO command to resume execution
- BACKGROUND command to halt core or wake CPU from low-power modes
- Oscillator runs in stop mode, if BDM enabled

Based on these features, BDM is useful for the following reasons:

- 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 memory downloading, especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This feature allows quick hardware debugging with the same tool set used for firmware development.

### 26.4.1.1 CPU Halt

Although certain 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. Recall that the default configuration of the Version 1 ColdFire core (CF1Core) defines the occurrence of certain exception types to automatically generate a system reset. Some of these fault types include illegal instructions, privilege errors, address errors, and bus error terminations, with the response under control of the processor's CPUCR[ARD, IRD] bits.

**Table 26-23. CPU Halt Sources**

Halt Source	Halt Timing	Description			
Fault-on-fault	Immediate	Refers to the occurrence of any fault while exception processing. For example, a bus error is signaled during exception stack frame writes or while fetching the first instruction in the exception service routine.			
		CPUCR[ARD] = 1	Immediately enters halt.		
		CPUCR[ARD] = 0	Reset event is initiated.		
Hardware breakpoint trigger	Pending	Halt is made pending in the processor. The processor 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.			
HALT instruction	Immediate	BDM disabled	CPUCR[IRD] = 0	A reset is initiated since attempted execution of an illegal instruction	
			CPUCR[IRD] = 1	An illegal instruction exception is generated.	
		BDM enabled, supervisor mode	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.		
			BDM enabled, user mode	CSR[UHE] = 0 CPUCR[IRD] = 0	A reset event is initiated, because a privileged instruction was attempted in user mode.
				CSR[UHE] = 0 CPUCR[IRD] = 1	A privilege violation exception is generated.
				CSR[UHE] = 1	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.

Table 26-23. CPU Halt Sources (continued)

Halt Source	Halt Timing	Description		
BACKGROUND command	Pending	BDM disabled or flash secure	Illegal command response and BACKGROUND command is ignored.	
		BDM enabled and flash unsecure	Processor is running	Halt is made pending in the processor. The processor 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.
			Processor is stopped	Processing of the BACKGROUND command is treated in a special manner. 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 (the instruction following STOP).
PSTB full condition	Pending	PSTB	PSTB obtrusive recording mode pends halt in the processor if the trace buffer reaches its full threshold (full is defined as before the buffer is overwritten). When a pending condition is asserted, the processor halts at the next sample point.	
BKGD held low for $\geq 2$ bus clocks after reset negated for POR or BDM reset	Immediate	Flash unsecure	Enters debug mode with XCSR[ENBDM, CLKSW] set. The full set of BDM commands is available and debug can proceed. If the core is reset into a debug halt condition, the processor's response to the GO command depends on the BDM command(s) performed while it was halted. 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.	
		Flash secure	Enters debug mode with XCSR[ENBDM, CLKSW] set. The allowable commands are limited to the always-available group. A GO command to start the processor is not allowed. The only recovery actions in this mode are: <ul style="list-style-type: none"> <li>Issue a BDM reset setting CSR2[BDFR] with CSR2[BDHBR] cleared and the BKGD pin held high to reset into normal operating mode</li> <li>Erase the flash to unsecure the memory and then proceed with debug</li> <li>Power cycle the device with the BKGD pin held high to reset into the normal operating mode</li> </ul>	

If the VBus is enabled, 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 behavior of the  $\overline{\text{BKPT}}$  input pin is equivalent to receiving a BACKGROUND command via the 1-pin BDM interface.

The processor's run/stop/halt status is always accessible in XCSR[CPUHALT,CPUSTOP]. Additionally, CSR[27–24] indicate the halt source, showing the highest priority source for multiple halt conditions. This field is cleared by a read of the CSR. A processor halt due to the PSTB full condition as indicated by CSR2[PSTH] is also reflected in CSR[BKPT]. The debug GO command clears CSR[26–24] and CSR2[PSTBH].

### 26.4.1.2 Background Debug Serial Interface Controller (BDC)

BDC serial communications use a custom serial protocol first introduced on the M68HC12 Family of microcontrollers and later used in the M68HCS08 family. This protocol assumes that the host knows the communication clock rate determined by the target BDC clock rate. The BDC clock rate may be the system bus clock frequency or an alternate frequency source depending on the state of XCSR[CLKSW]. All communication is initiated and controlled by the host which drives a high-to-low edge to signal the beginning of each bit time. Commands and data are sent most significant bit (msb) first. For a detailed description of the communications protocol, refer to [Section 26.4.1.3, “BDM Communication Details”](#).

If a host is attempting to communicate with a target MCU that has an unknown BDC clock rate, a SYNC command may be sent to the target MCU to request a timed synchronization response signal from which the host can determine the correct communication speed. After establishing communications, the host can read XCSR and write the clock switch (CLKSW) control bit to change the source of the BDC clock for further serial communications if necessary.

BKGD is a pseudo-open-drain pin and there is an on-chip pullup so no external pullup resistor is required. Unlike typical open-drain pins, the external RC time constant on this pin, which is influenced by external capacitance, plays almost no role in signal rise time. The custom protocol provides for brief, actively driven speed-up pulses to force rapid rise times on this pin without risking harmful drive level conflicts. Refer to [Section 26.4.1.3, “BDM Communication Details,”](#) for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector ([Section 26.4.5, “Freescale-Recommended BDM Pinout”](#)), the internal pullup on BKGD chooses normal operating mode. When a development system is connected, it can pull BKGD and  $\overline{\text{RESET}}$  low, release  $\overline{\text{RESET}}$  to select active background (halt) mode rather than normal operating mode, and then release BKGD. It is not necessary to reset the target MCU to communicate with it through the background debug interface. There is also a mechanism to generate a reset event in response to setting CSR2[BDFR].

### 26.4.1.3 BDM Communication Details

The BDC serial interface requires the external host controller to generate a falling edge on the BKGD pin to indicate the start of each bit time. The external controller provides this falling edge whether data is transmitted or received.

BKGD is a pseudo-open-drain pin that can be driven by an external controller or by the MCU. Data is transferred msb first at 16 BDC clock cycles per bit (nominal speed). The interface times-out if 512 BDC clock cycles occur between falling edges from the host. If a time-out occurs, the status of any command in progress must be determined before new commands can be sent from the host. To check the status of the command, follow the steps detailed in the bit description of XCSR[CSTAT] in [Table 26-7](#).

The custom serial protocol requires the debug pod to know the target BDC communication clock speed. The clock switch (CLKSW) control bit in the XCSR[31–24] register allows you to select the BDC clock source. The BDC clock source can be the bus clock or the alternate BDC clock source. When the MCU is reset in normal user mode, CLKSW is cleared and that selects the alternate clock source. This clock source is a fixed frequency independent of the bus frequency so it does change if the user modifies clock generator settings. This is the preferred clock source for general debugging.

When the MCU is reset in active background (halt) mode, CLKSW is set which selects the bus clock as the source of the BDC clock. This CLKSW setting is most commonly used during flash memory programming because the bus clock can usually be configured to operate at the highest allowed bus frequency to ensure the fastest possible flash programming times. Because the host system is in control of changes to clock generator settings, it knows when a different BDC communication speed should be used. The host programmer also knows that no unexpected change in bus frequency could occur to disrupt BDC communications.

Normally, setting CLKSW should not be used for general debugging because there is no way to ensure the application program does not change the clock generator settings. This is especially true in the case of application programs that are not yet fully debugged.

After any reset (or at any other time), the host system can issue a SYNC command to determine the speed of the BDC clock. CLKSW may be written using the serial WRITE\_XCSR\_BYTE command through the BDC interface. CLKSW is located in the special XCSR byte register in the BDC module and it is not accessible in the normal memory map of the ColdFire core. This means that no program running on the processor can modify this register (intentionally or unintentionally).

The BKGD pin can receive a high- or low-level or transmit a high- or low-level. The following diagrams show timing for each of these cases. Interface timing is synchronous to clocks in the target BDC, but asynchronous to the external host. The internal BDC clock signal is shown for reference in counting cycles.

Figure 26-17 shows an external host transmitting a logic 1 or 0 to the BKGD pin of a target MCU. The host is asynchronous to the target so there is a 0–1 cycle delay from the host-generated falling edge to where the target perceives the beginning of the bit time. Ten target BDC clock cycles later, the target senses the bit level on the BKGD pin. Typically, the host actively drives the pseudo-open-drain BKGD pin during host-to-target transmissions to speed up rising edges. Because the target does not drive the BKGD pin during the host-to-target transmission period, there is no need to treat the line as an open-drain signal during this period.

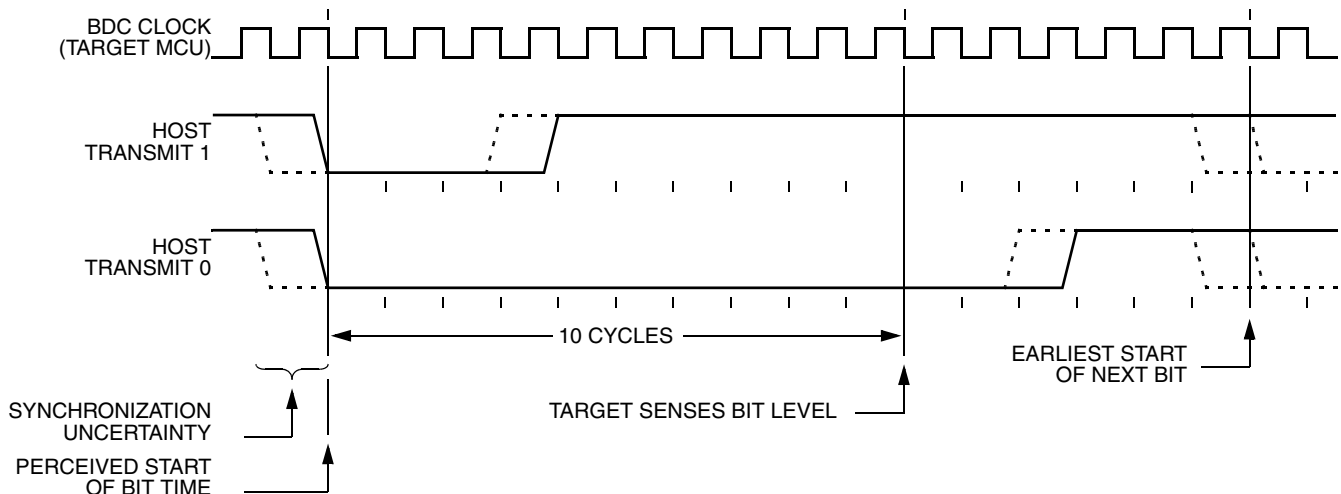
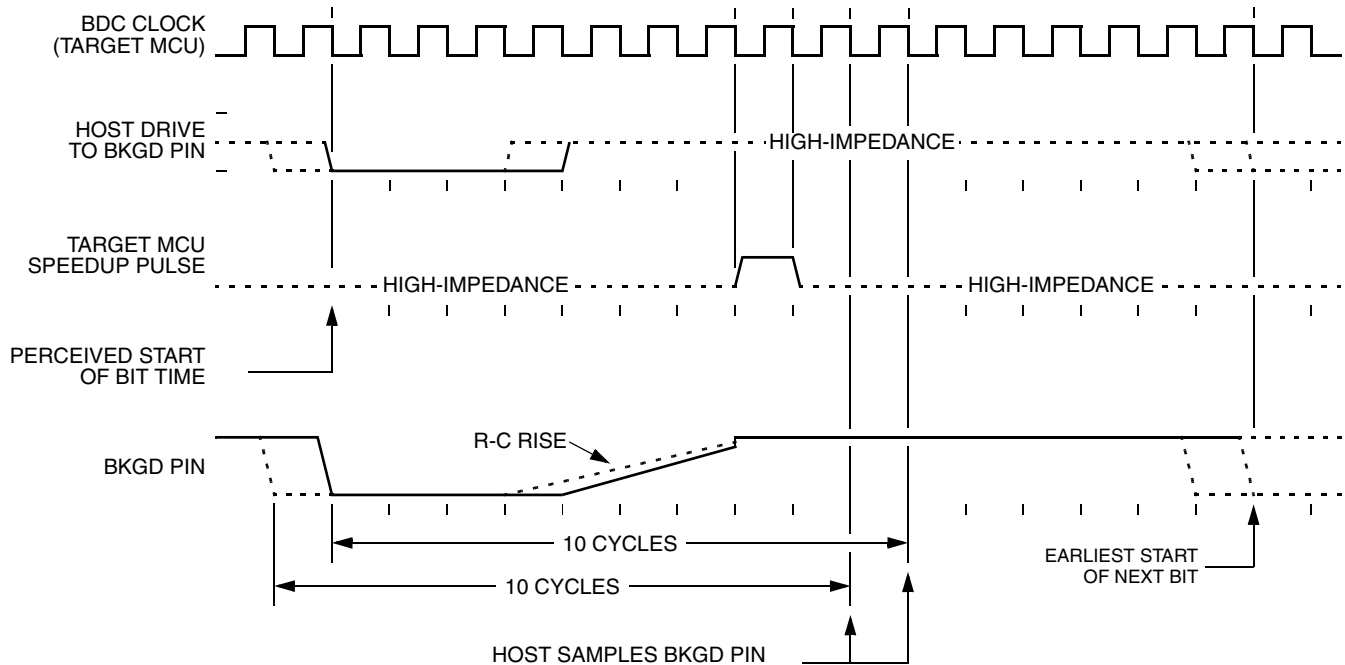


Figure 26-17. BDC Host-to-Target Serial Bit Timing

Figure 26-18 shows the host receiving a logic 1 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the perceived start of the bit time in the target MCU. The host holds the BKGD pin low long enough for the target to recognize it (at least two target BDC cycles). The host must release the low drive before the target MCU drives a brief active-high speedup pulse seven cycles after the perceived start of the bit time. The host should sample the bit level about 10 cycles after it started the bit time.



