MCF51AC256 ColdFire® Integrated Microcontroller Reference Manual

Devices Supported:

MCF51AC256A
MCF51AC256B
MCF51AC128A
MCF51AC128C

Document Number: MCF51AC256RM
Rev. 6
9/2011

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

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

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

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

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

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

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
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.

MCF51AC256RM
Rev. 6
9/2011

Chapter 1
Device Overview

Chapter 2
Pins and Connections

Chapter 3
Modes of Operation

Chapter 4
Memory

Chapter 5
Resets, Interrupts, and General System Control

Chapter 6
Parallel Input/Output Control

Chapter 7
ColdFire Core

Chapter 8
Analog Comparator (ACMPV3)

Chapter 9
Analog-to-Digital Converter (ADC12V1)

Chapter 10
Cyclic Redundancy Check Generator (CRCV2)

Chapter 11
FlexTimer Module (FTMV1)

Chapter 12
Inter-Integrated Circuit (IICV2)

Chapter 13
Interrupt Controller (CF1_INTC)

Chapter 14
Keyboard Interrupt (KBIV1)

Chapter 15
Freescale’s Controller Area Network (MSCANV1)

Chapter 16
Multipurpose Clock Generator (MCGV3)

Chapter 17
Rapid GPIO (RGPIO)

Chapter 18
Serial Communications Interface (SCIV4)

Chapter 19
8-Bit Serial Peripheral Interface (SPIV3)

Chapter 20
16-Bit Serial Peripheral Interface (SPI16V2)

Chapter 21
Timer/PWM Module (TPMV3)

Chapter 22
Version 1 ColdFire Debug (CF1_DEBUG)

Appendix A
Revision History

Chapter 1
Device Overview

1.1 MCF51AC256 Series Microcontrollers

1.1.1 Definition

The MCF51AC256 series microcontrollers are systems-on-chips (SoCs) that are based on the V1 ColdFire core. MCF51AC256 series microcontrollers

- 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 controller area network (CAN), FTM/TPM modules and CRC hardware accelerator
- Are the ideal upgrade for designs based on the MC9S08AC128 series microcontrollers

1.1.2 MCF51AC256 Series Package Availability

The device packages available for the MCF51AC256 series are summarized in Table 1-1.

Table 1-1. MCF51AC256 Series Package Availability

Packages	MCF51AC256A	MCF51AC256B	MCF51AC128A	MCF51AC128C
80-pin LQFP	Yes	Yes	Yes	Yes
64-pin LQFP	Yes	Yes	Yes	Yes
64-pin QFP	Yes	Yes	Yes	Yes
44-pin LQFP	No	Yes	No	Yes

1.1.3 MCF51AC256 Series Device Comparison

Table 1-2 compares the MCF51AC256 series microcontrollers.

Table 1-2. MCF51AC256 Series Device Comparison

<table>
<tr><th rowspan="2">Feature</th><th colspan="2">MCF51AC256A</th><th colspan="3">MCF51AC256B</th><th colspan="2">MCF51AC128A</th><th colspan="3">MCF51AC128C</th></tr>
<tr><th>80-pin</th><th>64-pin</th><th>80-pin</th><th>64-pin</th><th>44-pin</th><th>80-pin</th><th>64-pin</th><th>80-pin</th><th>64-pin</th><th>44-pin</th></tr>
<tr><td>Flash memory size (Kbytes)</td><td colspan="5">256</td><td colspan="5">128</td></tr>
<tr><td>RAM size (Kbytes)</td><td colspan="5">32</td><td colspan="5">32 or 16[1]</td></tr>
</table>

Table 1-2. MCF51AC256 Series Device Comparison (continued)

Feature	MCF51AC256A		MCF51AC256B			MCF51AC128A		MCF51AC128C		
	80-pin	64-pin	80-pin	64-pin	44-pin	80-pin	64-pin	80-pin	64-pin	44-pin
V1 ColdFire core with BDM (background debug module)	Yes									
ACMP1 (analog comparator)	Yes									
ACMP2 (analog comparator)	Yes		Yes		No	Yes				No
ADC (analog-to-digital converter) channels (12-bit)	24	20	24	20	9	24	20	24	20	9
CAN (controller area network)	Yes		No			Yes		No		
COP (computer operating properly)	Yes									
CRC (cyclic redundancy check)	Yes									
RTI	Yes									
DBG (debug)	Yes									
IIC1 (inter-integrated circuit)	Yes									
IRQ (interrupt request input)	Yes									
INTC (interrupt controller)	Yes									
KBI (keyboard interrupts)	Yes									
LVD (low-voltage detector)	Yes									
MCG (multipurpose clock generator)	Yes									
OSC (crystal oscillator)	Yes									
Port I/O[2]	69	54	69	54	36	69	54	69	54	36
RGPIO (rapid general-purpose I/O)	16				12	16				12
SCI1, SCI2 (serial communications interfaces)	Yes									
SPI1 (serial peripheral interface)	Yes									
SPI2 (serial peripheral interface)	Yes	No	Yes	No		Yes	No	Yes	No	
FTM1 (flexible timer module) channels	6				4	6				4
FTM2 channels	6	2	6	2	2	6	2	6	2	2
TPM3 (timer pulse-width modulator) channels	2									
VBUS (debug visibility bus)	Yes	No	Yes	No		Yes	No	Yes	No	

[1] The members of MCF51AC128A with CAN support have 32 KB RAM. The other members have 16 KB RAM.

[2] Up to 16 pins on Ports E and F are shared with the ColdFire Rapid GPIO module.

1.2 MCF51AC256 Series Block Diagram

1.2.1 Block Diagram

Figure 1-1 shows the connections between the MCF51AC256 series pins and functional units.

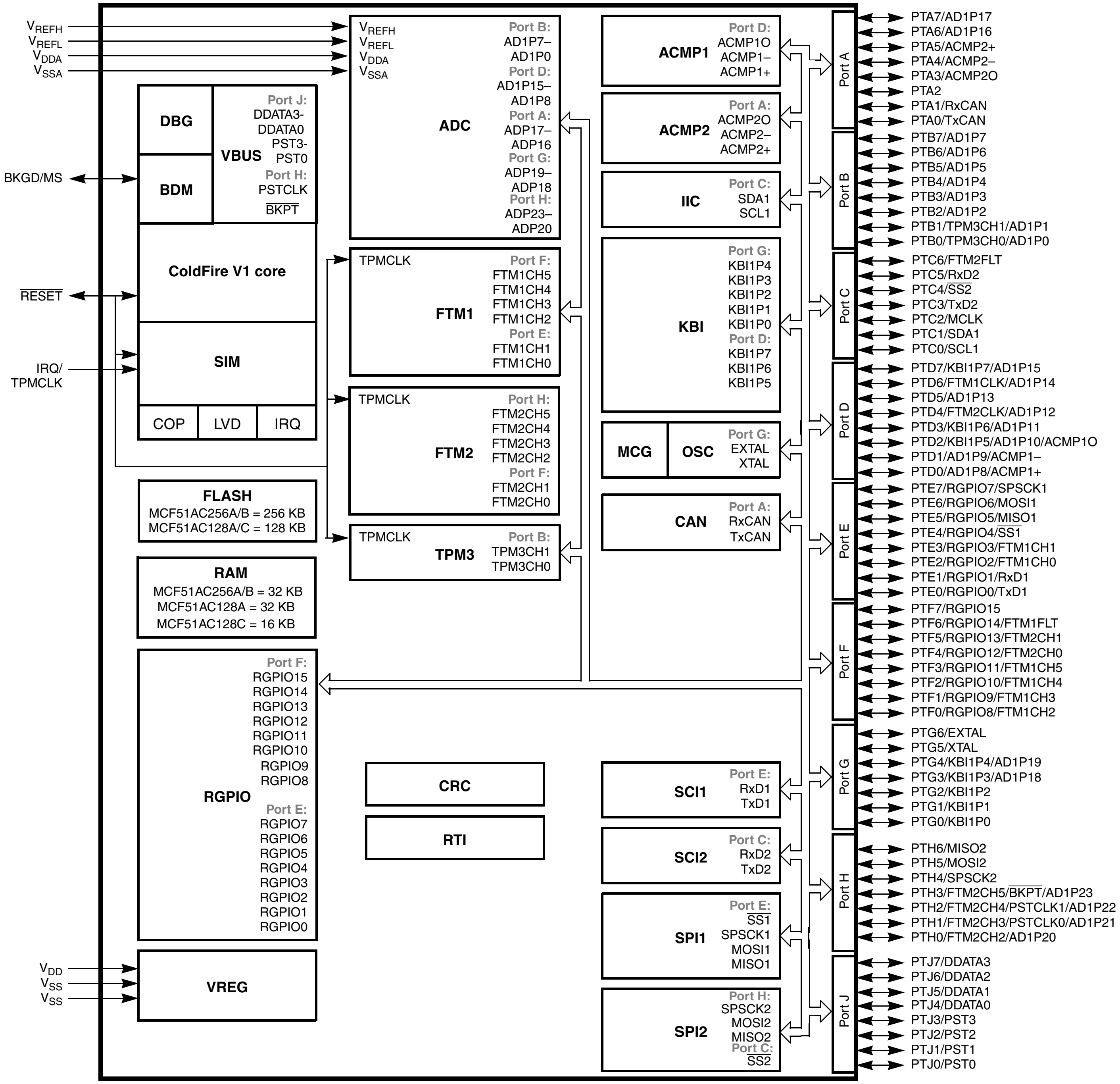


Figure 1-1. MCF51AC256 Series Block Diagram

1.2.2 Functional Units

Table 1-3 describes the functional units of the MCF51AC256 series microcontrollers.

Table 1-3. MCF51AC256 Series Functional Units

Functional Unit	Function
CF1 Core (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
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
COP (computer operating properly)	Monitors a countdown timer and generates a reset if the timer is not regularly reset by the software
LVD (low-voltage detect)	Monitors internal and external supply voltage levels, and generates a reset or interrupt when the voltages are too low
CF1_INTC (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)	Provides 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
ACMP1, ACMP2 (analog comparators)	Compares two analog inputs
IIC (inter-integrated circuit)	Supports standard IIC communications protocol
KBI (keyboard interrupt)	Provides pin interrupt capabilities
MCG (multipurpose clock generator)	Provides clocking options for the device, including a phase-locked loop (PLL) and 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 PLL or FLL
CAN (controller area network)	Supports standard CAN communications protocol
SCI1, SCI2 (serial communications interfaces)	Serial communications UARTs capable of supporting RS-232 and LIN protocols
SPI1 (8-bit serial peripheral interfaces)	Provides 8-bit 4-pin synchronous serial interface
SPI2 (16-bit serial peripheral interfaces)	Provides 16-bit 4-pin synchronous serial interface with FIFO

1.2.3 Module Versions

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

Table 1-4. Module Versions

Module		Version
Analog Comparator	(ACMP)	3
Analog-to-Digital Converter	(ADC12)	1
V1 ColdFire Core	(CF1CORE)	1
Cyclic Redundancy Check	(CRC)	2
Flexible Timer Pulse Width Modulator	(FTM)	1
General Purpose I/O	(GPIO)	2
Rapid GPIO	(RGPIO)	1
Interrupt Controller	(CF1_INTC)	1
Inter-Integrated Circuit	(IIC)	2
Keyboard Interrupt	(KBI)	1
MSCAN 2.0B	(msCAN)	1
Multi-purpose Clock Generator	(MCG)	3
Oscillator	(XOSC)	1
Serial Communications Interface	(SCI)	4
8-Bit Serial Peripheral Interface	(SPI)	3
16-Bit Serial Peripheral Interface	(SPI16)	2
Timer Pulse Width Modulator	(TPM)	3
Voltage Regulator	(PMC)	1

1.3 V1 ColdFire Core

The MCF51AC256 series devices contain a version of the V1 ColdFire core that is optimized for area of low power. This CPU implements ColdFire instruction set architecture revision C (ISA_C) plus adds support for real-time program trace capability:

- No hardware support for MAC/EMAC and DIV instructions[1]
- 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.”