**Figure 26-18. BDC Target-to-Host Serial Bit Timing (Logic 1)**

Figure 26-19 shows the host receiving a logic 0 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the start of the bit time as perceived by the target MCU. The host initiates the bit time, but the target MCU finishes it. Because the target wants the host to receive a logic 0, it drives the BKGD pin low for 13 BDC clock cycles, then briefly drives it high to speed up the rising edge. The host samples the bit level about 10 cycles after starting the bit time.



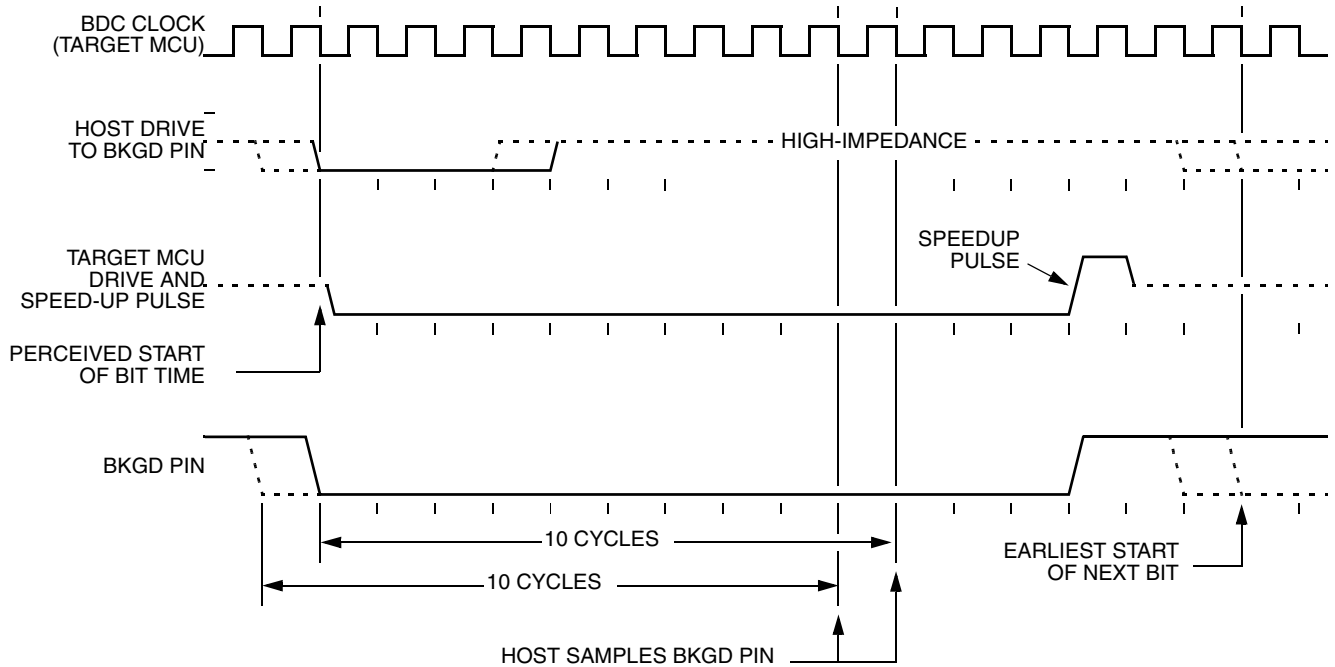


Figure 26-19. BDM Target-to-Host Serial Bit Timing (Logic 0)

#### 26.4.1.4 BDM Command Set Descriptions

This section presents detailed descriptions of the BDM commands.

The V1 BDM command set is based on transmission of one or more 8-bit data packets per operation. Each operation begins with a host-to-target transmission of an 8-bit command code packet. The command code definition broadly maps the operations into four formats as shown in [Figure 26-20](#).

**Miscellaneous Commands**

	7	6	5	4	3	2	1	0
W	0	0	R/ $\bar{W}$	0	MSCMD			
R/W	Optional Command Extension Byte (Data)							

**Memory Commands**

	7	6	5	4	3	2	1	0
W	0	0	R/ $\bar{W}$	1	SZ		MCMD	
W if addr, R/W if data	Command Extension Bytes (Address, Data)							

**Core Register Commands**

	7	6	5	4	3	2	1	0
W	CRG		R/ $\bar{W}$	CRN				
R/W	Command Extension Bytes (Data)							

**PST Trace Buffer Read Commands**

	7	6	5	4	3	2	1	0
W	0	1	0	CRN				
R	Trace Buffer Data[31–24], see <a href="#">Figure 26-16</a>							
R	Trace Buffer Data[23–16], see <a href="#">Figure 26-16</a>							
R	Trace Buffer Data[15–08], see <a href="#">Figure 26-16</a>							
R	Trace Buffer Data[07–00], see <a href="#">Figure 26-16</a>							

**Figure 26-20. BDM Command Encodings**

Table 26-24. BDM Command Field Descriptions

Field	Description																								
5 R/W	Read/Write. 0 Command is performing a write operation. 1 Command is performing a read operation.																								
3–0 MSCMD	Miscellaneous command. Defines the miscellaneous command to be performed. 0000 No operation 0001 Display the CPU's program counter (PC) plus optional capture in the PST trace buffer 0010 Enable the BDM acknowledge communication mode 0011 Disable the BDM acknowledge communication mode 0100 Force a CPU halt (background) 1000 Resume CPU execution (go) 1101 Read/write of the debug XCSR most significant byte 1110 Read/write of the debug CSR2 most significant byte 1111 Read/write of the debug CSR3 most significant byte																								
3–2 SZ	Memory operand size. Defines the size of the memory reference. 00 8-bit byte 01 16-bit word 10 32-bit long																								
1–0 MCMD	Memory command. Defines the type of the memory reference to be performed. 00 Simple write if R/W = 0; simple read if R/W = 1 01 Write + status if R/W = 0; read + status if R/W = 1 10 Fill if R/W = 0; dump if R/W = 1 11 Fill + status if R/W = 0; dump + status if R/W = 1																								
7–6 CRG	Core register group. Defines the core register group to be referenced. 01 CPU's general-purpose registers (An, Dn) or PST trace buffer 10 Debug's control registers 11 CPU's control registers (PC, SR, VBR, CPUCR,...)																								
4–0 CRN	Core register number. Defines the specific core register (its number) to be referenced. All other CRN values are reserved. <table border="1" data-bbox="418 1192 1305 1654"> <thead> <tr> <th>CRG</th> <th>CRN</th> <th>Register</th> </tr> </thead> <tbody> <tr> <td rowspan="3">01</td> <td>0x00–0x07</td> <td>D0–7</td> </tr> <tr> <td>0x08–0x0F</td> <td>A0–7</td> </tr> <tr> <td>0x10–0x1B</td> <td>PST Buffer 0–11</td> </tr> <tr> <td>10</td> <td colspan="2">DRc[4:0] as described in <a href="#">Table 26-4</a></td> </tr> <tr> <td rowspan="5">11</td> <td>0x00</td> <td>OTHER_A7</td> </tr> <tr> <td>0x01</td> <td>VBR</td> </tr> <tr> <td>0x02</td> <td>CPUCR</td> </tr> <tr> <td>0x0E</td> <td>SR</td> </tr> <tr> <td>0x0F</td> <td>PC</td> </tr> </tbody> </table>	CRG	CRN	Register	01	0x00–0x07	D0–7	0x08–0x0F	A0–7	0x10–0x1B	PST Buffer 0–11	10	DRc[4:0] as described in <a href="#">Table 26-4</a>		11	0x00	OTHER_A7	0x01	VBR	0x02	CPUCR	0x0E	SR	0x0F	PC
CRG	CRN	Register																							
01	0x00–0x07	D0–7																							
	0x08–0x0F	A0–7																							
	0x10–0x1B	PST Buffer 0–11																							
10	DRc[4:0] as described in <a href="#">Table 26-4</a>																								
11	0x00	OTHER_A7																							
	0x01	VBR																							
	0x02	CPUCR																							
	0x0E	SR																							
	0x0F	PC																							

## 26.4.1.5 BDM Command Set Summary

Table 26-25 summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in Table 26-25 to describe the structure of the BDM commands.

Commands begin with an 8-bit hexadecimal command code in the host-to-target direction (most significant bit first)

/	=	separates parts of the command
d	=	delay 32 target BDC clock cycles
ad24	=	24-bit memory address in the host-to-target direction
rd8	=	8 bits of read data in the target-to-host direction
rd16	=	16 bits of read data in the target-to-host direction
rd32	=	32 bits of read data in the target-to-host direction
rd.sz	=	read data, size defined by sz, in the target-to-host direction
wd8	=	8 bits of write data in the host-to-target direction
wd16	=	16 bits of write data in the host-to-target direction
wd32	=	32 bits of write data in the host-to-target direction
wd.sz	=	write data, size defined by sz, in the host-to-target direction
ss	=	the contents of XCSR[31:24] in the target-to-host direction (STATUS)
sz	=	memory operand size (0b00 = byte, 0b01 = word, 0b10 = long)
crn	=	core register number
WS	=	command suffix signaling the operation is with status

**Table 26-25. BDM Command Summary**

Command Mnemonic	Command Classification	ACK if Enb? <sup>1</sup>	Command Structure	Description
SYNC	Always Available	N/A	N/A <sup>2</sup>	Request a timed reference pulse to determine the target BDC communication speed
ACK_DISABLE	Always Available	No	0x03/d	Disable the communication handshake. This command does not issue an ACK pulse.
ACK_ENABLE	Always Available	Yes	0x02/d	Enable the communication handshake. Issues an ACK pulse after the command is executed.
BACKGROUND	Non-Intrusive	Yes	0x04/d	Halt the CPU if ENBDM is set. Otherwise, ignore as illegal command.
DUMP_MEM.sz	Non-Intrusive	Yes	(0x32+4 x sz)/d/rd.sz	Dump (read) memory based on operand size (sz). Used with READ_MEM to dump large blocks of memory. An initial READ_MEM is executed to set up the starting address of the block and to retrieve the first result. Subsequent DUMP_MEM commands retrieve sequential operands.

Table 26-25. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb? <sup>1</sup>	Command Structure	Description
DUMP_MEM.sz_WS	Non-Intrusive	No	(0x33+4 x sz)/d/ss/rd.sz	Dump (read) memory based on operand size (sz) and report status. Used with READ_MEM{_WS} to dump large blocks of memory. An initial READ_MEM{_WS} is executed to set up the starting address of the block and to retrieve the first result. Subsequent DUMP_MEM{_WS} commands retrieve sequential operands.
FILL_MEM.sz	Non-Intrusive	Yes	(0x12+4 x sz)/wd.sz/d	Fill (write) memory based on operand size (sz). Used with WRITE_MEM to fill large blocks of memory. An initial WRITE_MEM is executed to set up the starting address of the block and to write the first operand. Subsequent FILL_MEM commands write sequential operands.
FILL_MEM.sz_WS	Non-Intrusive	No	(0x13+4 x sz)/wd.sz/d/ss	Fill (write) memory based on operand size (sz) and report status. Used with WRITE_MEM{_WS} to fill large blocks of memory. An initial WRITE_MEM{_WS} is executed to set up the starting address of the block and to write the first operand. Subsequent FILL_MEM{_WS} commands write sequential operands.
GO	Non-Intrusive	Yes	0x08/d	Resume the CPU's execution <sup>3</sup>
NOP	Non-Intrusive	Yes	0x00/d	No operation
READ_CREG	Active Background	Yes	(0xE0+CRN)/d/rd32	Read one of the CPU's control registers
READ_DREG	Non-Intrusive	Yes	(0xA0+CRN)/d/rd32	Read one of the debug module's control registers
READ_MEM.sz	Non-Intrusive	Yes	(0x30+4 x sz)/ad24/d/rd.sz	Read the appropriately-sized (sz) memory value from the location specified by the 24-bit address
READ_MEM.sz_WS	Non-Intrusive	No	(0x31+4 x sz)/ad24/d/ss/rd.sz	Read the appropriately-sized (sz) memory value from the location specified by the 24-bit address and report status
READ_PSTB	Non-Intrusive	Yes	(0x40+CRN)/d/rd32	Read the requested longword location from the PST trace buffer
READ_Rn	Active Background	Yes	(0x60+CRN)/d/rd32	Read the requested general-purpose register (An, Dn) from the CPU
READ_XCSR_BYTE	Always Available	No	0x2D/rd8	Read the most significant byte of the debug module's XCSR
READ_CSR2_BYTE	Always Available	No	0x2E/rd8	Read the most significant byte of the debug module's CSR2
READ_CSR3_BYTE	Always Available	No	0x2F/rd8	Read the most significant byte of the debug module's CSR3

Table 26-25. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb? <sup>1</sup>	Command Structure	Description
SYNC_PC	Non-Intrusive	Yes	0x01/d	Display the CPU's current PC and capture it in the PST trace buffer
WRITE_CREG	Active Background	Yes	(0xC0+CRN)/wd32/d	Write one of the CPU's control registers
WRITE_DREG	Non-Intrusive	Yes	(0x80+CRN)/wd32/d	Write one of the debug module's control registers
WRITE_MEM.sz	Non-Intrusive	Yes	(0x10+4 x sz)/ad24/wd.sz/d	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address
WRITE_MEM.sz_WS	Non-Intrusive	No	(0x11+4 x sz)/ad24/wd.sz/d/ss	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address and report status
WRITE_Rn	Active Background	Yes	(0x40+CRN)/wd32/d	Write the requested general-purpose register (An, Dn) of the CPU
WRITE_XCSR_BYTE	Always Available	No	0x0D/wd8	Write the most significant byte of the debug module's XCSR
WRITE_CSR2_BYTE	Always Available	No	0x0E/wd8	Write the most significant byte of the debug module's CSR2
WRITE_CSR3_BYTE	Always Available	No	0x0F/wd8	Write the most significant byte of the debug module's CSR3

<sup>1</sup> This column identifies if the command generates an ACK pulse if operating with acknowledge mode enabled. See Section 26.4.1.5.3, "ACK\_ENABLE," for additional information.

<sup>2</sup> The SYNC command is a special operation which does not have a command code.

<sup>3</sup> If a GO command is received while the processor is not halted, it performs no operation.

### 26.4.1.5.1 SYNC

The SYNC command is unlike other BDC commands because the host does not necessarily know the correct speed to use for serial communications until after it has analyzed the response to the SYNC command.

To issue a SYNC command, the host:

1. Drives the BKGD pin low for at least 128 cycles of the slowest possible BDC clock (bus clock or device-specific alternate clock source).
2. Drives BKGD high for a brief speed-up pulse to get a fast rise time. (This speedup pulse is typically one cycle of the host clock which is as fast as the maximum target BDC clock.)
3. Removes all drive to the BKGD pin so it reverts to high impedance.
4. Listens to the BKGD pin for the sync response pulse.

Upon detecting the sync request from the host (which is a much longer low time than would ever occur during normal BDC communications), the target:

1. Waits for BKGD to return to a logic high.

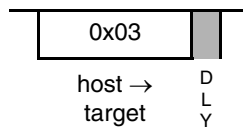
2. Delays 16 cycles to allow the host to stop driving the high speed-up pulse.
3. Drives BKGD low for 128 BDC clock cycles.
4. Drives a 1-cycle high speed-up pulse to force a fast rise time on BKGD.
5. Removes all drive to the BKGD pin so it reverts to high impedance.

The host measures the low time of this 128-cycle sync response pulse and determines the correct speed for subsequent BDC communications. Typically, the host can determine the correct communication speed within a few percent of the actual target speed and the serial protocol can easily tolerate this speed error.

#### 26.4.1.5.2 ACK\_DISABLE

Disable host/target handshake protocol

Always Available

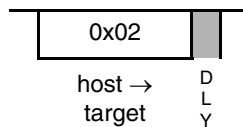


Disables the serial communication handshake protocol. The subsequent commands, issued after the ACK\_DISABLE command, do not execute the hardware handshake protocol. This command is not followed by an ACK pulse.

#### 26.4.1.5.3 ACK\_ENABLE

Enable host/target handshake protocol

Always Available



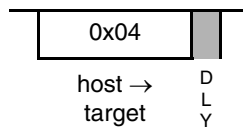
Enables the hardware handshake protocol in the serial communication. The hardware handshake is implemented by an acknowledge (ACK) pulse issued by the target MCU in response to a host command. The ACK\_ENABLE command is interpreted and executed in the BDC logic without the need to interface with the CPU. However, an acknowledge (ACK) pulse is issued by the target device after this command is executed. This feature can be used by the host to evaluate if the target supports the hardware handshake protocol. If the target supports the hardware handshake protocol, subsequent commands are enabled to execute the hardware handshake protocol, otherwise this command is ignored by the target.

For additional information about the hardware handshake protocol, refer to [Section 26.4.1.6, “Serial Interface Hardware Handshake Protocol,”](#) and [Section 26.4.1.7, “Hardware Handshake Abort Procedure.”](#)

### 26.4.1.5.4 BACKGROUND

Enter active background mode (if enabled)

Non-intrusive



Provided XCSR[ENBDM] is set (BDM enabled), the BACKGROUND command causes the target MCU to enter active background (halt) mode as soon as the current CPU instruction finishes. If ENBDM is cleared (its default value), the BACKGROUND command is ignored.

A delay of 32 BDC clock cycles is required after the BACKGROUND command to allow the target MCU to finish its current CPU instruction and enter active background mode before a new BDC command can be accepted.

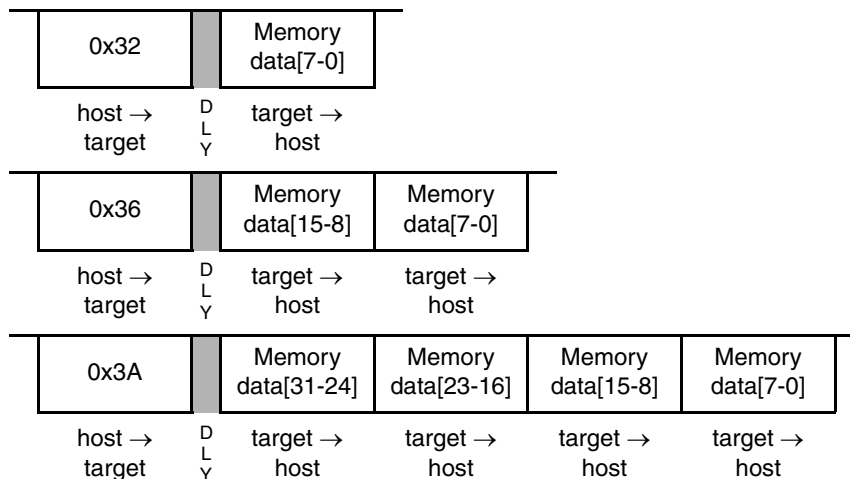
After the target MCU is reset into a normal operating mode, the host debugger would send a WRITE\_XCSR\_BYTE command to set ENBDM before attempting to send the BACKGROUND command the first time. Normally, the development host would set ENBDM once at the beginning of a debug session or after a target system reset, and then leave the ENBDM bit set during debugging operations. During debugging, the host would use GO commands to move from active background mode to normal user program execution and would use BACKGROUND commands or breakpoints to return to active background mode.

### 26.4.1.5.5 DUMP\_MEM.sz, DUMP\_MEM.sz\_WS

DUMP\_MEM.sz

Read memory specified by debug address register, then increment address

Non-intrusive

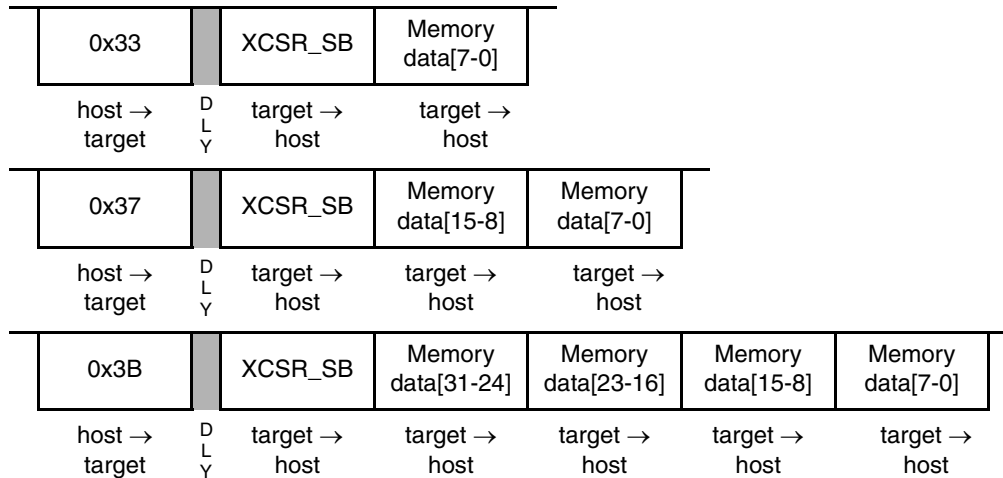




**DUMP\_MEM.sz\_WS**

Read memory specified by debug address register with status,  
then increment address

Non-intrusive



DUMP\_MEM{ \_WS } is used with the READ\_MEM{ \_WS } command to access large blocks of memory. An initial READ\_MEM{ \_WS } is executed to set-up the starting address of the block and to retrieve the first result. If an initial READ\_MEM{ \_WS } is not executed before the first DUMP\_MEM{ \_WS }, an illegal command response is returned. The DUMP\_MEM{ \_WS } command retrieves subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent DUMP\_MEM{ \_WS } commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned before the read data. XCSR\_SB reflects the state after the memory read was performed.

**NOTE**

DUMP\_MEM{ \_WS } does not check for a valid address; it is a valid command only when preceded by NOP, READ\_MEM{ \_WS }, or another DUMP\_MEM{ \_WS } command. Otherwise, an illegal command response is returned. NOP can be used for inter-command padding without corrupting the address pointer.

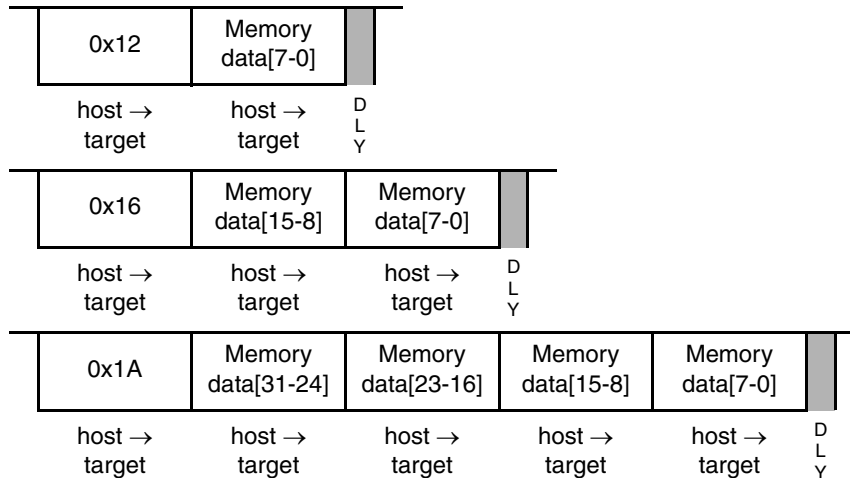
The size field (sz) is examined each time a DUMP\_MEM{ \_WS } command is processed, allowing the operand size to be dynamically altered. The examples show the DUMP\_MEM.B{ \_WS }, DUMP\_MEM.W{ \_WS } and DUMP\_MEM.L{ \_WS } commands.

### 26.4.1.5.6 FILL\_MEM.sz, FILL\_MEM.sz\_WS

#### FILL\_MEM.sz

Write memory specified by debug address register, then increment address

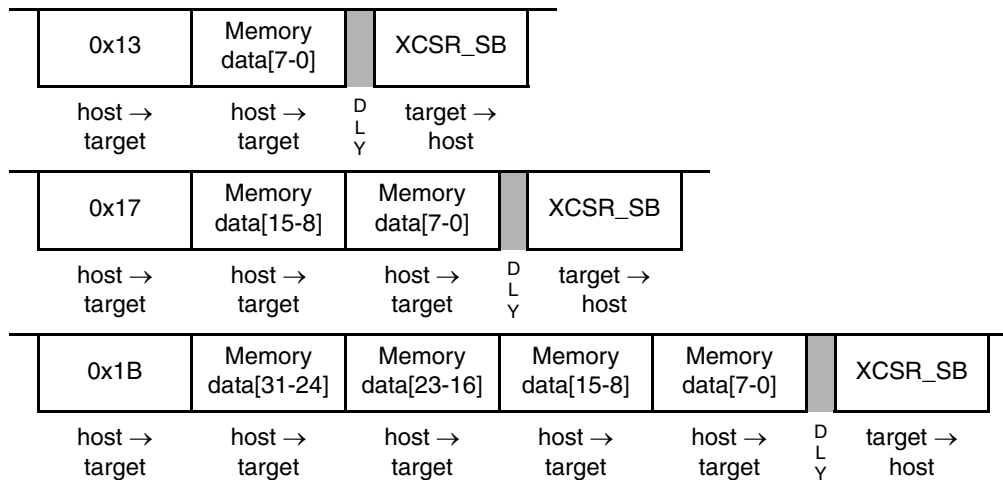
Non-intrusive



#### FILL\_MEM.sz\_WS

Write memory specified by debug address register with status, then increment address

Non-intrusive



FILL\_MEM{ \_WS } is used with the WRITE\_MEM{ \_WS } command to access large blocks of memory. An initial WRITE\_MEM{ \_WS } is executed to set up the starting address of the block and write the first datum. If an initial WRITE\_MEM{ \_WS } is not executed before the first FILL\_MEM{ \_WS }, an illegal command response is returned. The FILL\_MEM{ \_WS } command stores subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent WRITE\_MEM{ \_WS } commands use this address, perform the memory write, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified,

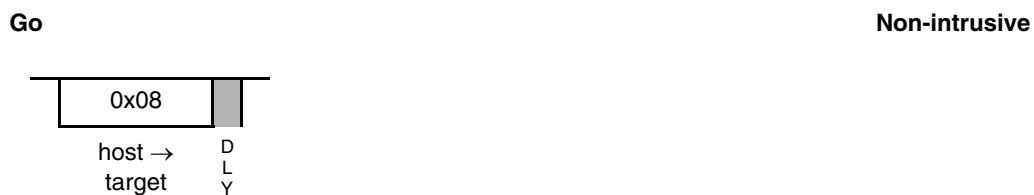
the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned after the write data. XCSR\_SB reflects the state after the memory write was performed.