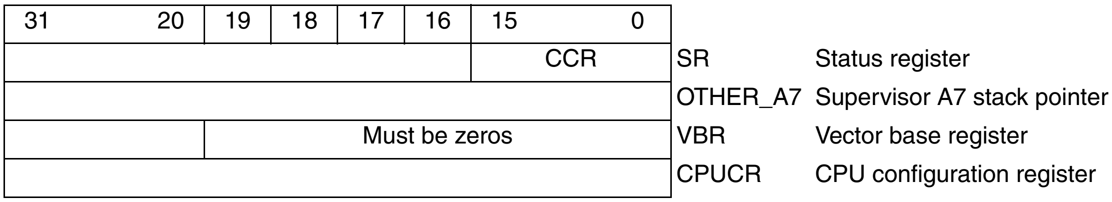
1.3.1 User Programming Model

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

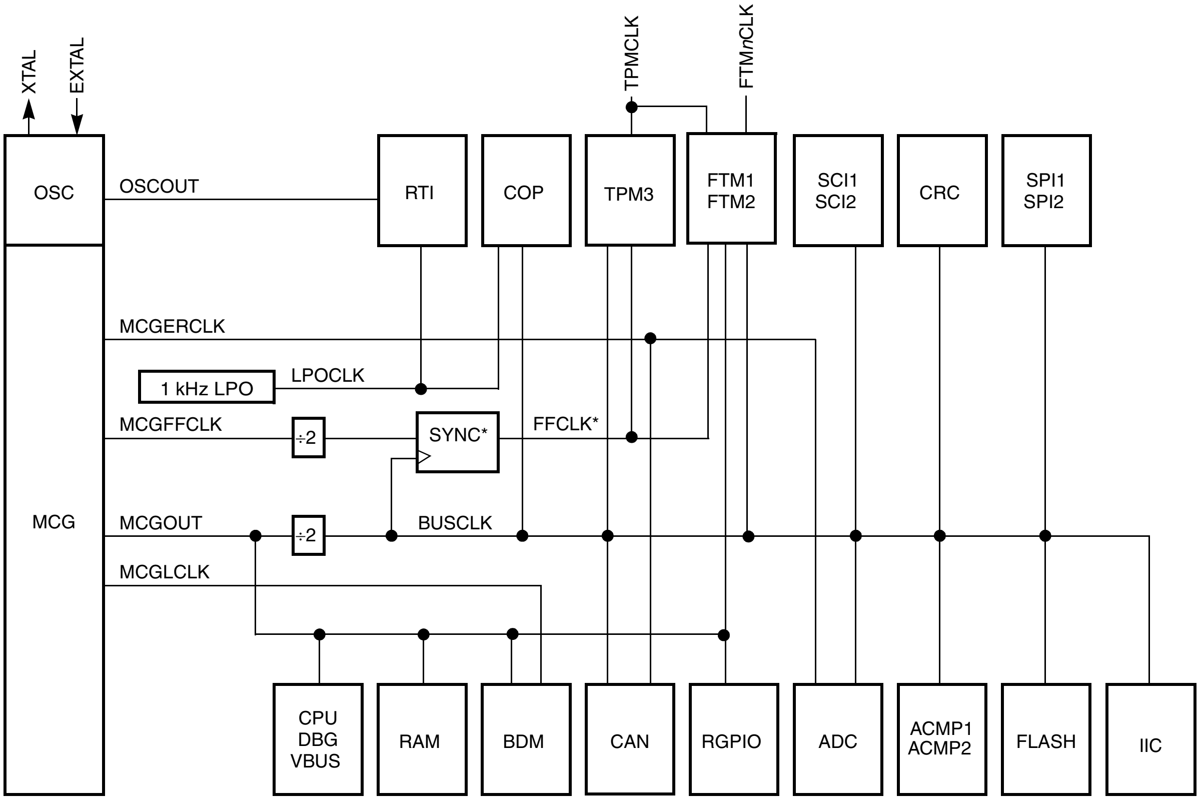
Figure 1-2. Supervisor Programming Model

[1] 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 MCG 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 (MCGOUT).



Note: The ADC has minimum and maximum frequency requirements. See the ADC chapter and the MCF51AC256 *Series Data Sheet*. Flash memory has frequency requirements for program and erase operations. See the MCF51AC256 *Series Data Sheet*.

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

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
OSCOUT	This is the direct output of the external oscillator module and can be selected as the real-time interrupt (RTI) clock source.
MCGOUT	This clock source is used as the CPU clock and is divided by two to generate the peripheral bus clock. Control bits in the MCG control registers determine which of three clock sources is connected: • Internal reference clock • External reference clock • Frequency-locked loop (FLL) or phase-locked loop (PLL) output This clock drives the CPU, debug, RAM, RGPIO and BDM directly and is divided by two to clock all peripherals (BUSCLK). The FTM modules can select MCGOUT as their clock input. See Chapter 16, “Multipurpose Clock Generator (MCGV3)” for details on configuring the MCGOUT clock.
MCGLCLK	This clock source is derived from the 10/20 MHz DCO (digitally controlled oscillator) or the 20/50 MHz phase-locked loop (PLL) of the MCG 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.
MCGERCLK	MCG External Reference Clock — This is the external reference clock and can be selected as the alternate clock for the ADC and CAN modules.
MCGIRCLK	MCG Internal Reference Clock—This is the internal reference clock and this clock signal is not used outside of the MSG.
MCGFFCLK	MCG Fixed-Frequency Clock — This generates the fixed frequency clock (FFCLK) after being synchronized to the bus clock. It can be selected as clock source for the TPM and FTM modules. The frequency of the FFCLK is determined by the settings of the MCG.
LPOCLK	Low-Power Oscillator Clock — This clock is generated from an internal low-power oscillator that is completely independent of the MCG module. The LPOCLK can be selected as the clock source to the COP and RTI.
TPMCLK	TPM Clock — An optional external clock source for the FTMs 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 Section 5.9.9, “System Options 2 (SOPT2) Register,” for details on using TPMCLK with the FTMs.
FTM*n*CLK	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.
ADACK	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.

1.4.3 Multi-Purpose Clock Generator (MCG)

The multi-purpose clock generator (MCG) module provides several clock source choices for the MCU. The module contains a frequency-locked loop (FLL) and a phase-locked loop (PLL) that are controllable by an internal or an external reference clock. This module can select either the FLL or PLL clock outputs, or either the internal or external reference clocks as a source for the MCU system clock.

Table 1-6 shows the modes of operation for the MCG.

Table 1-6. MCG Clock Modes

MCG Mode	Description
FEI	FLL Engaged Internal — The output of the FLL is used as the MCU clock source. The FLL uses the internal clock as a reference.
FEE	FLL Engaged External — The output of the FLL is used as the MCU clock source. The FLL uses the external clock as a reference.
FBI	FLL Bypassed Internal — The internal reference clock is used directly as the MCU clock source. The FLL is active and using the internal reference, but is not output.
FBE	FLL Bypassed External — The external reference clock is used directly as the MCU clock source. The FLL is active and using the external reference, but is not output.
PEE	PLL Engaged External — The output of the PLL is used as the MCU clock source. The PLL uses the external clock as a reference.
PBE	PLL Bypassed External — The external reference clock is used directly as the MCU clock source. The PLL is active and using the external reference, but is not output.
BLPI	Bypassed Low Power Internal — The internal reference clock is used directly as the MCU clock source. Both the PLL and FLL are disabled.
BLPE	Bypassed Low Power External — The external reference clock is used directly as the MCU clock source. Both the PLL and FLL are disabled.
Stop	MCG Stopped — MCG output is stopped. MCG is in low power mode.

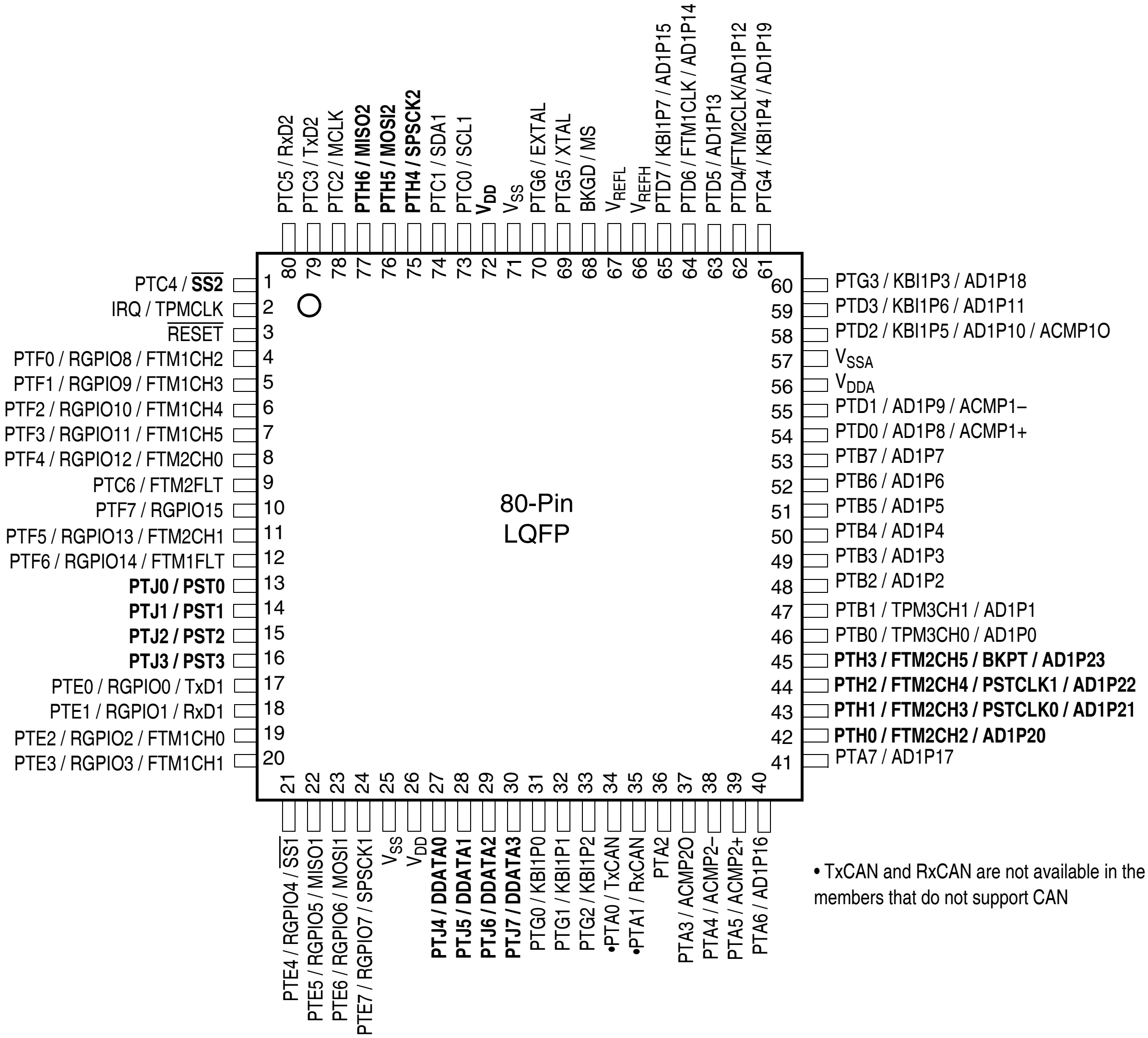
Chapter 2
Pins and Connections

This chapter 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.

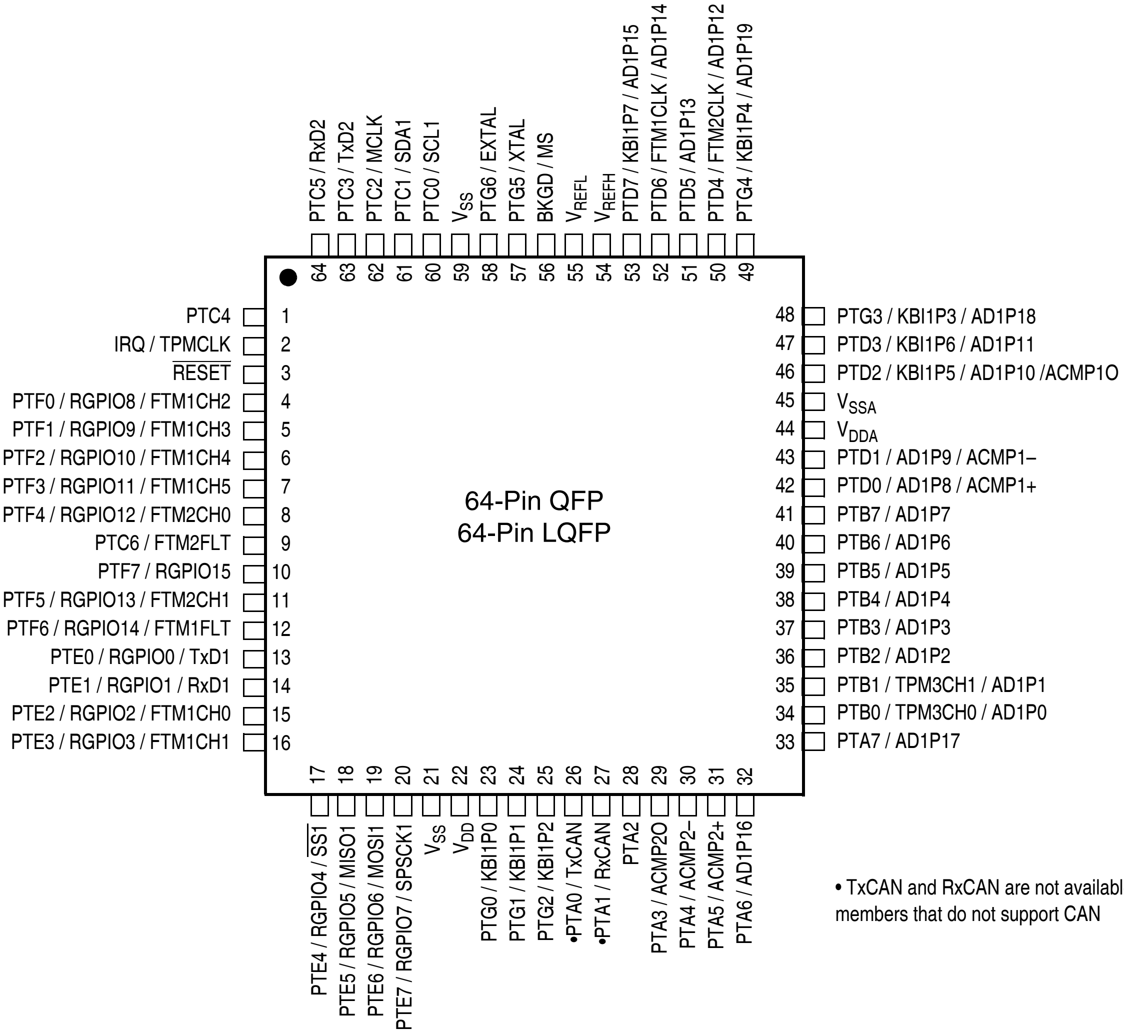


• TxCAN and RxCAN are not available in the members that do not support CAN

Figure 2-1. 80-Pin LQFP

2.1.2 Pinout: 64-Pin LQFP and QFP

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



• TxCAN and RxCAN are not available in the members that do not support CAN

Figure 2-2. 64-Pin LQFP/QFP

2.1.3 Pinout: 44-Pin LQFP

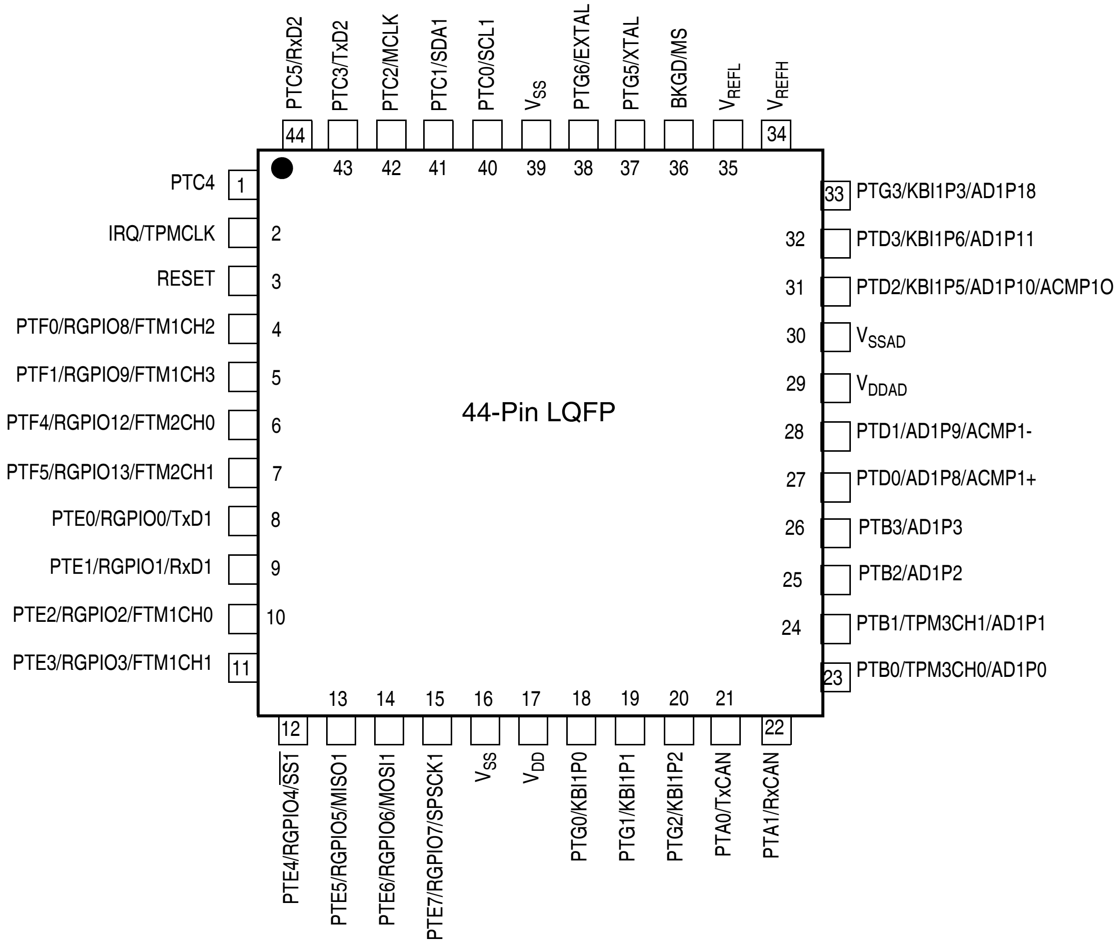


Figure 2-3. 44-Pin LQFP

2.1.4 Pin Assignments

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

Pin Number			Lowest <-- Priority --> Highest			
80	**64**	**44**	**Port Pin**	**Alt 1**	**Alt 2**	**Alt 3**
1	1	1	PTC4	$\overline{\text{SS2}}$		
2	2	2	IRQ	TPMCLK[1]		
3	3	3	$\overline{\text{RESET}}$			
4	4	4	PTF0	RGPIO8	FTM1CH2	
5	5	5	PTF1	RGPIO9	FTM1CH3	
6	6	—	PTF2	RGPIO10	FTM1CH4	
7	7	—	PTF3	RGPIO11	FTM1CH5	
8	8	6	PTF4	RGPIO12	FTM2CH0	
9	9	—	PTC6	FTM2FLT		
10	10	—	PTF7	RGPIO15		
11	11	7	PTF5	RGPIO13	FTM2CH1	
12	12	—	PTF6	RGPIO14	FTM1FLT	
13	—	—	PTJ0	PST0		
14	—	—	PTJ1	PST1		
15	—	—	PTJ2	PST2		
16	—	—	PTJ3	PST3		
17	13	8	PTE0	RGPIO0	TxD1	
18	14	9	PTE1	RGPIO1	RxD1	
19	15	10	PTE2	RGPIO2	FTM1CH0	
20	16	11	PTE3	RGPIO3	FTM1CH1	
21	17	12	PTE4	RGPIO4	$\overline{\text{SS1}}$	
22	18	13	PTE5	RGPIO5	MISO1	
23	19	14	PTE6	RGPIO6	MOSI1	
24	20	15	PTE7	RGPIO7	SPSCK1	
25	21	16	V_{SS}			
26	22	17	V_{DD}			
27	—	—	PTJ4	DDATA0		
28	—	—	PTJ5	DDATA1		
29	—	—	PTJ6	DDATA2		
30	—	—	PTJ7	DDATA3		
31	23	18	PTG0	KBI1P0		
32	24	19	PTG1	KBI1P1		
33	25	20	PTG2	KBI1P2		
34	26	21	PTA0	TxCAN[2]		
35	27	22	PTA1	RxCAN[3]		
36	28	—	PTA2			
37	29	—	PTA3	ACMP2O		
38	30	—	PTA4	ACMP2–		
39	31	—	PTA5	ACMP2+		