### NOTE

FILL\_MEM{ \_WS } does not check for a valid address; it is a valid command only when preceded by NOP, WRITE\_MEM{ \_WS }, or another FILL\_MEM{ \_WS } command. Otherwise, an illegal command response is returned. NOP can be used for intercommand padding without corrupting the address pointer.

The size field (sz) is examined each time a FILL\_MEM{ \_WS } command is processed, allowing the operand size to be dynamically altered. The examples show the FILL\_MEM.B{ \_WS }, FILL\_MEM.W{ \_WS } and FILL\_MEM.L{ \_WS } commands.

#### 26.4.1.5.7 GO



This command is used to exit active background (halt) mode and begin (or resume) execution of the application's instructions. The CPU's 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 is issued and the CPU is not halted, the command is ignored.

#### 26.4.1.5.8 NOP

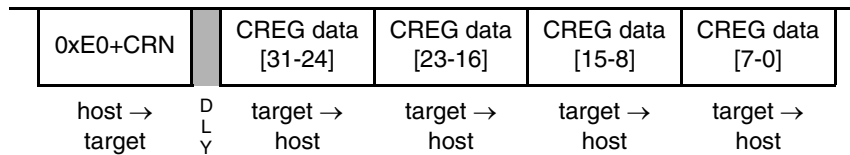


NOP performs no operation and may be used as a null command where required.

### 26.4.1.5.9 READ\_CREG

Read CPU control register

Active Background



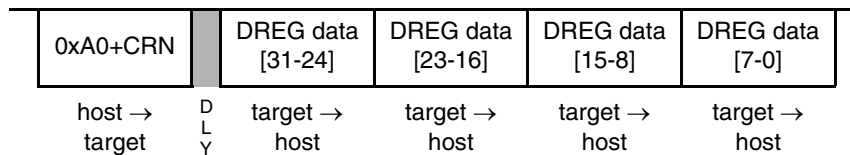
If the processor is halted, this command reads the selected control register and returns the 32-bit result. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER\_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 26-24](#) for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

### 26.4.1.5.10 READ\_DREG

Read debug control register

Non-intrusive



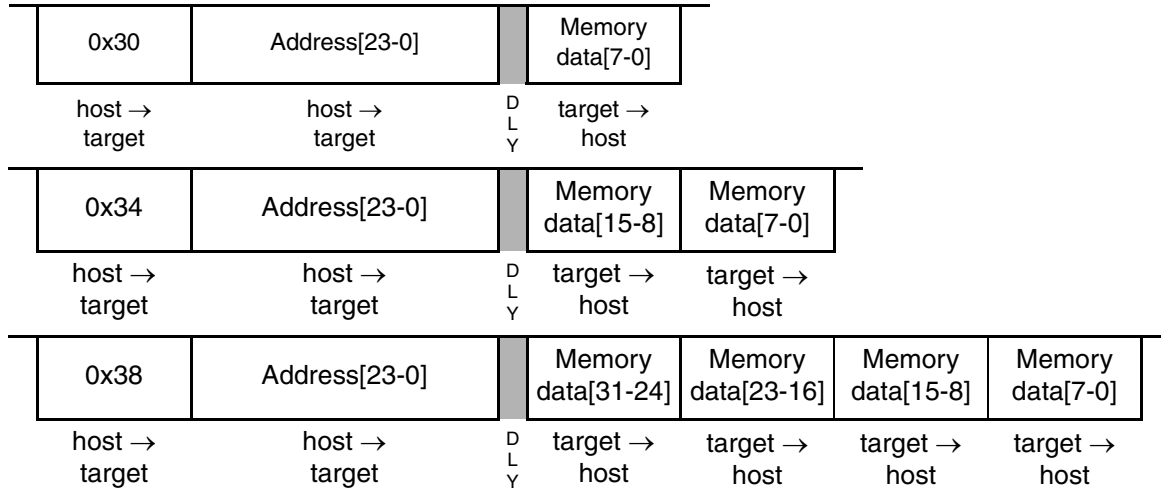
This command reads the selected debug control register and returns the 32-bit result. This register grouping includes the CSR, XCSR, CSR2, and CSR3. Accesses to debug control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 26-4](#) for CRN details.

### 26.4.1.5.11 READ\_MEM.sz, READ\_MEM.sz\_WS

#### READ\_MEM.sz

Read memory at the specified address

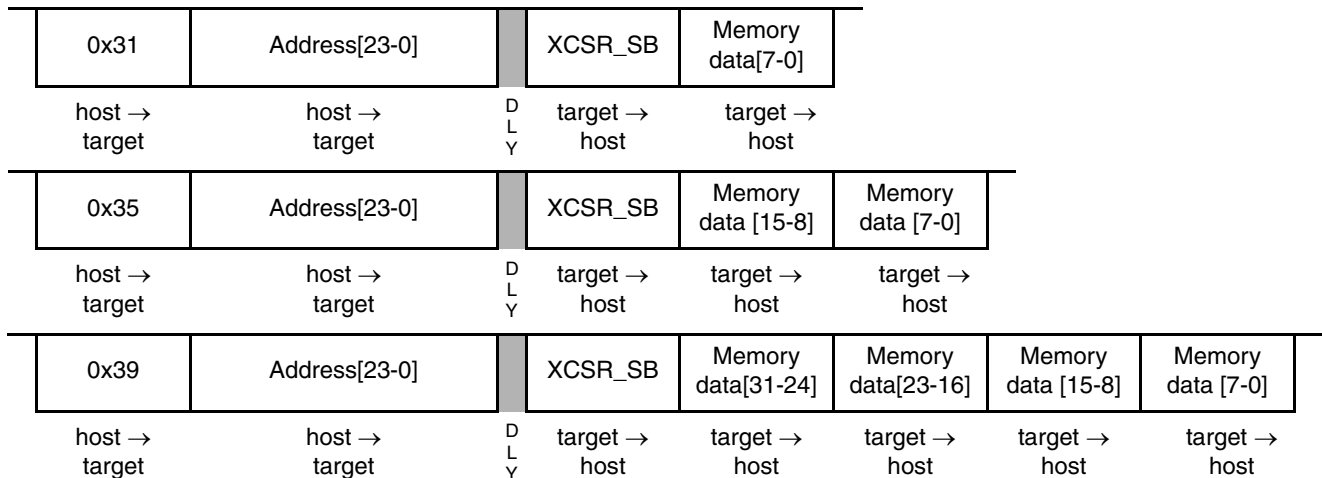
Non-intrusive



#### READ\_MEM.sz\_WS

Read memory at the specified address with status

Non-intrusive



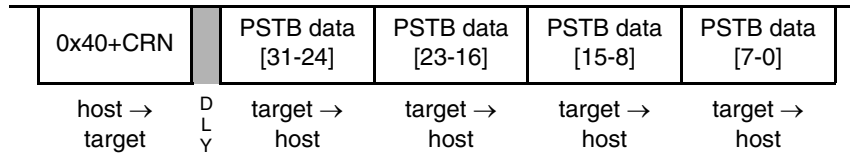
Read data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT,TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these accesses are on 0-modulo-size alignments. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31–24] is returned before the read data. XCSR\_SB reflects the state after the memory read was performed.

The examples show the READ\_MEM.B{ \_WS}, READ\_MEM.W{ \_WS} and READ\_MEM.L{ \_WS} commands.

#### 26.4.1.5.12 READ\_PSTB

Read PST trace buffer at the specified address

Non-intrusive

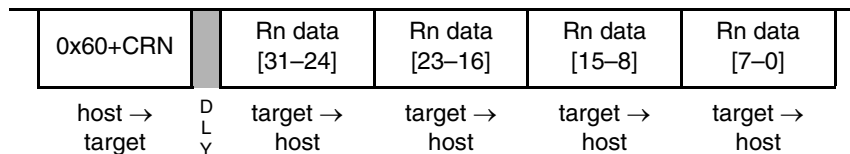


Read 32 bits of captured PST/DDATA values from the trace buffer at the specified address. The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. See [Figure 26-16](#) for an illustration of how the buffer entries are packed.

#### 26.4.1.5.13 READ\_Rn

Read general-purpose CPU register

Active Background



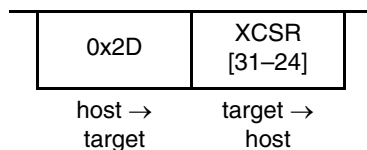
If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See [Table 26-24](#) for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

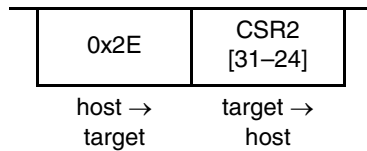
#### 26.4.1.5.14 READ\_XCSR\_BYTE

Read XCSR Status Byte

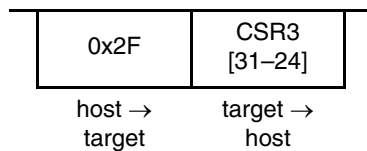
Always Available



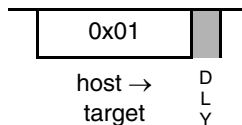
Read the special status byte of XCSR (XCSR[31-24]). This command can be executed in any mode.

**26.4.1.5.15 READ\_CSR2\_BYTE****Read CSR2 Status Byte****Always Available**

Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

**26.4.1.5.16 READ\_CSR3\_BYTE****Read CSR3 Status Byte****Always Available**

Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

**26.4.1.5.17 SYNC\_PC****Synchronize PC to PST/DDATA Signals****Non-intrusive**

Capture the processor's current PC (program counter) and display it on the PST/DDATA signals. After the debug module receives the command, it sends a signal to the ColdFire core that the current PC must be displayed. The core responds by forcing an instruction fetch to the next PC with the address being captured by the DDATA logic. The DDATA logic captures a 2- or 3-byte instruction address, based on CSR[9]. If CSR[9] is cleared, then a 2-byte address is captured, else a 3-byte address is captured. The specific sequence of PST and DDATA values is defined as:

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, generating a PST = 0x5 value indicating a taken branch. DDATA captures the instruction address corresponding to the PC. DDATA generates a PST marker signalling a 2- or 3-byte address as defined by CSR[9] (CSR[9] = 0, 2-byte; CSR[9] = 1, 3-byte) and displays the captured PC address.

This command can be used to provide a PC synchronization point between the core's execution and the application code in the PST trace buffer. It can also be used to dynamically access the PC for performance

monitoring as the execution of this command is considerably less obtrusive to the real-time operation of an application than a BACKGROUND/read-PC/GO command sequence.

#### 26.4.1.5.18 WRITE\_CREG

Write CPU control register				Active Background	
0xC0+CRN	CREG data [31–24]	CREG data [23–16]	CREG data [15–8]	CREG data [7–0]	
host → target	host → target	host → target	host → target	host → target	D L Y

If the processor is halted, this command writes the 32-bit operand to the selected control register. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER\_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 26-24](#) for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 26.4.1.5.19 WRITE\_DREG

Write debug control register				Non-intrusive	
0x80+CRN	DREG data [31–24]	DREG data [23–16]	DREG data [15–8]	DREG data [7–0]	
host → target	host → target	host → target	host → target	host → target	D L Y

This command writes the 32-bit operand to the selected debug control register. This grouping includes all the debug control registers ( $\{X\}CSR_n$ , BAAR, AATR, TDR, PBR<sub>n</sub>, PBMR, AB<sub>x</sub>R, DBR, DBMR). Accesses to debug control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See [Table 26-4](#) for CRN details.

#### NOTE

When writing XCSR, CSR2, or CSR3, WRITE\_DREG only writes bits 23–0. The upper byte of these debug registers is only written with the special WRITE\_XCSR\_BYTE, WRITE\_CSR2\_BYTE, and WRITE\_CSR3\_BYTE commands.

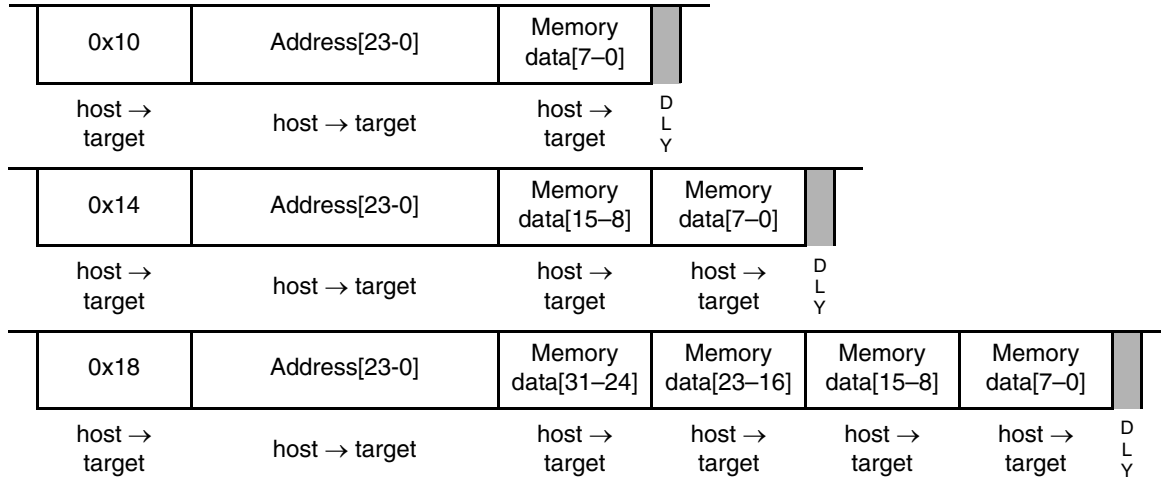


## 26.4.1.5.20 WRITE\_MEM.sz, WRITE\_MEM.sz\_WS

## WRITE\_MEM.sz

Write memory at the specified address

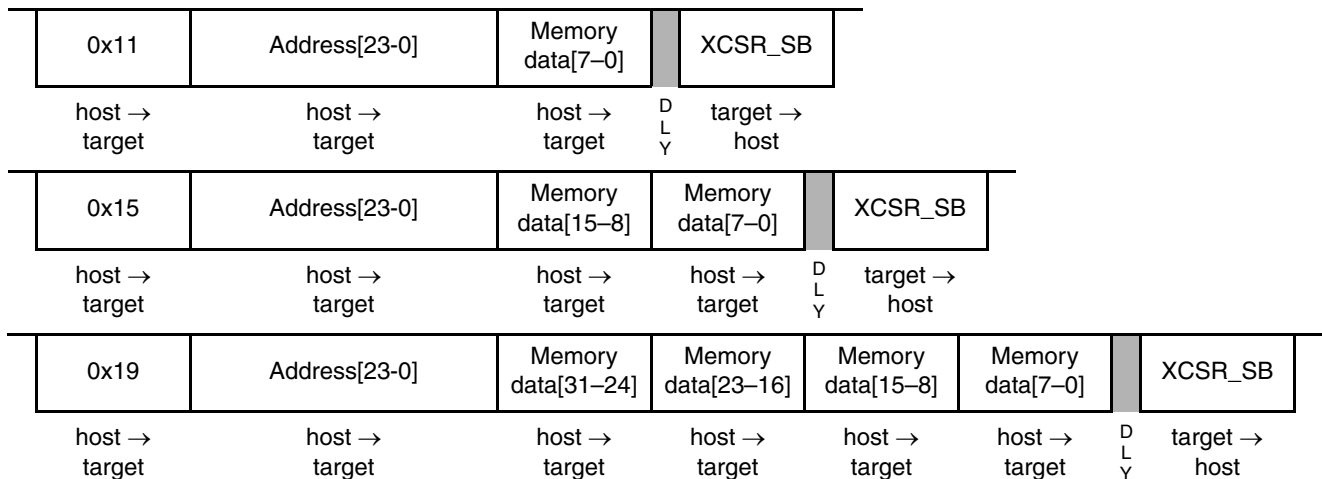
Non-intrusive



## WRITE\_MEM.sz\_WS

Write memory at the specified address with status

Non-intrusive



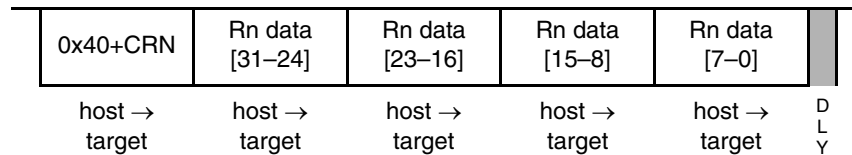
Write data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT, TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these accesses are on 0-modulo-size alignments. If the with-status option is specified, the core status byte (XCSR\_SB) contained in XCSR[31-24] is returned after the read data. XCSR\_SB reflects the state after the memory write was performed.

The examples show the WRITE\_MEM.B{ \_WS }, WRITE\_MEM.W{ \_WS }, and WRITE\_MEM.L{ \_WS } commands.

### 26.4.1.5.21 WRITE\_Rn

Write general-purpose CPU register

Active Background



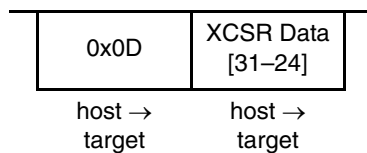
If the processor is halted, this command writes the 32-bit operand to the selected CPU general-purpose register (An, Dn). See [Table 26-24](#) for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

### 26.4.1.5.22 WRITE\_XCSR\_BYTE

Write XCSR Status Byte

Always Available

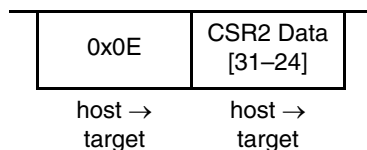


Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

### 26.4.1.5.23 WRITE\_CSR2\_BYTE

Write CSR2 Status Byte

Always Available

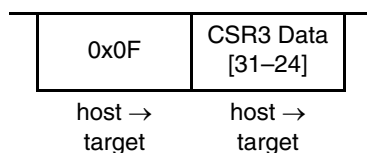


Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

### 26.4.1.5.24 WRITE\_CSR3\_BYTE

Write CSR3 Status Byte

Always Available



Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

### 26.4.1.6 Serial Interface Hardware Handshake Protocol

BDC commands that require CPU execution are ultimately treated at the core clock rate. Because the BDC clock source can be asynchronous relative to the bus frequency when CLKS<sub>W</sub> is cleared, it is necessary to provide a handshake protocol so the host can determine when an issued command is executed by the CPU. This section describes this protocol.

The hardware handshake protocol signals to the host controller when an issued command was successfully executed by the target. This protocol is implemented by a low pulse (16 BDC clock cycles) followed by a brief speedup pulse on the BKGD pin, generated by the target MCU when a command, issued by the host, has been successfully executed. See Figure 26-21. This pulse is referred to as the ACK pulse. After the ACK pulse is finished, the host can start the data-read portion of the command if the last-issued command was a read command, or start a new command if the last command was a write command or a control command (BACKGROUND, GO, NOP, SYNC\_PC). The ACK pulse is not issued earlier than 32 BDC clock cycles after the BDC command was issued. The end of the BDC command is assumed to be the 16th BDC clock cycle of the last bit. This minimum delay assures enough time for the host to recognize the ACK pulse. There is no upper limit for the delay between the command and the related ACK pulse, because the command execution depends on the CPU bus frequency, which in some cases could be slow compared to the serial communication rate. This protocol allows great flexibility for pod designers, because it does not rely on any accurate time measurement or short response time to any event in the serial communication.

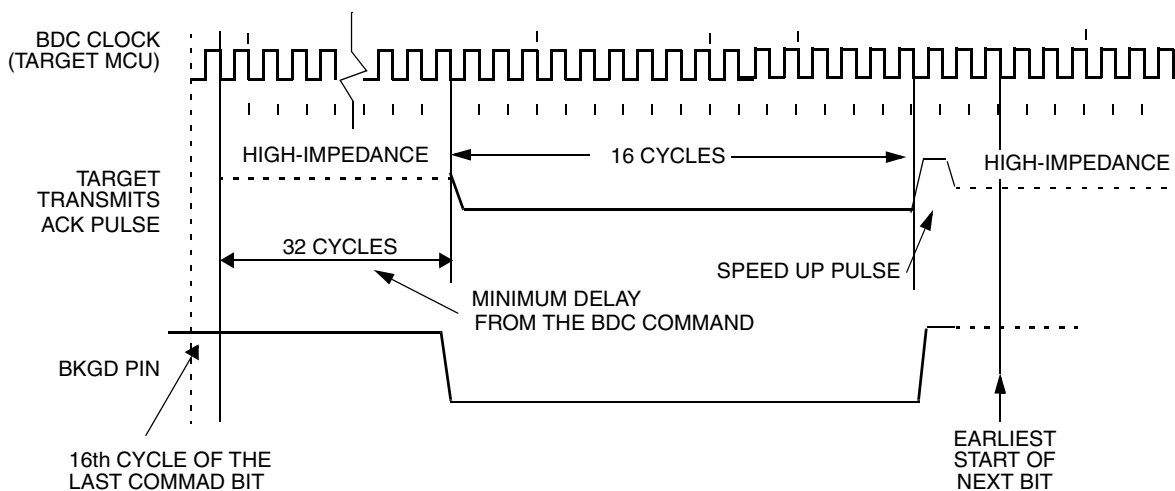


Figure 26-21. Target Acknowledge Pulse (ACK)

#### NOTE

If the ACK pulse was issued by the target, the host assumes the previous command was executed. If the CPU enters a stop mode prior to executing a non-intrusive command, the command is discarded and the ACK pulse is not issued. After entering a stop mode, the BDC command is no longer pending and the XCSR[CSTAT] value of 001 is kept until the next command is successfully executed.

Figure 26-22 shows the ACK handshake protocol in a command level timing diagram. A READ\_MEM.B command is used as an example:

1. The 8-bit command code is sent by the host, followed by the address of the memory location to be read.
2. The target BDC decodes the command and sends it to the CPU.
3. Upon receiving the BDC command request, the CPU schedules a execution slot for the command.
4. The CPU temporarily stalls the instruction stream at the scheduled point, executes the READ\_MEM.B command and then continues.

This process is referred to as cycle stealing. The READ\_MEM.B appears as a single-cycle operation to the processor, even though the pipelined nature of the Operand Execution Pipeline requires multiple CPU clock cycles for it to actually complete. After that, the debug module tracks the execution of the READ\_MEM.b command as the processor resumes the normal flow of the application program. After detecting the READ\_MEM.B command is done, the BDC issues an ACK pulse to the host controller, indicating that the addressed byte is ready to be retrieved. After detecting the ACK pulse, the host initiates the data-read portion of the command.

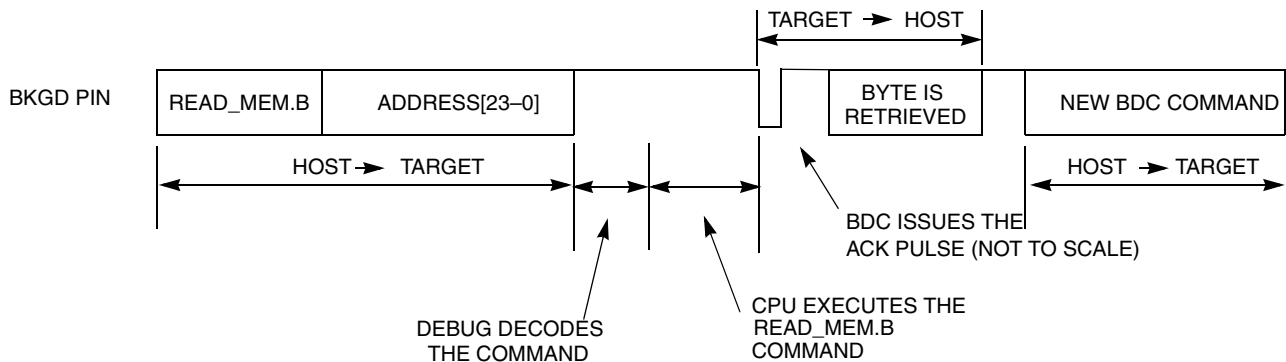


Figure 26-22. Handshake Protocol at Command Level

Unlike a normal bit transfer, where the host initiates the transmission by issuing a negative edge in the BKGD pin, the serial interface ACK handshake pulse is initiated by the target MCU. The hardware handshake protocol in Figure 26-22 specifies the timing when the BKGD pin is being driven, so the host should follow these timing relationships to avoid the risks of an electrical conflict at the BKGD pin.

The ACK handshake protocol does not support nested ACK pulses. If a BDC command is not acknowledged by an ACK pulse, the host first needs to abort the pending command before issuing a new BDC command. When the CPU enters a stop mode at about the same time the host issues a command that requires CPU execution, the target discards the incoming command. Therefore, the command is not acknowledged by the target, meaning that the ACK pulse is not issued in this case. After a certain time, the host could decide to abort the ACK protocol to allow a new command. Therefore, the protocol provides a mechanism where a command (a pending ACK) could be aborted. Unlike a regular BDC command, the ACK pulse does not provide a timeout. In the case of a STOP instruction where the ACK is prevented from being issued, it would remain pending indefinitely if not aborted. See the handshake abort procedure described in Section 26.4.1.7, “Hardware Handshake Abort Procedure.”