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

Pin Number			Lowest <-- Priority --> Highest			
80	**64**	**44**	**Port Pin**	**Alt 1**	**Alt 2**	**Alt 3**
40	32	—	PTA6	AD1P16		
41	33	—	PTA7	AD1P17		
42	—	—	PTH0	FTM2CH2	AD1P20	
43	—	—	PTH1	FTM2CH3	PSTCLK0	AD1P21
44	—	—	PTH2	FTM2CH4	PSTCLK1	AD1P22
45	—	—	PTH3	FTM2CH5	BKPT	AD1P23
46	34	23	PTB0	TPM3CH0	AD1P0	
47	35	24	PTB1	TPM3CH1	AD1P1	
48	36	25	PTB2	AD1P2		
49	37	26	PTB3	AD1P3		
50	38	—	PTB4	AD1P4		
51	39	—	PTB5	AD1P5		
52	40	—	PTB6	AD1P6		
53	41	—	PTB7	AD1P7		
54	42	27	PTD0	AD1P8	ACMP1+	
55	43	28	PTD1	AD1P9	ACMP1–	
56	44	29	V_{DDA}			
57	45	30	V_{SSA}			
58	46	31	PTD2	KBI1P5	AD1P10	ACMP1O
59	47	32	PTD3	KBI1P6	AD1P11	
60	48	33	PTG3	KBI1P3	AD1P18	
61	49	—	PTG4	KBI1P4	AD1P19	
62	50	—	PTD4	FTM2CLK	AD1P12	
63	51	—	PTD5	AD1P13		
64	52	—	PTD6	FTM1CLK	AD1P14	
65	53	—	PTD7	KBI1P7	AD1P15	
66	54	34	V_{REFH}			
67	55	35	V_{REFL}			
68	56	36	BKGD	MS		
69	57	37	PTG5	XTAL		
70	58	38	PTG6	EXTAL		
71	59	39	V_{SS}			
72	—	—	V_{DD}			
73	60	40	PTC0	SCL1		
74	61	41	PTC1	SDA1		
75	—	—	PTH4	SPCK2		
76	—	—	PTH5	MOSI2		
77	—	—	PTH6	MISO2		
78	62	42	PTC2	MCLK		
79	63	43	PTC3	TxD2		
80	64	44	PTC5	RxD2		

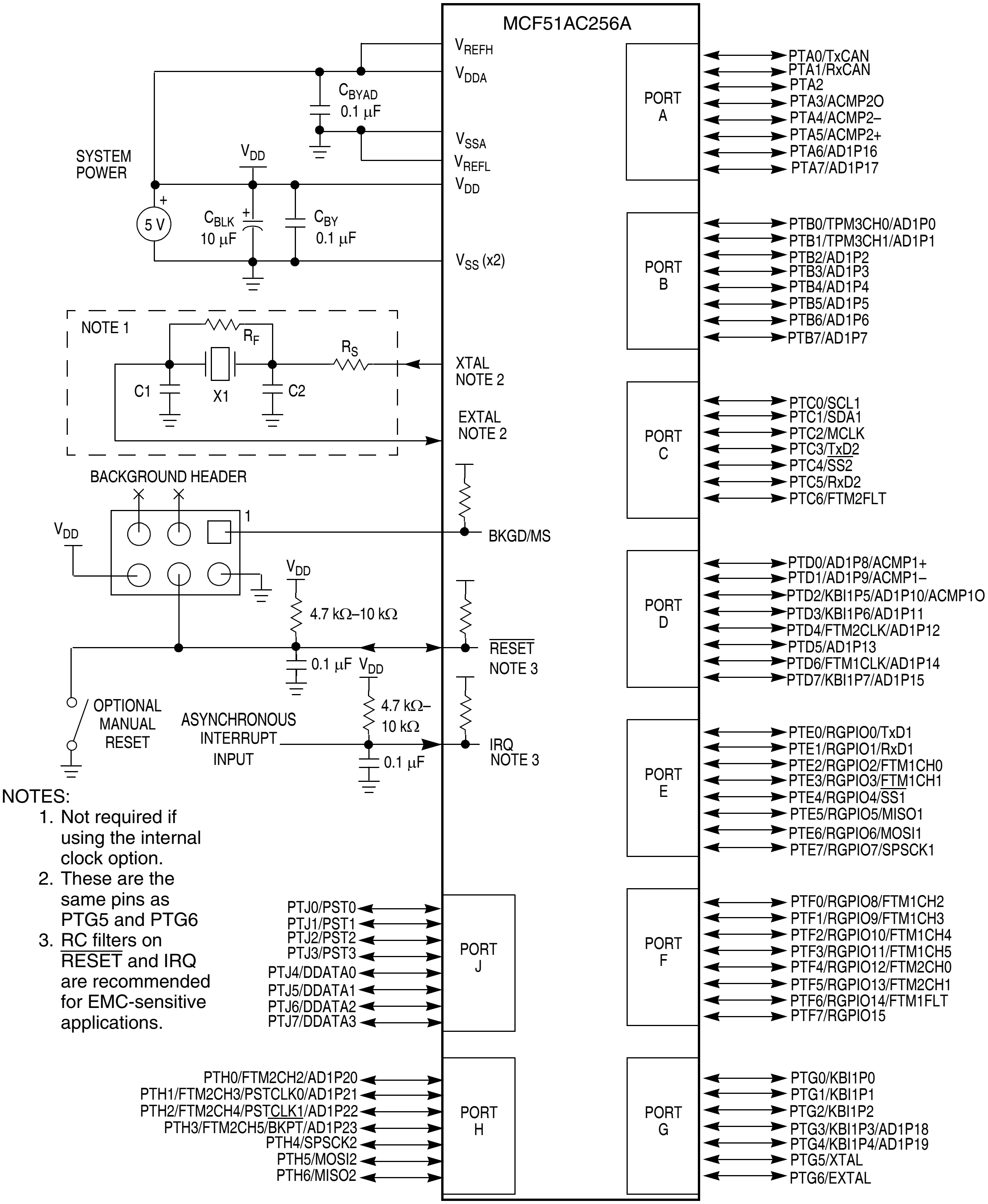
[1] TPMCLK, FTM1CLK, and FTM2CLK options are configured via software; out of reset, FTM1CLK, FTM2CLK, and TPMCLK are available to FTM1, FTM2, and TPM3 respectively.

[2] TxCAN is available in the member that supports CAN.

[3] RxCAN is available in the member that supports CAN.

2.1.5 Recommended System Connections

Figure 2-4 shows pin connections that are common to MCF51AC256 series application systems.



NOTES:

1. Not required if using the internal clock option.
2. These are the same pins as PTG5 and PTG6
3. RC filters on RESET and IRQ are recommended for EMC-sensitive applications.

Figure 2-4. Basic System Connections

2.1.6 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 10µF tantalum capacitor, to provide bulk charge storage for the overall system and a 0.1 µF ceramic bypass capacitor located as close to the microcontroller power pins as practical to suppress high-frequency noise. The MCF51AC256 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 module. A 0.1 µF ceramic bypass capacitor must be located as close to the microcontroller analog power pins as practical to suppress high-frequency noise.

2.1.7 Oscillator (XTAL and EXTAL)

Immediately after reset, the microcontroller uses an internally generated clock provided by the multipurpose clock generation (MCG) 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 MΩ to 10 MΩ. 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).

2.1.8 $\overline{\text{RESET}}$

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 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 n 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.

2.1.9 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.

2.1.10 Background / Mode Select (BKGD/MS)

During a power-on-reset (POR) or background debug force reset (see bit ENBDM in Section 22.3.2, “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[1], 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.

2.1.11 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.

2.1.12 General-Purpose I/O and Peripheral Ports

The MCF51AC256 series microcontrollers support up to 69 general-purpose I/O pins, which are shared with on-chip peripheral functions (timers, serial I/O, ADC, ACMP, 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 pullup device. Immediately after reset, all of these pins are configured as high-impedance general-purpose inputs with internal pullup devices disabled.

When a pin is being used as an input, an input filter is enabled that filters out high frequency transients to prevent system noise from causing an invalid value to be read. When the TMPCLK, FTM1CLK or FTM2CLK signals are enabled, the input filter is disabled on that particular pin. In addition, the input filters can be disabled on the two SPI modules by clearing the SOPT2[SPI1FE] or SOPT2[SPI2FE] bits. The input filters must be disabled to run the SPI module above 5 Mbps.

When an on-chip peripheral system is controlling a pin, data direction control bits 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. For information about controlling these pins as general-purpose I/O pins, see Chapter 6, “Parallel Input/Output Control.”

NOTE

> To avoid extra current drain from floating input pins, the reset initialization routine in the application program must either enable on-chip pullup devices or change the direction of unused or non-bonded pins to outputs so they do not float.

Chapter 3
Modes of Operation

3.1 Introduction

This chapter describes the operating modes of the MCF51AC256 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 16.4.1, “MCG 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 MCF51AC256 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), LVD and/or BDM is enabled. Exited by RESET or active interrupt. Stop4 must be used to enable the ADC in stop mode.
  - Stop3 — All internal circuits are powered but in a standby state unless enabled in stop mode. LVD and BDM must be disabled. Exited by RESET or active interrupt. Stop3 is used to put the MCU into its lowest power state while maintaining RAM and register contents.
  - Stop2 — Most internal circuits are powered-down. RAM content is retained but registers are powered-down. Exited by RESET, IRQ pin or RTI. Stop2 is the lowest power mode for this device.

On this series of 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 WAITE control bit is set when STOP is executed, the wait mode is entered. Otherwise, if the STOPE bit is set, one of the stop modes will be entered. The case of STOPE or WAITE set coupled with a STOP instruction is illegal, and will result in two possible outcomes:

- if CPUCR[IRD]=0, a reset will be asserted;
- otherwise, an illegal instruction exception will be generated.

Table 3-1. CPU / Power Mode Selections

Mode of Operation	Register and Bit names						CPU and Peripheral Clocks	Affects on Sub-System
	SOPT		XCSR	SPMSC1		SPMS C2		
	STOPE	WAITE	ENBDM [1]	LVDE	LVDSE	PPDC		
RUN mode — processor and peripherals clocked normally (RUN-NOM).	x	x	x	x	x	x	On. MCG in any mode	x
WAIT mode — processor and peripherals clocked normally (RUN-NOM).	x	1	x	x	x	x	Peripherals on and clocked. MCG in any mode	x
Stop modes disabled; illegal opcode reset if STOP instruction executed because of CPUCR[IRD] = 0, else an illegal instruction exception is generated; and reset on illegal instruction is enabled.	0	0	function of BKGD/MS at reset	x	x	x	On. MCG in any mode	Function of BKGD/MS at reset
Stop4 — low voltage detections are enabled or ENBDM = 1.	1	0	x	1	1	0	MCG PLL/FLL On. CPU clock and bus clocks off	LVD enabled
			1	x	x		MCG PLL/FLL ON. bus clocks OFF. CPU clock ON	CPU halted. BDC clock enabled only if ENBDM = 1 prior to entering stop

Table 3-1. CPU / Power Mode Selections (continued)

Mode of Operation	Register and Bit names						CPU and Peripheral Clocks	Affects on Sub-System
	SOPT		XCSR	SPMSC1		SPMS C2		
	STOPE	WAITE	ENBDM [1]	LVDE	LVDSE	PPDC		
Stop3 — Low voltage detections in stop are disabled. If BDC is enabled, stop4 will be invoked rather than stop3.	1	0	0	x	0	0	MCG in stop mode. CPU and bus clocks are off. LPO, internal or external Reference clock can be enabled for RTI wakeup.	LVD disabled in stop only.
				0	x			LVD disabled in all modes
Stop2 — Low voltage detections in stop are disabled. If BDC is enabled, stop4 will be invoked rather than stop2.	1	0	0	x	0	1	MCG powered off. LPO clock can be enabled for RTI wakeup.	LVD disabled in stop only. Only RAM powered.
				0	x			LVD disabled. Only RAM powered.

[1] ENBDM is located in the upper byte of the XCSR register which is write accessable only through BDC commands, see Debug module Section 22.3.3, “Configuration/Status Register 2 (CSR2).”

3.4 Secure Mode

While the MCU is in secure mode, there are severe restrictions on which 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 22, “Version 1 ColdFire Debug (CF1_DEBUG),” for details.

3.5 Run Mode

Run mode is the normal operating mode for the MCF51AC256 series MCUs. This mode is selected when the BKGD/MS pin is high at the rising edge of the internal 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 value of the PC.

3.6 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.

The ENBDM bit in the XCSR register 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.

3.7 Stop Modes

One of three stop modes are entered upon execution of a STOP instruction when SOPT[STOPE] is set. SOPT[WAITE] must be clear. 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 MCG module can be configured to leave the reference clocks running. See Chapter 16, “Multipurpose Clock Generator (MCGV3),” 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 stop modes. It 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 following 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 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.

3.7.1 Stop2 Mode

Stop2 mode is entered by executing a STOP instruction under the conditions as shown in Table 3-1. Most of the internal circuitry of the MCU is powered off in stop2 mode, with the exception of the RAM. Upon 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: $\overline{\text{RESET}}$ or IRQ.

NOTE

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

Upon wakeup 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, and the MCU remains in the reset state if V_{DD} is below the LVD trip point (low trip point selected due to POR).
- The CPU initiates the reset exception process.

In addition to the above, upon waking up from stop2 mode, SPMSC2[PPDF] is set. This flag is used to direct user code to go to a stop2 recovery routine. 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], then the pins 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.

3.7.2 Stop3 Mode

Stop3 mode is entered by executing a STOP instruction under the conditions as 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 RESET, or by an interrupt from one of the following sources: RTI, MSCAN wakeup interrupt, SCI edge detect interrupt, IRQ, KBI or ACMP. If stop3 mode is exited by means of the $\overline{\text{RESET}}$ pin, then the MCU is 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

> The interrupt source must not be masked by software if it is active and enabled in stop3 or stop4. Failure to do so may lead to a high current condition with the CPU remaining in stop mode.

3.7.3 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 RESET, or by an interrupt from one of the following sources: RTI, MSCAN wakeup interrupt, SCI edge detect interrupt, LVD, ADC, IRQ, KBI, or ACMP. If stop4 is exited by means of the RESET pin, then the MCU is reset and operation will resume 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

The interrupt source must not be masked by software if it is active and enabled in stop3 or stop4. Failure to do so may lead to a high current condition with the CPU remaining in stop mode.

3.7.3.1 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, then the voltage regulator 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. For the ADC to operate, the LVD must be left enabled when entering stop4 mode. For the ACMP to operate when ACMPSC[ACGBS] is set, the LVD must be left enabled when entering stop4. For the OSC to operate with an external reference when MCGC2[RANGE] is set, the LVD must be left enabled when entering stop4 mode.

3.8 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 are stopped. Even in the exception case (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 3.7.1, “Stop2 Mode,” and Section 3.7.2, “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 (SCGC1 and SCGC2).

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[1]	Not Clocked
Full Regulation	FullOn	FullNoClk FullADACK[2]

Table 3-2. Abbreviations used in Table 3-3 (continued)

Voltage Regulator	Clocked[1]	Not Clocked
Soft Regulation	SoftOn[3]	SoftNoClk Disabled SoftADACK[4]
Off	N/A	Off

[1] Subject to module enables and settings of System Clock Gating Control Registers 1 and 2 (SCGC1 and SCGC2).

[2] This ADC-specific mode defines the case 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.

[3] Analog modules must be in their low power mode when the device is operated in this state.

[4] This ADC-specific mode defines the case where the device is in soft regulation and the normal peripheral clock is stopped. In this case, the ADC can only be 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
CF1CORE	Off	SoftNoClk	FullNoClk	FullNoClk
RAM	SoftNoClk	SoftNoClk	FullNoClk	FullNoClk
Flash	Off	SoftNoClk	FullNoClk	FullNoClk
Port I/O Registers	Off	SoftNoClk	FullNoClk	FullOn
ADC[1,2]	Off	SoftNoClk	FULLADACK (Wakeup)	FullOn
ACMP*x*	Off	SoftNoClk (Wake Up)	FullNoClk (Wakeup)	FullOn
BDC	Off	SoftNoClk	On	FullOn
COP	Off	SoftNoClk	FullNoClk	FullOn
CRC	Off	SoftNoClk	SoftNoClk	FullOn

Crystal Oscillator	Off	RANGE=0
		HGO=0
			All Modes	All Modes

FTMx	Off	SoftNoClk	FullNoClk	FullOn
IIC*x*	Off	SoftNoClk	FullNoClk	FullOn
IRQ	Off (Wakeup via POR)[3]	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
KBI*x*	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
LVD/LVW	Off	Disabled	On (Wakeup)	FullOn

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

Peripheral	Mode			
	STOP2	STOP3	STOP4	WAIT
MCG	Off	STOP or BLPE[4]	STOP or any mode	Any mode
MSCAN	Off	SoftNoClk (Wakeup)	FullNoClk	FullOn
RTI	Soft Regulation, LPOCLK if enabled (Wake Up via POR)	SoftOn LPOCLK or ICSERCLK (Wakeup)	Full Regulation LPOCLK, ICSERCLK or ICSIRCLK only (Wakeup)	FullOn
SCI*x*	Off	SoftNoClk (Wakeup)	FullNoClk (Wakeup)	FullOn
SPI*x*	Off	SoftNoClk	FullNoClk	FullOn
TPM*3*	Off	SoftNoClk	FullNoClk	FullOn
Voltage Regulator / PMC	Partial Shutdown. 1 kHz osc if enabled	Soft Regulation. 1 kHz osc if enabled	Full Regulation 1 kHz osc on	FullOn 1 kHz osc on
GPI/O Pins	States Held	SoftNoClk	FullNoClk	FullOn

[1] LP mode for the ADC is invoked by setting ADLPC = 1. ADACK is selected via the ADCCFG[ADICLK] field in the ADC. See Chapter 9, “Analog-to-Digital Converter (ADC12V1),” for details.

[2] LVD must be enabled to run in stop if converting the bandgap channel.

[3] The PTA5/IRQ/TPM1CLK/$\overline{\text{RESET}}$ pin has a direct connection to the on-chip regulator wakeup input. Asserting a low on this pin while in stop2 will trigger the PMC to wake. As a result, the device will undergo a power-on-reset sequence.

[4] BLPE refers to the MCG “Bypassed Low Power External” state. See Chapter 16, “Multipurpose Clock Generator (MCGV3),” for more details.

3.9 Debug Mode

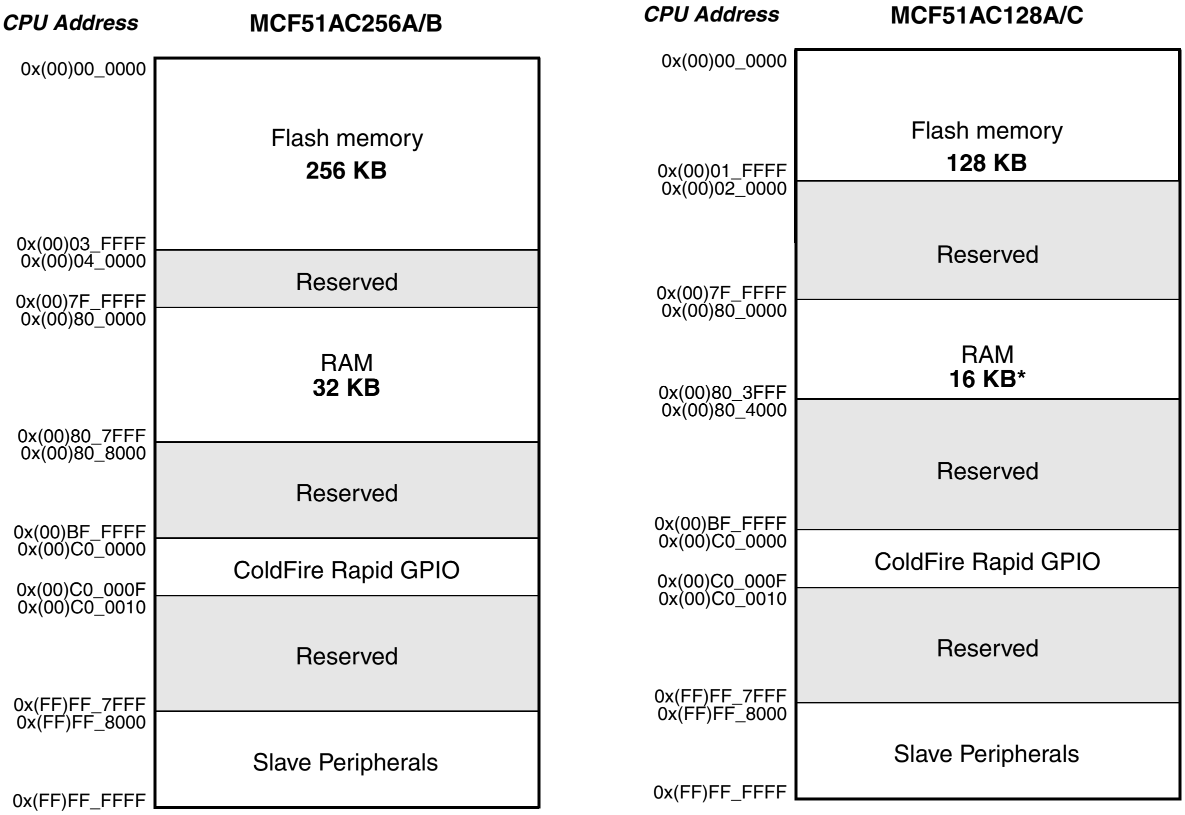
The debug interface is used to program a bootloader or user application program into the flash program memory before the MCU is operated in run mode for the first time. When a MCF51AC256 series MCUs are shipped from the Freescale Semiconductor factory, the flash program memory is erased by default unless 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 22, “Version 1 ColdFire Debug (CF1_DEBUG).”

Chapter 4
Memory

4.1 MCF51AC256 Series Memory Map

As shown in Figure 4-1, on-chip memory in the MCF51AC256 series microcontrollers consist of RAM and flash program memory for nonvolatile data storage, plus I/O and control/status registers.



Note: MCF51AC128A has 32 KB RAM.

Figure 4-1. MCF51AC256 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 slave peripherals section of the memory map is further broken into the following sub-sections:

```
0x(FF)FF_8000 – 0x(FF)FF_807F       Direct-page equivalent peripheral regs
0x(FF)FF_9800 – 0x(FF)FF_98FF       High-page equivalent peripheral regs
0x(FF)FF_FFC0 – 0x(FF)FF_FFFF       Interrupt controller
```

The section of memory at 0x(00)C0_0000 is assigned for use by the ColdFire Rapid GPIO module. See Table 4-5 for the rapid GPIO memory map and Chapter 17, “Rapid GPIO (RGPIO),” for further details on the module.

The MCF51AC256 series microcontrollers utilize an 8-bit peripheral bus. The bus bridge from the ColdFire system bus to the peripheral bus is capable of serializing 16-bit accesses 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.

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] = 0 or an address error exception if CPUCR[ARD] = 1.

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 Register Addresses and Bit Assignments

Peripheral registers in the MCF51AC256 series microcontrollers are divided into two groups:

- Direct-page equivalent registers begin at 0x(FF)FF_8000 in the memory map.
- High-page equivalent registers begin at 0x(FF)FF_9800 in the memory map.

There is no functional advantage to locating peripherals in the direct-page versus the high-page peripheral space for an MCF51AC256 series microcontroller. Both sets of registers may be efficiently accessed using the ColdFire absolute short addressing mode. The areas are differentiated to maintain documentation compatibility with the MC9S08AC128.

Peripheral register addresses for the MCF51AC256 series microcontrollers are shifted 0x(FF)FF_8000 compared with the MC9S08AC128 devices. The address offsets between registers are maintained compared to the MC9S08AC128 devices in order to maintain code compatibility.