### 26.4.1.7 Hardware Handshake Abort Procedure

The abort procedure is based on the SYNC command. To abort a command that has not responded with an ACK pulse, the host controller generates a sync request (by driving BKGD low for at least 128 serial clock cycles and then driving it high for one serial clock cycle as a speedup pulse). By detecting this long low pulse on the BKGD pin, the target executes the sync protocol (see [Section 26.4.1.5.1, “SYNC”](#)), and assumes that the pending command and therefore the related ACK pulse, are being aborted. Therefore, after the sync protocol completes, the host is free to issue new BDC commands.

Because the host knows the target BDC clock frequency, the SYNC command does not need to consider the lowest possible target frequency. In this case, the host could issue a SYNC close to the 128 serial clock cycles length, providing a small overhead on the pulse length to assure the sync pulse is not misinterpreted by the target.

It is important to notice that any issued BDC command that requires CPU execution is scheduled for execution by the pipeline based on the dynamic state of the machine, provided the processor does not enter any of the stop modes. If the host aborts a command by sending the sync pulse, it should then read XCSR[CSTAT] after the sync response is issued by the target, checking for CSTAT cleared, before attempting to send any new command that requires CPU execution. This prevents the new command from being discarded at the debug/CPU interface, due to the pending command being executed by the CPU. Any new command should be issued only after XCSR[CSTAT] is cleared.

There are multiple reasons that could cause a command to take too long to execute, measured in terms of the serial communication rate: The BDC clock frequency could be much faster than the CPU clock frequency, or the CPU could be accessing a slow memory, which would cause pipeline stall cycles to occur. All commands referencing the CPU registers or memory require access to the processor’s local bus to complete. If the processor is executing a tight loop contained within a single aligned longword, the processor may never successfully grant the internal bus to the debug command. For example:

```

        align    4
label1:  nop
        bra.b   label1
or

```

```

        align    4
label2:  bra.w   label2

```

These two examples of tight loops exhibit the BDM lockout behavior. If the loop spans across two longwords, there are no issues, so the recommended construct is:

```

        align    4
label3:  bra.l   label3

```

The hardware handshake protocol is appropriate for these situations, but the host could also decide to use the software handshake protocol instead. In this case, if XCSR[CSTAT] is 001, there is a BDC command pending at the debug/CPU interface. The host controller should monitor XCSR[CSTAT] and wait until it is 000 to be able to issue a new command that requires CPU execution. However, if the XCSR[CSTAT] is 1xx, the host should assume the last command failed to execute. To recover from this condition, the following sequence is suggested:

1. Issue a SYNC command to reset the BDC communication channel.
2. The host issues a BDM NOP command.

3. The host reads the channel status using a READ\_XCSR\_BYTE command.
4. If XCSR[CSTAT] is 000
  - then the status is okay; proceed
  - else
    - Halt the CPU using a BDM BACKGROUND command
    - Repeat steps 1,2,3
    - If XCSR[CSTAT] is 000, then proceed, else reset the device

Figure 26-23 shows a SYNC command aborting a READ\_MEM.B. After the command is aborted, a new command could be issued by the host.

### NOTE

Figure 26-23 signal timing is not drawn to scale.

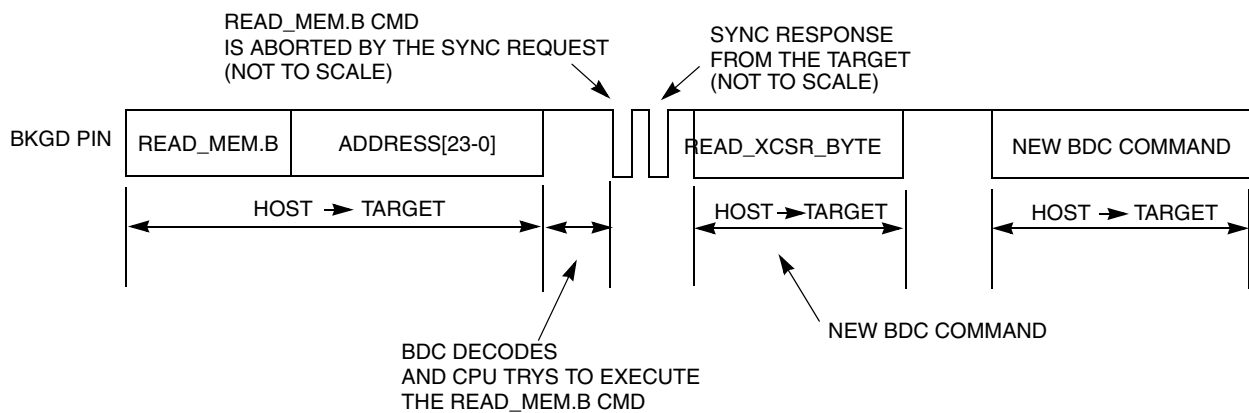
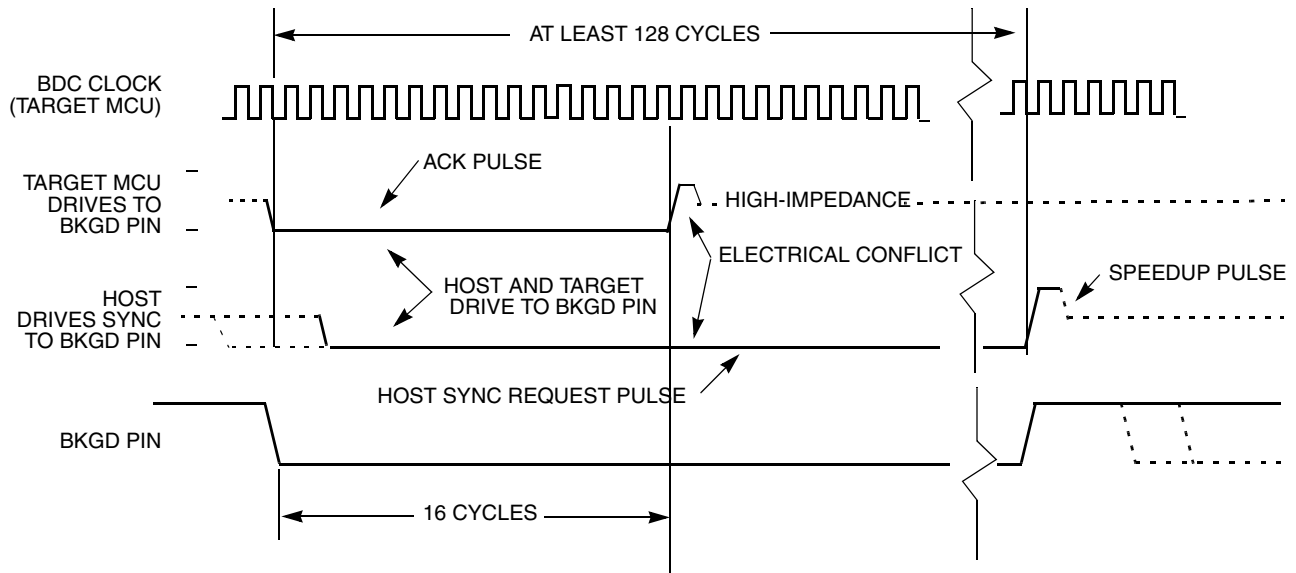


Figure 26-23. ACK Abort Procedure at the Command Level

Figure 26-24 shows a conflict between the ACK pulse and the sync request pulse. This conflict could occur if a pod device is connected to the target BKGD pin and the target is already executing a BDC command. Consider that the target CPU is executing a pending BDC command at the exact moment the pod is being connected to the BKGD pin. In this case, an ACK pulse is issued at the same time as the SYNC command. In this case there is an electrical conflict between the ACK speedup pulse and the sync pulse. Because this is not a probable situation, the protocol does not prevent this conflict from happening.



**Figure 26-24. ACK Pulse and SYNC Request Conflict**

The hardware handshake protocol is enabled by the `ACK_ENABLE` command and disabled by the `ACK_DISABLE` command. It also allows for pod devices to choose between the hardware handshake protocol or the software protocol that monitors the XCSR status byte. The `ACK_ENABLE` and `ACK_DISABLE` commands are:

- `ACK_ENABLE` — Enables the hardware handshake protocol. The target issues the ACK pulse when a CPU command is executed. The `ACK_ENABLE` command itself also has the ACK pulse as a response.
- `ACK_DISABLE` — Disables the ACK pulse protocol. In this case, the host should verify the state of `XCSR[CSTAT]` to evaluate if there are pending commands and to check if the CPU's operating state has changed to or from active background mode via `XCSR[31–30]`.

The default state of the protocol, after reset, is hardware handshake protocol disabled.

The commands that do not require CPU execution, or that have the status register included in the retrieved bit stream, do not perform the hardware handshake protocol. Therefore, the target does not respond with an ACK pulse for those commands even if the hardware protocol is enabled. Conversely, only commands that require CPU execution and do not include the status byte perform the hardware handshake protocol. See the third column in [Table 26-25](#) for the complete enumeration of this function.

An exception is the `ACK_ENABLE` command, which does not require CPU execution but responds with the ACK pulse. This feature can be used by the host to evaluate if the target supports the hardware handshake protocol. If an ACK pulse is issued in response to this command, the host knows that the target supports the hardware handshake protocol. If the target does not support the hardware handshake protocol the ACK pulse is not issued. In this case, the `ACK_ENABLE` command is ignored by the target, because it is not recognized as a valid command.

## 26.4.2 Real-Time Debug Support

The ColdFire family supports 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 with minimal effect on real-time operation.

### NOTE

The details regarding real-time debug support will be supplied at a later time.

## 26.4.3 Real-Time Trace Support with the Visibility Bus Enabled (CSR[VBD] = 0)

Real-time trace, which defines the dynamic execution path and is also known as instruction trace, is a fundamental debug function. The ColdFire solution includes 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 the 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 26.4.4.1, “Begin Execution of Taken Branch \(PST = 0x05\)”](#). 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 26-26](#) shows the encoding of these signals.

**Table 26-26. CF1 Debug Processor Status Encodings with VBus Enabled**

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.



**Table 26-26. CF1 Debug Processor Status Encodings with VBus Enabled (continued)**

PST[3:0]	Definition
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[BTB] also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See <a href="#">Section 26.4.3.1, “Begin Execution of Taken Branch (PST = 0x5)”</a> . 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. 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
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 <a href="#">Section 26.4.1.1, “CPU Halt”</a> .

### 26.4.3.1 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[BTB] also controls the number of address bytes displayed, which is indicated by the PST marker value immediately preceding the DDATA nibble in the PSTB 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 clock cycles:

1. Drive PST=0x5 to identify that a taken branch is executed.
2. Using the PST pins, optionally signal the target address to be displayed sequentially on the DDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.
3. The new target address is optionally available on subsequent cycles using the DDATA port. The number of bytes of the target address displayed on this port is configurable (2, 3, or 4 bytes, where the encoding is 0x9, 0xA, and 0xB, respectively).

Another example of a variant branch instruction would be a JMP (A0) instruction. Figure 26-26 shows the PST and DDATA outputs that indicate a JMP (A0) execution, assuming the CSR was programmed to display the lower two bytes of an address.

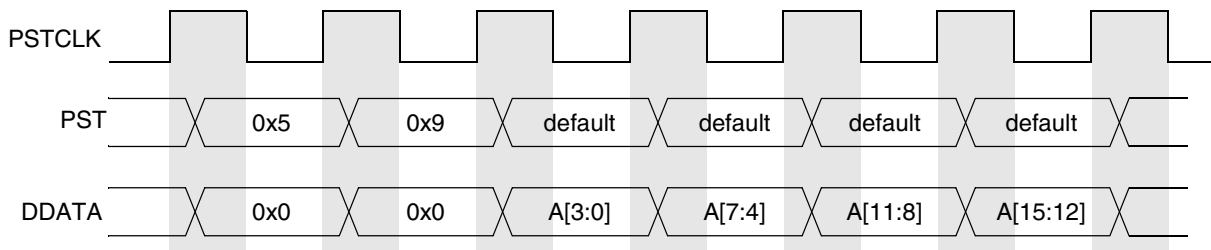


Figure 26-25. Example JMP Instruction Output on PST/DDATA

PST of 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Thus, the subsequent four nibbles of DDATA display the lower two bytes of address register A0, right-shifted by 1, 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.

#### 26.4.4 Trace Support With the Visibility Bus Disabled (CSR[VBD] = 1)

For the baseline V1 ColdFire core and its single debug signal, support for trace functionality is completely redefined. The V1 solution provides an on-chip PST/DDATA trace buffer (known as the PSTB) to record the stream of PST and DDATA values.