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 a user 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 4-2, Table 4-3, Table 4-5 and Table 4-6, 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.

Recall that ColdFire has a big endian byte addressable memory architecture. The most significant byte of each address is the lowest numbered as shown in Figure 4-2. Multi-byte operands (e.g., 16-bit words and 32-bit longwords) are referenced using an address pointing to the most significant (first) byte.

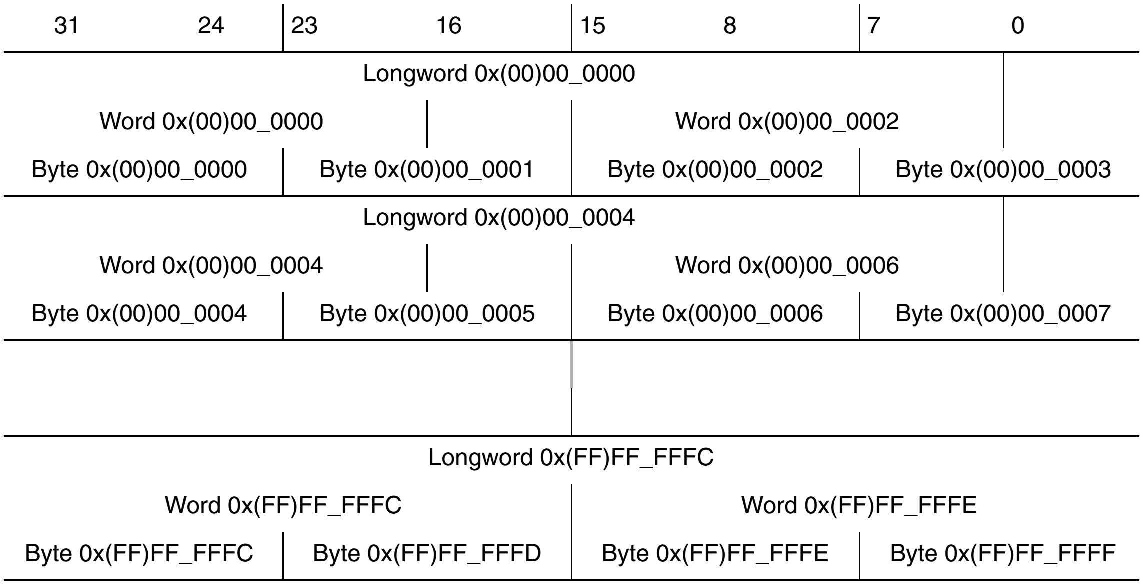


Figure 4-2. ColdFire Memory Organization

4.2.1 I/O and Control Registers

Table 4-2 is a summary of all user-accessible direct-page equivalent registers and control bits.

Table 4-2. Direct-Page Equivalent Register Summary (Sheet 1 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8000	**PTAD**	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x(FF)FF_8001	**PTADD**	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
0x(FF)FF_8002	**PTBD**	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
0x(FF)FF_8003	**PTBDD**	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
0x(FF)FF_8004	**PTCD**	0	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0x(FF)FF_8005	**PTCDD**	0	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
0x(FF)FF_8006	**PTDD**	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0x(FF)FF_8007	**PTDDD**	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
0x(FF)FF_8008	**PTED**	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
0x(FF)FF_8009	**PTEDD**	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
0x(FF)FF_800A	**PTFD**	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
0x(FF)FF_800B	**PTFDD**	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
0x(FF)FF_800C	**PTGD**	0	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
0x(FF)FF_800D	**PTGDD**	0	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
0x(FF)FF_800E	**PTHD**	0	PTHD6	PTHD5	PTHD4	PTHD3	PTHD2	PTHD1	PTHD0
0x(FF)FF_800F	**PTHDD**	0	PTHDD6	PTHDD5	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
0x(FF)FF_8010	**ADCSC1**	COCO	AIEN	ADCO	ADCH				
0x(FF)FF_8011	**ADCSC2**	ADACT	ADTRG	ACFE	ACFGT	0	0	R	R
0x(FF)FF_8012	**ADCRH**	0	0	0	0	ADR11	ADR10	ADR9	ADR8
0x(FF)FF_8013	**ADCRL**	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
0x(FF)FF_8014	**ADCCVH**	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8
0x(FF)FF_8015	**ADCCVL**	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0
0x(FF)FF_8016	**ADCCFG**	ADLPC	ADIV		ADLSMP	MODE		ADICLK	
0x(FF)FF_8017	**APCTL1**	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
0x(FF)FF_8018	**APCTL2**	ADPC15	ADPC14	ADPC13	ADPC12	ADPC11	ADPC10	ADPC9	ADPC8
0x(FF)FF_8019	**APCTL3**	ADPC23	ADPC22	ADPC21	ADPC20	ADPC19	ADPC18	ADPC17	ADPC16
0x(FF)FF_801A	**PTJD**	PTJD7	PTJD6	PTJD5	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0
0x(FF)FF_801B	**PTJDD**	PTJDD7	PTJDD6	PTJDD5	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0
0x(FF)FF_801C	**IRQSC**	0	IRQPDD	IRQEDG	IRQPE	IRQF	IRQACK	IRQIE	IRQMOD
0x(FF)FF_801D	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_801E	**KBISC**	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBF	KBACK	KBIE	KBIMOD
0x(FF)FF_801F	**KBIPE**	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0
0x(FF)FF_8020– 0x(FF)FF_8037	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8038	**SCI1BDH**	LBKDIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8039	**SCI1BDL**	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_803A	**SCI1C1**	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_803B	**SCI1C2**	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_803C	**SCI1S1**	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_803D	**SCI1S2**	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF

Table 4-2. Direct-Page Equivalent Register Summary (Sheet 2 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_803E	**SCI1C3**	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_803F	**SCI1D**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8040	**SCI2BDH**	LBKDIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8041	**SCI2BDL**	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8042	**SCI2C1**	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8043	**SCI2C2**	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_8044	**SCI2S1**	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_8045	**SCI2S2**	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_8046	**SCI2C3**	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_8047	**SCI2D**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8048	**MCGC1**	CLKS		RDIV			IREFS	IRCLKEN	IREFSTEN
0x(FF)FF_8049	**MCGC2**	BDIV		RANGE	HGO	LP	EREFS	ERCLKEN	EREFSTEN
0x(FF)FF_804A	**MCGTRM**	TRIM							
0x(FF)FF_804B	**MCGSC**	LOLS	LOCK	PLLST	IREFST	CLKST		OSCINIT	FTRIM
0x(FF)FF_804C	**MCGC3**	LOLIE	PLLS	CME	DIV32	VDIV			
0x(FF)FF_804D	**MCGC4**	0	0	DMX32	0	0	0	DRS/DRST	
0x(FF)FF_804E	**MCGT**	RESERVED FOR FACTORY USE							
0x(FF)FF_804F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8050	**SPI1C1**	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_8051	**SPI1C2**	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
0x(FF)FF_8052	**SPI1BR**	0	SPPR			0	SPR		
0x(FF)FF_8053	**SPI1S**	SPRF	0	SPTEF	MODF	0	0	0	0
0x(FF)FF_8054	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_8055	**SPI1D**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8056	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8057	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8058	**IICA**	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
0x(FF)FF_8059	**IICF**	MULT		ICR					
0x(FF)FF_805A	**IICC1**	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
0x(FF)FF_805B	**IICS**	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
0x(FF)FF_805C	**IICD**	DATA							
0x(FF)FF_805D	**IICC2**	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
0x(FF)FF_805E	**ACMP1SC**	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD	
0x(FF)FF_805F	**ACMP2SC**	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD	
0x(FF)FF_8060– 0x(FF)FF_807F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8080	**FTM1SC**	TOF	TOIE	CPWMS	CLKS		PS		
0x(FF)FF_8081	**FTM1CNTH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8082	**FTM1CNTL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8083	**FTM1MODH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8084	**FTM1MODL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8085	**FTM1C0SC**	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8086	**FTM1C0VH**	Bit 15	14	13	12	11	10	9	Bit 8

Table 4-2. Direct-Page Equivalent Register Summary (Sheet 3 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8087	**FTM1C0VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8088	**FTM1C1SC**	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_8089	**FTM1C1VH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_808A	**FTM1C1VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_808B	**FTM1C2SC**	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	0
0x(FF)FF_808C	**FTM1C2VH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_808D	**FTM1C2VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_808E	**FTM1C3SC**	CH3F	CH3IE	MS3B	MS3A	ELS3B	ELS3A	0	0
0x(FF)FF_808F	**FTM1C3VH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8090	**FTM1C3VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8091	**FTM1C4SC**	CH4F	CH4IE	MS4B	MS4A	ELS4B	ELS4A	0	0
0x(FF)FF_8092	**FTM1C4VH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8093	**FTM1C4VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8094	**FTM1C5SC**	CH5F	CH5IE	MS5B	MS5A	ELS5B	ELS5A	0	0
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_809C	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_809D	**FTM1CNTINH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_809E	**FTM1CNTINL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_809F	**FTM1STATUS**	0	0	CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
0x(FF)FF_80A0	**FTM1MODE**	FAULTIE	FAULTM		0	0	WPDIS	INIT	FTMEN
0x(FF)FF_80A1	**FTM1SYNC**	SWSYNC	TRIG2	TRIG1	TRIG0	SYNCHOM	REINIT	CNTMAX	CNTMIN
0x(FF)FF_80A2	**FTM1OUTINIT**	0	0	CH5OI	CH4OI	CH3OI	CH2OI	CH1OI	CH0OI
0x(FF)FF_80A3	**FTM1OUTMASK**	0	0	CH5OM	CH4OM	CH3OM	CH2OM	CH1OM	CH0OM
0x(FF)FF_80A4	**FTM1COMBINE0**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80A5	**FTM1COMBINE1**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80A6	**FTM1COMBINE2**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80A7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80A8	**FTM1DEADTIME**	DTPS		DTVAL					
0x(FF)FF_80A9	**FTM1EXTTRIG**	TRIGF	—	0	0	CH5TRIG	CH4TRIG	CH3TRIG	CH2TRIG
0x(FF)FF_80AA	**FTM1POL**	0	0	POL5	POL4	POL3	POL2	POL1	POL0
0x(FF)FF_80AB	**FTM1FMS**	FAULTF	WPEN	FAULTIN	0	0	0	0	0
0x(FF)FF_80AC	**FTM1FILTER0**	CH1FVAL				CH0FVAL			
0x(FF)FF_80AD	**FTM1FILTER1**	CH3FVAL				CH2FVAL			
0x(FF)FF_80AE	**FTM1FLTFILTER**	0	0	0	0	FFVAL			
0x(FF)FF_80AF	**FTM1RFU**	0	0	0	0	0	0	0	0
0x(FF)FF_80B0– 0x(FF)FF_80BF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80C0	**FTM2SC**	TOF	TOIE	CPWMS	CLKS		PS		
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

Table 4-2. Direct-Page Equivalent Register Summary (Sheet 4 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
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	0
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	0
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	0
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	0
0x(FF)FF_80CF	**FTM2C3VH**	Bit 15	14	13	12	11	10	9	Bit 8
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	0
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	0
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_80DC	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80DD	**FTM2CNTINH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_80DE	**FTM2CNTINL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_80DF	**FTM2STATUS**	0	0	CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
0x(FF)FF_80E0	**FTM2MODE**	FAULTIE	FAULTM		0	0	WPDIS	INIT	FTMEN
0x(FF)FF_80E1	**FTM2SYNC**	SWSYNC	TRIG2	TRIG1	TRIG0	SYNCHOM	REINIT	CNTMAX	CNTMIN
0x(FF)FF_80E2	**FTM2OUTINIT**	0	0	CH5OI	CH4OI	CH3OI	CH2OI	CH1OI	CH0OI
0x(FF)FF_80E3	**FTM2OUTMASK**	0	0	CH5OM	CH4OM	CH3OM	CH2OM	CH1OM	CH0OM
0x(FF)FF_80E4	**FTM2COMBINE0**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80E5	**FTM2COMBINE1**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80E6	**FTM2COMBINE2**	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
0x(FF)FF_80E7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80E8	**FTM2DEADTIME**	DTPS		DTVAL					
0x(FF)FF_80E9	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_80EA	**FTM2POL**	0	0	POL5	POL4	POL3	POL2	POL1	POL0
0x(FF)FF_80EB	**FTM2FMS**	FAULTF	WPEN	FAULTIN	0	0	0	0	0
0x(FF)FF_80EC	**FTM2FILTER0**	CH1FVAL				CH0FVAL			
0x(FF)FF_80ED	**FTM2FILTER1**	CH3FVAL				CH2FVAL			
0x(FF)FF_80EE	**FTM2FLTFILTER**	0	0	0	0	FFVAL			
0x(FF)FF_80EF	**FTM2RFU**	0	0	0	0	0	0	0	0
0x(FF)FF_80F0– 0x(FF)FF_80FF	Reserved	—	—	—	—	—	—	—	—

Table 4-3 shows the registers that equate to the high-page register of the MCF51AC256 series. These registers are typically accessed less often than other I/O and control registers.