As a review, the classic ColdFire debug architecture supports real-time trace via the PST/DDATA output signals. For this functionality, the following apply:

- One (or more) PST value is generated for each executed instruction
- Branch target instruction address information is displayed on all non-PC-relative change-of-flow instructions, where the user selects a programmable number of bytes of target address
  - Displayed information includes PST marker plus target instruction address as DDATA
  - Captured address creates the appropriate number of DDATA entries, each with 4 bits of address
- Optional data trace capabilities are provided for accesses mapped to the slave peripheral bus
  - Displayed information includes PST marker plus captured operand value as DDATA

— Captured operand creates the appropriate number of DDATA entries, each with 4 bits of data

The resulting PST/DDATA output stream, with the application program memory image, provides an instruction-by-instruction dynamic trace of the execution path. For this device, the visibility bus optionally supports this type of real-time trace as explained in [Section 26.4.3, “Real-Time Trace Support with the Visibility Bus Enabled \(CSR\[VBD\] = 0\).”](#)

Even with the application of a PST trace buffer, problems associated with the PST bandwidth and associated fill rate of the buffer remain. Given that there is one (or more) PST entry per instruction, the PSTB would fill rapidly without some type of data compression.

Consider the following example to illustrate the PST compression algorithm. Most sequential instructions generate a single PST = 1 value. Without compression, the execution of ten sequential instructions generates a stream of ten PST = 1 values. With PST compression, the reporting of any PST = 1 value is delayed so that consecutive PST = 1 values can be accumulated. When a PST ≠ 1 value is reported, the maximum accumulation count is reached, or a debug data value is captured, a single accumulated PST value is generated. Returning to the example with compression enabled, the execution of ten sequential instructions generates a single PST value indicating ten sequential instructions have been executed.

This technique has proven to be effective at significantly reducing the average PST entries per instruction and PST entries per machine cycle. The application of this compression technique makes the application of a useful PST trace buffer for the V1 ColdFire core realizable. The resulting 5-bit PST definitions are shown in [Table 26-27](#).

**Table 26-27. CF1 Debug Processor Status Encodings with VBus Disabled**

PST[4:0]	Definition
0x00	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more processor clock cycles, subsequent clock cycles are indicated by driving PST with this encoding.
0x01	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.
0x02	Reserved
0x03	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x04	Begin execution of PULSE and WDDATA instructions. PULSE defines triggers or markers 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 0x04 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. The number of captured data bytes depends on the WDDATA operand size.
0x05	Begin execution of taken branch or SYNC_PC BDM command. 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. This encoding also indicates that the SYNC_PC command has been processed.
0x06	Reserved
0x07	Begin execution of return from exception (RTE) instruction.

Table 26-27. CF1 Debug Processor Status Encodings with VBus Disabled (continued)

PST[4:0]	Definition
0x08–0x0B	Indicates the number of data bytes to be loaded into the PST trace buffer. The capturing of peripheral bus data references is controlled by CSR[DDC]. 0x08 Begin 1-byte data transfer on DDATA 0x09 Begin 2-byte data transfer on DDATA 0x0A Reserved 0x0B Begin 4-byte data transfer on DDATA
0x0C–0x0F	Indicates the number of address bytes to be loaded into the PST trace buffer. The capturing of branch target addresses is controlled by CSR[BTB]. 0x0C Reserved 0x0D Begin 2-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[16:1]) 0x0E Begin 3-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[23:1]) 0x0F Reserved
0x10–0x11	Reserved
0x12	Completed execution of 2 sequential instructions
0x13	Completed execution of 3 sequential instructions
0x14	Completed execution of 4 sequential instructions
0x15	Completed execution of 5 sequential instructions
0x16	Completed execution of 6 sequential instructions
0x17	Completed execution of 7 sequential instructions
0x18	Completed execution of 8 sequential instructions
0x19	Completed execution of 9 sequential instructions
0x1A	Completed execution of 10 sequential instructions
0x1B	This value signals there has been a change in the breakpoint trigger state machine. It appears as a single marker for each state change and is immediately followed by a DDATA value signaling the new breakpoint trigger state encoding. The DDATA breakpoint trigger state value is defined as $(0x20 + 2 \times \text{CSR}[\text{BSTAT}])$ : 0x20 No breakpoints enabled 0x22 Waiting for a level-1 breakpoint 0x24 Level-1 breakpoint triggered 0x2A Waiting for a level-2 breakpoint 0x2C Level-2 breakpoint triggered
0x1C	Exception processing. This value signals the processor has encountered an exception condition. Although this is a multi-cycle mode, there are only two PST = 0x1C values recorded before the mode value is suppressed.
0x1D	Emulator mode exception processing. This value signals the processor has encountered a debug interrupt or a properly-configured trace exception. Although this is a multi-cycle mode, there are only two PST = 0x1D values recorded before the mode value is suppressed.
0x1E	Processor is stopped. This value signals the processor has executed a STOP instruction. Although this is a multi-cycle mode because the ColdFire processor remains stopped until an interrupt or reset occurs, there are only two PST = 0x1E values recorded before the mode value is suppressed.
0x1F	Processor is halted. This value signals the processor has been halted. Although this is a multi-cycle mode because the ColdFire processor remains halted until a BDM go command is received or reset occurs, there are only two PST = 0x1F values recorded before the mode value is suppressed.

### 26.4.4.1 Begin Execution of Taken Branch (PST = 0x05)

The PST is 0x05 when a taken branch is executed. For some opcodes, a branch target address may be loaded into the trace buffer (PSTB) depending on the CSR settings. CSR also controls the number of address bytes loaded that is indicated by the PST marker value immediately preceding the DDATA entry in the PSTB that begins the address entries.

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 loads the PSTB as follows:

1. Load PST=0x05 to identify that a taken branch is executed.
2. Optionally load the marker for the target address capture. Encodings 0x0D or 0x0E identify the number of bytes loaded into the PSTB.
3. The new target address is optionally available in the PSTB. The number of bytes of the target address loaded is configurable (2 or 3 bytes, where the encoding is 0x0D and 0x0E, respectively).

Another example of a variant branch instruction is a JMP (A0) instruction. [Figure 26-26](#) shows the PSTB entries that indicate a JMP (A0) execution, assuming CSR[BTB] was programmed to display the lower two bytes of an address.

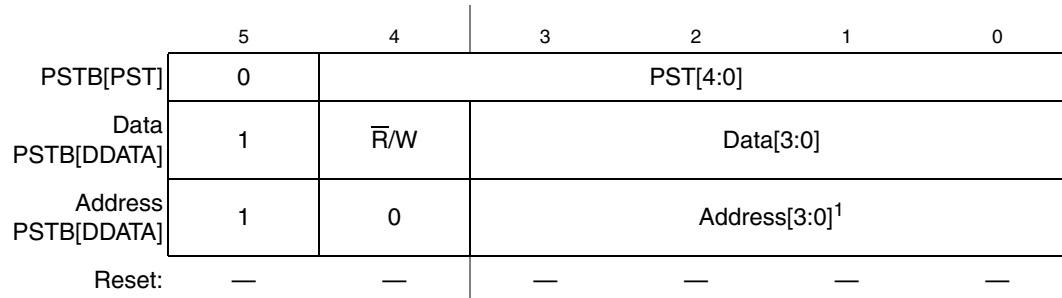
PST/DDATA Values	Description
0x05	Taken Branch
0x0D	2-byte Address Marker
{10, Address[4:1]}	Address >> 1
{10, Address[8:5]}	
{10, Address[12:9]}	
{10, Address[16:13]}	

**Figure 26-26. Example JMP Instruction Output in PSTB**

The PST of 0x05 indicates a taken branch and the marker value 0x0D indicates a 2-byte address. Therefore, the following entries display the lower two bytes of address register A0, right-shifted by 1, in least-to-most-significant nibble order. The next PST entry after the JMP instruction completes depends on the target instruction. See [Section 26.4.4.2, “PST Trace Buffer \(PSTB\) Entry Format,”](#) for entry descriptions explaining the 2-bit prefix before each address nibble.

### 26.4.4.2 PST Trace Buffer (PSTB) Entry Format

As PST and DDATA values are captured and loaded in the trace buffer, each entry is six bits in size therefore, the type of the entry can easily be determined when post-processing the PSTB. See [Figure 26-27](#).



<sup>1</sup> Depending on which nibble is displayed (as determined by CSR[9:8]), Address[3:0] sequentially (least-to-most-significant nibble order) displays four bits of the real CPU address [16:1] or [24:1].

**Figure 26-27. V1 PST/DDATA Trace Buffer Entry Format**

### 26.4.4.3 PST/DDATA Example

In this section, an example showing the behavior of the PST/DDATA functionality is detailed. Consider the following interrupt service routine that counts the interrupt, negates the IRQ, performs a software IACK, and then exits. This example is presented here because it exercises a considerable set of the PST/DDATA capabilities.

```

_isr:
01074: 46fc 2700      mov.w   &0x2700,%sr      # disable interrupts
01078: 2f08          mov.l   %a0,-(%sp)      # save a0
0107a: 2f00          mov.l   %d0,-(%sp)      # save d0
0107c: 302f 0008      mov.w   (8,%sp),%d0     # load format/vector word
01080: e488          lsr.l   &2,%d0         # align vector number
01082: 0280 0000 00ff  andi.l  &0xff,%d0       # isolate vector number
01088: 207c 0080 1400  mov.l   &int_count,%a0  # base of interrupt counters

_isr_entry1:
0108e: 52b0 0c00      addq.l  &1,(0,%a0,%d0.l*4) # count the interrupt
01092: 11c0 a021      mov.b   %d0,IGCR0+1.w   # negate the irq
01096: 1038 a020      mov.b   IGCR0.w,%d0     # force the write to complete
0109a: 4e71          nop                    # synchronize the pipelines
0109c: 71b8 ffe0      mvz.b   SWIACK.w,%d0    # software iack: pending irq?
010a0: 0c80 0000 0041  cmpi.l  %d0,&0x41       # level 7 or none pending?
010a6: 6f08          ble.b   _isr_exit      # yes, then exit
010a8: 52b9 0080 145c  addq.l  &1,swiack_count # increment the swiack count
010ae: 60de          bra.b   _isr_entry1    # continue at entry1

_isr_exit:
010b0: 201f          mov.l   (%sp)+,%d0     # restore d0
010b2: 205f          mov.l   (%sp)+,%a0     # restore a0
010b4: 4e73          rte                    # exit

```

This ISR executes mostly as straight-line code: there is a single conditional branch @ PC = 0x10A6, which is taken in this example. The following description includes the PST and DDATA values generated as this

code snippet executes. In this example, the CSR setting enables only the display of 2-byte branch addresses. Operand data captures are not enabled. The sequence begins with an interrupt exception:

```

interrupt exception occurs @ pc = 5432 while in user mode
# pst   = 1c, 1c, 05, 0d
# ddata = 2a, 23, 28, 20
#       trg_addr = 083a << 1
#       trg_addr = 1074

_isr:
01074: 46fc 2700      mov.w   &0x2700,%sr      # pst   = 01
01078: 2f08        mov.l   %a0,-(%sp)       # pst   = 01
0107a: 2f00        mov.l   %d0,-(%sp)       # pst   = 01
0107c: 302f 0008      mov.w   (8,%sp),%d0      # pst   = 01
01080: e488        lsr.l   &2,%d0           # pst   = 01
01082: 0280 0000 00ff  andi.l  &0xff,%d0        # pst   = 01
01088: 207c 0080 1400  mov.l   &int_count,%a0   # pst   = 01
0108e: 52b0 0c00      addq.l  &1,(0,%a0,%d0.l*4) # pst   = 01
01092: 11c0 a021      mov.b   %d0,IGCR0+1.w    # pst   = 01, 08
#       ddata = 30, 30
#       wdata.b = 0x00
01096: 1038 a020      mov.b   IGCR0.w,%d0      # pst   = 01, 08
#       ddata = 28, 21
#       rdata.b = 0x18
0109a: 4e71        nop                    # pst   = 01
0109c: 71b8 ffe0      mvz.b   SWIACK.w,%d0     # pst   = 01, 08
#       ddata = 20, 20
#       rdata.b = 0x00
010a0: 0c80 0000 0041  cmpi.l  %d0,&0x41        # pst   = 01
010a6: 6f08        ble.b   _isr_exit        # pst   = 05 (taken branch)
010b0: 201f        mov.l   (%sp)+,%d0       # pst   = 01
010b2: 205f        mov.l   (%sp)+,%a0       # pst   = 01
010b4: 4e73        rte                    # pst   = 07, 03, 05, 0d
#       ddata = 29, 21, 2a, 22
#       trg_addr = 2a19 << 1
#       trg_addr = 5432

```

As the PSTs are compressed, the resulting stream of 6-bit hexadecimal entries is loaded into consecutive locations in the PST trace buffer:

```

PSTB[*]= 1c, 1c, 05, 0d, // interrupt exception
         2a, 23, 28, 20, // branch target addr = 1074
         1a,           // 10 sequential insts
         13,           // 3 sequential insts
         05, 12,       // taken_branch + 2 sequential
         07, 03, 05, 0d, // rte, entry into user mode
         29, 21, 2a, 22 // branch target addr = 5432

```

Architectural studies on the compression algorithm determined an appropriate size for the PST trace buffer. Using a suite of ten MCU benchmarks, a 64-entry PSTB was found to capture an average window of time of 520 processor cycles with program trace using 2-byte addresses enabled.