Table 4-3. High-Page Equivalent Register Summary (Sheet 1 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9800	**SRS**	POR	PIN	COP	ILOP	ILAD	LOC	LVD	0
0x(FF)FF_9801	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9802	**SOPT**	COPE	COPT	STOPE	WAITE	0	0	0	0
0x(FF)FF_9803	**SMCLK**	0	0	0	MPE	0	MCSEL		
0x(FF)FF_9804– 0x(FF)FF_9805	Reserved	— —	— —	— —	— —	— —	— —	— —	— —
0x(FF)FF_9806	**SDIDH**	—	—	—	—	ID11	ID10	ID9	ID8
0x(FF)FF_9807	**SDIDL**	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0x(FF)FF_9808	**SRTISC**	RTIF	RTIACK	RTICLKS	RTIE	0	RTIS2	RTIS1	RTIS0
0x(FF)FF_9809	**SPMSC1**	LVDF	LVDACK	LVDIE	LVDRE	LVDSE	LVDE	0	BGBE
0x(FF)FF_980A	**SPMSC2**	LVWF	LVWACK	LVDV	LVWV	PPDF	PPDACK	0	PPDC
0x(FF)FF_980B	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_980C	**SOPT2**	COPCLKS	SPI2FE	SPI1FE	ACIC2	TPMCCFG	ACIC1	ADHWTS	
0x(FF)FF_980D	**SCGC1**	TPM3	FTM2	FTM1	ADC	CAN	IIC	SCI2	SCI1
0x(FF)FF_980E	**SCGC2**	CRC	FLS	IRQ	KBI*x*	ACMP*x*	RTI	SPI2	SPI1
0x(FF)FF_980F– 0x(FF)FF_981F	Reserved	— —	— —	— —	— —	— —	— —	— —	— —
0x(FF)FF_9820	**FCDIV**	FDIVLD	PRDIV8	FDIV					
0x(FF)FF_9821	**FOPT**	KEYEN		0	0	0	0	SEC	
0x(FF)FF_9822	**FRSV0 (Reserved)**	0	0	0	0	0	0	0	0
0x(FF)FF_9823	**FCNFG**	0	0	KEYACC	0	0	0	0	0
0x(FF)FF_9824	**FPROT**	FPS							FPOPEN
0x(FF)FF_9825	**FSTAT**	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
0x(FF)FF_9826	**FCMD**	0	FCMD						
0x(FF)FF_9827	**FRSV1 (Reserved)**	—	—	—	—	—	—	—	—
0x(FF)FF_9828	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9829	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_982A	**FRSV2 (Reserved)**	—	—	—	—	—	—	—	—
0x(FF)FF_982B	**FRSV3 (Reserved)**	—	—	—	—	—	—	—	—
0x(FF)FF_982C– 0x(FF)FF_982F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9830	**TPM3SC**	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0x(FF)FF_9831	**TPM3CNTH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9832	**TPM3CNTL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9833	**TPM3MODH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_9834	**TPM3MODL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9835	**TPM3C0SC**	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_9836	**TPM3C0VH**	Bit 15	14	13	12	11	10	9	Bit 8

Table 4-3. High-Page Equivalent Register Summary (Sheet 2 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9837	**TPM3C0VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9838	**TPM3C1SC**	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_9839	**TPM3C1VH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_983A	**TPM3C1VL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_983B– 0x(FF)FF_983F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9840	**PTAPE**	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
0x(FF)FF_9841	**PTASE**	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
0x(FF)FF_9842	**PTADS**	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
0x(FF)FF_9843	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9844	**PTBPE**	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
0x(FF)FF_9845	**PTBSE**	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
0x(FF)FF_9846	**PTBDS**	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
0x(FF)FF_9847	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9848	**PTCPE**	0	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0
0x(FF)FF_9849	**PTCSE**	0	PTCSE6	PTCSE5	PTCSE4	PTCSE3	PTCSE2	PTCSE1	PTCSE0
0x(FF)FF_984A	**PTCDS**	0	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
0x(FF)FF_984B	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_984C	**PTDPE**	PTDPE7	PTDPE6	PTDPE5	PTDPE4	PTDPE3	PTDPE2	PTDPE1	PTDPE0
0x(FF)FF_984D	**PTDSE**	PTDSE7	PTDSE6	PTDSE5	PTDSE4	PTDSE3	PTDSE2	PTDSE1	PTDSE0
0x(FF)FF_984E	**PTDDS**	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
0x(FF)FF_984F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9850	**PTEPE**	PTEPE7	PTEPE6	PTEPE5	PTEPE4	PTEPE3	PTEPE2	PTEPE1	PTEPE0
0x(FF)FF_9851	**PTESE**	PTESE7	PTESE6	PTESE5	PTESE4	PTESE3	PTESE2	PTESE1	PTESE0
0x(FF)FF_9852	**PTEDS**	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
0x(FF)FF_9853	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9854	**PTFPE**	PTFPE7	PTFPE6	PTFPE5	PTFPE4	PTFPE3	PTFPE2	PTFPE1	PTFPE0
0x(FF)FF_9855	**PTFSE**	PTFSE7	PTFSE6	PTFSE5	PTFSE4	PTFSE3	PTFSE2	PTFSE1	PTFSE0
0x(FF)FF_9856	**PTFDS**	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
0x(FF)FF_9857	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9858	**PTGPE**	0	PTGPE6	PTGPE5	PTGPE4	PTGPE3	PTGPE2	PTGPE1	PTGPE0
0x(FF)FF_9859	**PTGSE**	0	PTGSE6	PTGSE5	PTGSE4	PTGSE3	PTGSE2	PTGSE1	PTGSE0
0x(FF)FF_985A	**PTGDS**	0	PTGDS6	PTGDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
0x(FF)FF_985B	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_985C	**PTHPE**	0	PTHPE6	PTHPE5	PTHPE4	PTHPE3	PTHPE2	PTHPE1	PTHPE0
0x(FF)FF_985D	**PTHSE**	0	PTHSE6	PTHSE5	PTHSE4	PTHSE3	PTHSE2	PTHSE1	PTHSE0
0x(FF)FF_985E	**PTHDS**	0	PTHDS6	PTHDS5	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
0x(FF)FF_985F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9860	**PTJPE**	PTJPE7	PTJPE6	PTJPE5	PTJPE4	PTJPE3	PTJPE2	PTJPE1	PTJPE0
0x(FF)FF_9861	**PTJSE**	PTJSE7	PTJSE6	PTJSE5	PTJSE4	PTJSE3	PTJSE2	PTJSE1	PTJSE0
0x(FF)FF_9862	**PTJDS**	PTJDS7	PTJDS6	PTJDS5	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0
0x(FF)FF_9863– 0x(FF)FF_9867	Reserved	—	—	—	—	—	—	—	—

Table 4-3. High-Page Equivalent Register Summary (Sheet 3 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9868	**SPI2C1**	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_9869	**SPI2C2**	SPMIE	SPIMODE	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
0x(FF)FF_986A	**SPI2BR**	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
0x(FF)FF_986B	**SPI2S**	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
0x(FF)FF_986C	**SPI2DH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_986D	**SPI2DL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_986E	**SPI2MH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_986F	**SPI2ML**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9870	**SPI2C3**	0	0	TNEAREF MARK	RNFULL MARK	INTCLR	TNEARIEN	RNFULLIEN	FIFOMODE
0x(FF)FF_9871	**SPI2CI**	TXFERR	RXFERR	TXFOF	RXFOF	TNEAREFCI	RNFULLFCI	SPTEFCI	SPRFCI
0x(FF)FF_9872– 0x(FF)FF_987F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9880	**CANCTL0**	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
0x(FF)FF_9881	**CANCTL1**	CANE	CLKSRC	LOOPB	LISTEN	BORM	WUPM	SLPAK	INITAK
0x(FF)FF_9882	**CANBTR0**	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
0x(FF)FF_9883	**CANBTR1**	SAMP	TSEG22	TSEG21	TSEG20	TESEG13	TESEG12	TSEG11	TSEG10
0x(FF)FF_9884	**CANRFLG**	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
0x(FF)FF_9885	**CANRIER**	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
0x(FF)FF_9886	**CANTFLG**	0	0	0	0	0	TXE2	TXE1	TXE0
0x(FF)FF_9887	**CANTIER**	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
0x(FF)FF_9888	**CANTARQ**	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
0x(FF)FF_9889	**CANTAAK**	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
0x(FF)FF_988A	**CANTBSEL**	0	0	0	0	0	TX2	TX1	TX0
0x(FF)FF_988B	**CANIDAC**	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
0x(FF)FF_988C	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_988D	**CANMISC**	0	0	0	0	0	0	0	BOHOLD
0x(FF)FF_988E	**CANRXERR**	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
0x(FF)FF_988F	**CANTXERR**	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
0x(FF)FF_9890	**CANIDAR0**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_9891	**CANIDAR1**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_9892	**CANIDAR2**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_9893	**CANIDAR3**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_9894	**CANIDMR0**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_9895	**CANIDMR1**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_9896	**CANIDMR2**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_9897	**CANIDMR3**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_9898	**CANIDAR4**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_9899	**CANIDAR5**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_989A	**CANIDAR6**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_989B	**CANIDAR7**	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
0x(FF)FF_989C	**CANIDMR4**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_989D	**CANIDMR5**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0

Table 4-3. High-Page Equivalent Register Summary (Sheet 4 of 4)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_989E	**CANIDMR6**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_989F	**CANIDMR7**	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
0x(FF)FF_98A0– 0x(FF)FF_98AF	**MSCAN Rx Buffer**	For complete details on the MSCAN Rx buffer, see the MSCAN chapter							
0x(FF)FF_98B0– 0x(FF)FF_98BF	**MSCAN Tx Buffer**	For complete details on the MSCAN Tx buffer, see the MSCAN chapter							
0x(FF)FF_98C0	**CRCH**	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_98C1	**CRCL**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C2	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_98C3	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_98C4	**CRCL0**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C5	**CRCL1**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C6	**CRCL2**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C7	**CRCL3**	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C8– 0x(FF)FF_98CF	Reserved	—	—	—	—	—	—	—	—

4.2.2 Flash Module Reserved Memory Locations

Several reserved flash memory locations, shown in Table 4-4, 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 4-4 reflects the fact that ColdFire uses big endian addressing. See the ColdFire documentation for further details.