### 26.4.4.4 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:

$$\text{PST} = 0x01, \{ \text{PST} = 0x0[89B], \text{DDATA} = \text{operand} \}$$

where the {...} definition is optional operand information defined by the setting of the CSR, and [...] indicates the presence of one value from the list.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x08, 0x09, or 0x0B} identifies the size and presence of valid data to follow in the PST trace buffer (PSTB) {1, 2, or 4 bytes, respectively}. Additionally, CSR[DDC] specifies whether operand data capture is enabled and what size. Also, for certain change-of-flow instructions, CSR[BTB] provides the capability to display the target instruction address in the PSTB (2 or 3 bytes) using a PST value of 0x0D or 0x0E, respectively.

#### 26.4.4.4.1 User Instruction Set

Table 26-28 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 26-28. PST/DDATA Specification for User-Mode Instructions**

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
add.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
adda.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
addi.l	#<data>,Dx	PST = 0x01
addq.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
addx.l	Dy,Dx	PST = 0x01
and.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
and.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
andi.l	#<data>,Dx	PST = 0x01
asl.l	{Dy,#<data>},Dx	PST = 0x01
asr.l	{Dy,#<data>},Dx	PST = 0x01
bcc.{b,w,l}		if taken, then PST = 0x05, else PST = 0x01
bchg.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bchg.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bclr.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}



Table 26-28. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
bclr.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bitrev.l	Dx	PST = 0x01
bra.{b,w,l}		PST = 0x05
bset.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bset.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bsr.{b,w,l}		PST = 0x05, {PST = 0x0B, DD = destination operand}
btst.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
btst.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
byterev.l	Dx	PST = 0x01
clr.b	<ea>x	PST = 0x01, {PST = 0x08, DD = destination operand}
clr.l	<ea>x	PST = 0x01, {PST = 0x0B, DD = destination operand}
clr.w	<ea>x	PST = 0x01, {PST = 0x09, DD = destination operand}
cmp.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
cmp.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
cmp.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
cmpa.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
cmpa.w	<ea>y,Ax	PST = 0x01, {0x09, source operand}
cmpi.b	#<data>,Dx	PST = 0x01
cmpi.l	#<data>,Dx	PST = 0x01
cmpi.w	#<data>,Dx	PST = 0x01
eor.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
eori.l	#<data>,Dx	PST = 0x01
ext.l	Dx	PST = 0x01
ext.w	Dx	PST = 0x01
extb.l	Dx	PST = 0x01
illegal		PST = 0x01 <sup>1</sup>
jmp	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address} <sup>2</sup>
jsr	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}, {PST = 0x0B, DD = destination operand} <sup>2</sup>
lea.l	<ea>y,Ax	PST = 0x01
link.w	Ay,#<displacement>	PST = 0x01, {PST = 0x0B, DD = destination operand}
lsl.l	{Dy,#<data>},Dx	PST = 0x01
lsr.l	{Dy,#<data>},Dx	PST = 0x01

Table 26-28. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
mov3q.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = destination operand}
move.b	<ea>y,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x01, {PST = 0x09, DD = source}, {PST = 0x09, DD = destination}
move.w	CCR,Dx	PST = 0x01
move.w	{Dy,#<data>},CCR	PST = 0x01
movea.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source}
movea.w	<ea>y,Ax	PST = 0x01, {PST = 0x09, DD = source}
movem.l	#list,<ea>x	PST = 0x01, {PST = 0x0B, DD = destination},...
movem.l	<ea>y,#list	PST = 0x01, {PST = 0x0B, DD = source},...
moveq.l	#<data>,Dx	PST = 0x01
muls.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mvs.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvs.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
mvz.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvz.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
neg.l	Dx	PST = 0x01
negx.l	Dx	PST = 0x01
nop		PST = 0x01
not.l	Dx	PST = 0x01
or.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
or.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
ori.l	#<data>,Dx	PST = 0x01
pea.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = destination operand}
pulse		PST = 0x04
rts		PST = 0x01, {PST = 0x0B, DD = source operand}, PST = 0x05, {PST = 0x0[DE], DD = target address}
sats.l	Dx	PST = 0x01
scc.b	Dx	PST = 0x01
sub.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}

Table 26-28. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
sub.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
suba.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
subi.l	#<data>,Dx	PST = 0x01
subq.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
subx.l	Dy,Dx	PST = 0x01
swap.w	Dx	PST = 0x01
tas.b	<ea>x	PST = 0x01, {0x08, source}, {0x08, destination}
tpf		PST = 0x01
tpf.l	#<data>	PST = 0x01
tpf.w	#<data>	PST = 0x01
trap	#<data>	PST = 0x01 <sup>1</sup>
tst.b	<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
tst.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = source operand}
tst.w	<ea>y	PST = 0x01, {PST = 0x09, DD = source operand}
unlk	Ax	PST = 0x01, {PST = 0x0B, DD = destination operand}
wddata.b	<ea>y	PST = 0x04, {PST = 0x08, DD = source operand}
wddata.l	<ea>y	PST = 0x04, {PST = 0x0B, DD = source operand}
wddata.w	<ea>y	PST = 0x04, {PST = 0x09, DD = source operand}

<sup>1</sup> During normal exception processing, the PSTB is loaded with two successive 0x1C entries 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 = 0x1C, 0x1C,
{PST = 0x0B,DD = destination},           // stack frame
{PST = 0x0B,DD = destination},           // stack frame
{PST = 0x0B,DD = source},                // vector read
PST = 0x05,{PST = 0x0[DE],DD = target}   // handler PC

```

A similar set of PST/DD values is generated in response to an emulator mode excetion. For these events (caused by a debug interrupt or properly-enabled trace exception), the initial PST values are 0x1D, 0x1D and the remaining sequence is equivalent to normal exception processing.

The PST/DDATA specification for the reset exception is shown below:

Exception Processing:

```

PST = 0x1C, 0x1C,
PST = 0x05,{PST = 0x0[DE],DD = target}   // initial 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 = 0x1C (or 0x1D) value is driven for two trace buffer entries.

- <sup>2</sup> 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).

#### 26.4.4.4.2 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 26-29](#).

**Table 26-29. PST/DDATA Specification for Supervisor-Mode Instructions**

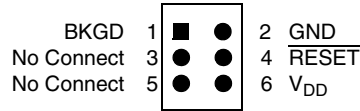
Instruction	Operand Syntax	PST/DDATA
halt		PST = 0x1F, PST = 0x1F
move.l	Ay,USP	PST = 0x01
move.l	USP,Ax	PST = 0x01
move.w	SR,Dx	PST = 0x01
move.w	{Dy,#<data>},SR	PST = 0x01, {PST = 0x03}
movec.l	Ry,Rc	PST = 0x01
rte		PST = 0x07, {PST = 0x0B, DD = source operand}, {PST = 0x03}, {PST = 0x0B, DD = source operand}, PST = 0x05, {PST = 0x0[DE], DD = target address}
stldsr.w	#imm	PST = 0x01, {PST = 0x0B, DD = destination operand, PST = 0x03}
stop	#<data>	PST = 0x1E, PST = 0x1E
wdebug.l	<ea>y	PST = 0x01, {PST = 0x0B, DD = source, PST = 0x0B, DD = source}

The move-to-SR, STLDSR, and RTE instructions include an optional PST = 0x3 value, indicating an entry into user mode.

Similar to the exception processing mode, the stopped state (PST = 0x1E) and the halted state (PST = 0x1F) display this status for two entries when the ColdFire processor enters the given mode.

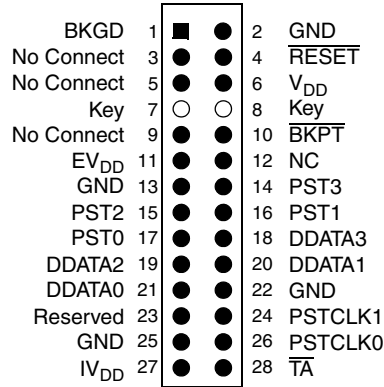
### 26.4.5 Freescale-Recommended BDM Pinout

Typically, a relatively simple interface pod is used to translate commands from a host computer into commands for the custom serial interface to the single-wire background debug system. Depending on the development tool vendor, this interface pod may use a standard RS-232 serial port, a parallel printer port, or some other type of communications such as a universal serial bus (USB) to communicate between the host PC and the pod. The pod typically connects to the target system with ground, the BKGD pin,  $\overline{\text{RESET}}$ , and sometimes  $V_{DD}$ . An open-drain connection to reset allows the host to force a target system reset, useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes  $V_{DD}$  can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.



**Figure 26-28. Recommended BDM Connector**

For applications using the VBus capabilities, it is recommended to use a merged version of the 1-pin BDM connector and the classic ColdFire 26-pin BDM connector, as shown in [Figure 26-29](#).



**Figure 26-29. Recommended VBus BDM Connector**

# Appendix A

## Revision History

This appendix lists major changes between versions of the MCF51AG128RM document.

### A.1 Changes Between Rev. 2 and Rev. 3

Table A-1. MCF51AG128RM Rev. 2 to Rev. 3 Changes

Chapter	Description
Device Overview	In <a href="#">Figure 1-3</a> , OSCOUT clock output of the XOSC is connected to RTC.
Modes of Operation	In <a href="#">Table 3-3</a> , modified the state of Crystal Oscillator in stop2 and stop3 modes.
Memory	<ul style="list-style-type: none"> <li>Corrected <a href="#">Table 4-6</a> and the description under this table to describe IFR.</li> <li>In <a href="#">Section 4.4.1</a>, “Features,” corrected the sector size of flash to 1024 byte.</li> </ul>
Resets, Interrupts, and General System Control	In <a href="#">Section 5.8.17</a> , “System Pin Positioning Register (SPINPS),” corrected SPINPS bit 0 and bit 1 definitions.
Parallel Input/Output Control	<ul style="list-style-type: none"> <li>In <a href="#">Section 6.2.3.6</a>, “PORTx Drive Strength Control Register (PTxDS),” corrected the description of PTxDS[7:0] bits</li> <li>In <a href="#">Section 6.3.3.5</a>, “Pin Interrupt Initialization,” added NOP instructions step between stop4 and stop5.</li> </ul>
Analog-to-Digital Converter (S08ADC12V3)	In <a href="#">Section 10.3.2</a> , “Status and Control Register 2 (ADCSC2),” changed bit 2 to writable as write is not reserved.
Cyclic Redundancy Check (S08CRCV3)	In <a href="#">Section 12.4.2</a> , “Programming Model Extension for CF1Core,” offset 0x2–0x3 is not described as reserved as it is used for TRANSPOSE register.
Generation 2008 Watchdog Timer	In <a href="#">Section 17.6.2.2</a> , “Watchdog Control and Status Register Low (WDOG_ST_CTRL_L)” for bits [14:0], added that the writes to these bits are not recommended.
External Watchdog Monitor (EWM)	In <a href="#">Section 18.4.2</a> , “EWM Compare High Register (EWMxCMPH),” added a note to clarify EWMCMPH register valid values.

## A.2 Changes Between Rev. 3 and Rev. 4

Table A-2. MCF51AG128RM Rev. 3 to Rev. 4 Changes

Chapter	Description
Device Overview	In <a href="#">Table 1-1</a> and <a href="#">Table 1-2</a> , added 48-pin LQFP package information.
Pins and Connections	In <a href="#">Table 2-1</a> , added 48-pin LQFP package information. Added <a href="#">Table 2-3</a> to show the 48-pin LQFP pin assignments.
Memory	<ul style="list-style-type: none"> <li>In <a href="#">Section 4.4.2.6</a>, “Flash Command Register (FCMD),” updated FCMD bit description.</li> </ul>
Resets, Interrupts, and General System Control	In SOPT2 register, modified CME bit description. Updated SPMSC2[PPDC] bitfield description.
FlexTimer Module (S08FTMV3)	Removed, Section, “Input Capture with FFCLK.”
Serial Peripheral Interface (S08SPIV6)	Updated <a href="#">Section 23.4.4.1</a> , “Transmit by DMA” section and added comments for receive overrun and empty transmit operation.
Version 1 ColdFire Debug (CF1_DEBUG)	Updated Note in <a href="#">Section 26.3.8</a> , “Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)” and <a href="#">Section 26.3.9</a> , “Address Breakpoint Registers (ABLR, ABHR)” to, “Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.”

## A.3 Changes Between Rev. 4 and Rev. 5

Table A-3. MCF51AG128RM Rev. 4 to Rev. 5 Changes

Chapter	Description
Device Overview	In <a href="#">Figure 1-3</a> , added “The ICSIRCLK is mapped on the IRCLK port of the RTC.” note. Standardized Generation 2008 Watchdog to Watchdog.
Pins and Connections	For 48-pin LQFP, updated the representation of PTG5/PTG6.
Modes of Operation	Updated <a href="#">Table 3-3</a> , for RTC operation in Stop3 and Stop4 modes.
Resets, Interrupts, and General System Control	Updated <a href="#">Section 5.3.4</a> , “Loss of Clock Reset (LOC).”
Serial Peripheral Interface (S08SPIV6)	Updated <a href="#">Section 23.4.4.1</a> , “Transmit by DMA.”
Internal Clock Source (S08ICSV3)	Updated ICS block diagram.
FlexTimer Module (S08FTMV3)	Updated block information for introduction, registers, and operating modes.
Watchdog Timer (WDOG)	Standardized Generation 2008 Watchdog to Watchdog.