Table 4-4. Reserved Flash Memory Addresses

Address	MSB (0x0)	(0x1)	(0x2)	LSB (0x3)
0x(00)00_03FC	Reserved		FTRIM (bit 0)	TRIM
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
0x(00)00_0408	Reserved			
0x(00)00_040C	Reserved	NVPROT	Reserved	NVOPT

The 1 kHz oscillator, bandgap, and MCG trim values are stored in the IFR[1]. Bandgap and MCG trim values will automatically be loaded from the IFR into the MCGTRM and MCGSC 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 4-4. These can be reprogrammed by third party programmers and can be copied into the corresponding MCG registers (MCGTRM and MCGSC) by user code to override the factory trim.

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.

4.2.3 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 4-5.

Table 4-5. 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**	15	14	13	12	11	10	9	8
0x(00)C0_0001	**RGPIO_DIR**	7	6	5	4	3	2	1	0
0x(00)C0_0002	**RGPIO_DATA**	15	14	13	12	11	10	9	8
0x(00)C0_0003	**RGPIO_DATA**	7	6	5	4	3	2	1	0
0x(00)C0_0004	**RGPIO_ENB**	15	14	13	12	11	10	9	8
0x(00)C0_0005	**RGPIO_ENB**	7	6	5	4	3	2	1	0
0x(00)C0_0006	**RGPIO_CLR**	15	14	13	12	11	10	9	8
0x(00)C0_0007	**RGPIO_CLR**	7	6	5	4	3	2	1	0
0x(00)C0_0008	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_0009	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000A	**RGPIO_SET**	15	14	13	12	11	10	9	8
0x(00)C0_000B	**RGPIO_SET**	7	6	5	4	3	2	1	0
0x(00)C0_000C	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000D	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000E	**RGPIO_TOG**	15	14	13	12	11	10	9	8
0x(00)C0_000F	**RGPIO_TOG**	7	6	5	4	3	2	1	0

4.2.4 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.

[1] **Flash IFR** — Nonvolatile information memory, consisting of 256 bytes, located in the flash block outside of flash main memory.

Table 4-6. V1 ColdFire Interrupt Controller Memory Map

Address	Register Name	msb			Bit Number				lsb
0x(FF)FF_FFC0– 0x(FF)FF_FFCF	Reserved	—	—	—	—	—	—	—	—
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– 0x(FF)FF_FFDD	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFDE	**INTC_SFRC**	0	0	SET					
0x(FF)FF_FFDF	**INTC_CFRC**	0	0	CLR					
0x(FF)FF_FFE0	**INTC_SWIACK**	0	VECN						
0x(FF)FF_FFE1– 0x(FF)FF_FFE3	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFE4	**INTC_LVL1IACK**	0	VECN						
0x(FF)FF_FFE5– 0x(FF)FF_FFE7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFE8	**INTC_LVL2IACK**	0	VECN						
0x(FF)FF_FFE9– 0x(FF)FF_FFEB	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFEC	**INTC_LVL3IACK**	0	VECN						
0x(FF)FF_FFED– 0x(FF)FF_FFEF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF0	**INTC_LVL4IACK**	0	VECN						
0x(FF)FF_FFF1– 0x(FF)FF_FFF3	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF4	**INTC_LVL5IACK**	0	VECN						
0x(FF)FF_FFF5– 0x(FF)FF_FFF7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF8	**INTC_LVL6IACK**	0	VECN						
0x(FF)FF_FFF9– 0x(FF)FF_FFFB	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFFC	**INTC_LVL7IACK**	0	VECN						
0x(FF)FF_FFFD– 0x(FF)FF_FFFF	Reserved	—	—	—	—	—	—	—	—

4.3 RAM

An MCF51AC256 series microcontrollers include up to 32 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 MCF51AC256 series MCUs must be programmed 32 bits at a time. The MCF51AC256 series flash memory is organized as two 16-bit wide blocks interleaved to yield a 32-bit data path.

4.4.1 Features

Features of the flash memory include:

- Flash size
  - MCF51AC256: 262,144 bytes (128 sectors of 2048 bytes each)
  - MCF51AC128: 131,072 bytes (64 sectors of 2048 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/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 is used to control the length of timed events in program and erase algorithms executed by the flash memory controller.

	7	6	5	4	3	2	1	0
R W	FDIVLD	PRDIV8	FDIV					
Reset	0	0	0	0	0	0	0	0

Figure 4-3. 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 4-7).

Table 4-7. FCDIV Field Descriptions

Field	Description
7 FDIVLD	**Clock Divider Load Control** — 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 a 0 to FDIVLD locks the FCDIV register contents; all future writes to FCDIV are ignored. 1 Writing a 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	**Enable Prescalar by 8.** 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]	**Clock Divider Bits** — 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). Please refer to Section 4.4.3.1, "Writing the FCDIV Register" 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.

	7	6	5	4	3	2	1	0
R	KEYEN		0	0	0	0	SEC	
W								
Reset	F	F	0	0	0	0	F	F

Figure 4-4. 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 4.2.2) during the reset sequence, indicated by F in Figure 4-4.

Table 4-8. FOPT Field Descriptions

Field	Description
7–6 KEYEN	**Backdoor Key Security Enable Bits** — The KEYEN[7:6] 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
5–2	Reserved, must be cleared.
1–0 SEC	**Flash Security Bits** — 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.

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

Figure 4-5. 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 4-9. 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.

	7	6	5	4	3	2	1	0
R W	FPS							FPOPEN
Reset	F	F	F	F	F	F	F	F

Figure 4-6. Flash Protection Register (FPROT)

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

During the reset sequence, the FPROT register is loaded from the flash protection byte in the flash configuration field (see Section 4.2.2). To change the flash protection that will be 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 will result in a protection violation error and the FPVIOL flag will be 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 4-10. FPROT Field Descriptions

Field	Description
7–1 FPS	**Flash Protection Size** — With FPOPEN set, the FPS bits determine the size of the protected flash address range as shown in Table 4-11.
0 FPOPEN	Flash Protection Open 0 Flash array fully protected. 1 Flash array protected address range determined by FPS bits.

Table 4-11. Flash Protection Address Range

FPS	FPOPEN	Protected Address Range Relative to Flash Array Base	Protected Size (KB)
—	0	0x0_0000–0x3_FFFF	256
0x00	1	0x0_0000–0x2_F7FF	254
0x01		0x0_0000–0x2_EFFF	252
0x02		0x0_0000–0x2_E7FF	250
...		...	...
0x3E		0x0_0000–0x2_07FF	130
0x3F		0x0_0000–0x1_FFFF	128
0x40		0x0_0000–0x1_F7FF	126
0x41		0x0_0000–0x1_EFFF	124
0x42		0x0_0000–0x1_E7FF	122
0x43		0x0_0000–0x1_DFFF	120

Table 4-11. Flash Protection Address Range (continued)

FPS	FPOPEN	Protected Address Range Relative to Flash Array Base	Protected Size (KB)
0x44	1	0x0_0000–0x1_D7FF	118
0x45		0x0_0000–0x1_CFFF	116
0x46		0x0_0000-0x1_C7FF	114
0x47		0x0_0000–0x1_BFFF	112
...		...	...
0x5B		0x0_0000–0x1_1FFF	72
0x5C		0x0_0000–0x1_17FF	70
0x5D		0x0_0000–0x1_0FFF	68
0x5E		0x0_0000–0x1_07FF	66
0x5F		0x0_0000–0x0_FFFF	64
0x60		0x0_0000–0x0_F7FF	62
0x61		0x0_0000–0x0_EFFF	60
0x62		0x0_0000–0x0_E7FF	58
0x63		0x0_0000–0x0_DFFF	56
...		...	...
0x77		0x0_0000–0x0_3FFF	16
0x78		0x0_0000–0x0_37FF	14
0x79		0x0_0000–0x0_2FFF	12
0x7A		0x0_0000–0x0_27FF	10
0x7B		0x0_0000–0x0_1FFF	8
0x7C		0x0_0000–0x0_17FF	6
0x7D		0x0_0000–0x0_0FFF	4
0x7E		0x0_0000–0x0_07FF	2
0x7F		No Protection	0

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 4-7. Flash Status Register (FSTAT)

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

Table 4-12. FSTAT Field Descriptions

Field	Description
7 FCBEF	**Command Buffer Empty Flag** — The FCBEF flag indicates that the command buffer is empty so that a new command write sequence can be started when performing burst programming. Writing a 0 to the FCBEF flag has no effect on FCBEF. Writing a 0 to FCBEF after writing an aligned address to the flash array memory, but before FCBEF is cleared, will abort a command write sequence and cause the FACCERR flag to be set. Writing a 0 to FCBEF outside of a command write sequence will not set the FACCERR flag. The FCBEF flag is cleared by writing a 1 to FCBEF. 0 Command buffers are full. 1 Command buffers are ready to accept a new command.
6 FCCF	**Command Complete Flag** — The FCCF flag indicates that there are no more commands pending. The FCCF flag is cleared when FCBEF is cleared and set automatically upon 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. 0 Command in progress. 1 All commands are completed.
5 FPVIOL	**Protection Violation Flag** —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 a 0 to the FPVIOL flag has no effect on FPVIOL. The FPVIOL flag is cleared by writing a 1 to FPVIOL. While FPVIOL is set, it is not possible to launch a command or start a command write sequence. 0 No protection violation detected. 1 Protection violation has occurred.
4 FACCERR	**Access Error Flag** — 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 Section 4.4.3.2, “Command Write Sequence”), issuing an illegal flash command (see Section 4.4.2.6, “Flash Command Register (FCMD)”), or the execution of a CPU STOP instruction while a command is executing (FCCF = 0). Writing a 0 to the FACCERR flag has no effect on FACCERR. The FACCERR flag is cleared by writing a 1 to FACCERR. While FACCERR is set, it is not possible to launch a command or start a command write sequence. 0 No access error detected. 1 Access error has occurred.
3	Reserved, must be cleared.
2 FBLANK	**Flag Indicating the Erase Verify Operation Status** — 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. 0 Flash block verified as not erased. 1 Flash block verified as erased.
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	FCMD						
W								
Reset	0	0	0	0	0	0	0	0

Figure 4-8. Flash Command Register (FCMD)

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

Table 4-13. FCMD Field Descriptions

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

4.4.3 Flash Command Operations

Flash command operations are used to 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, the user is required to write the FCDIV register to divide the bus clock down to within the 150 kHz to 200 kHz range.

If we define:

- FCLK as the clock of the flash timing control block
- INT(x) as taking the integer part of x (e.g. INT(4.323) = 4)

then the FCDIV[PRDIV8, FDIV] bits must be set as described in Figure 4-9.

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:

$$(200 - 195) \div 200 = 3\% \qquad \textit{Eqn. 4-1}$$

CAUTION

Program and erase command execution time will increase 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 erasing of the flash memory cells.

If the FCDIV register is written, the FDIVLD bit is set 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 will not execute and FSTAT[FACCERR] will be set.

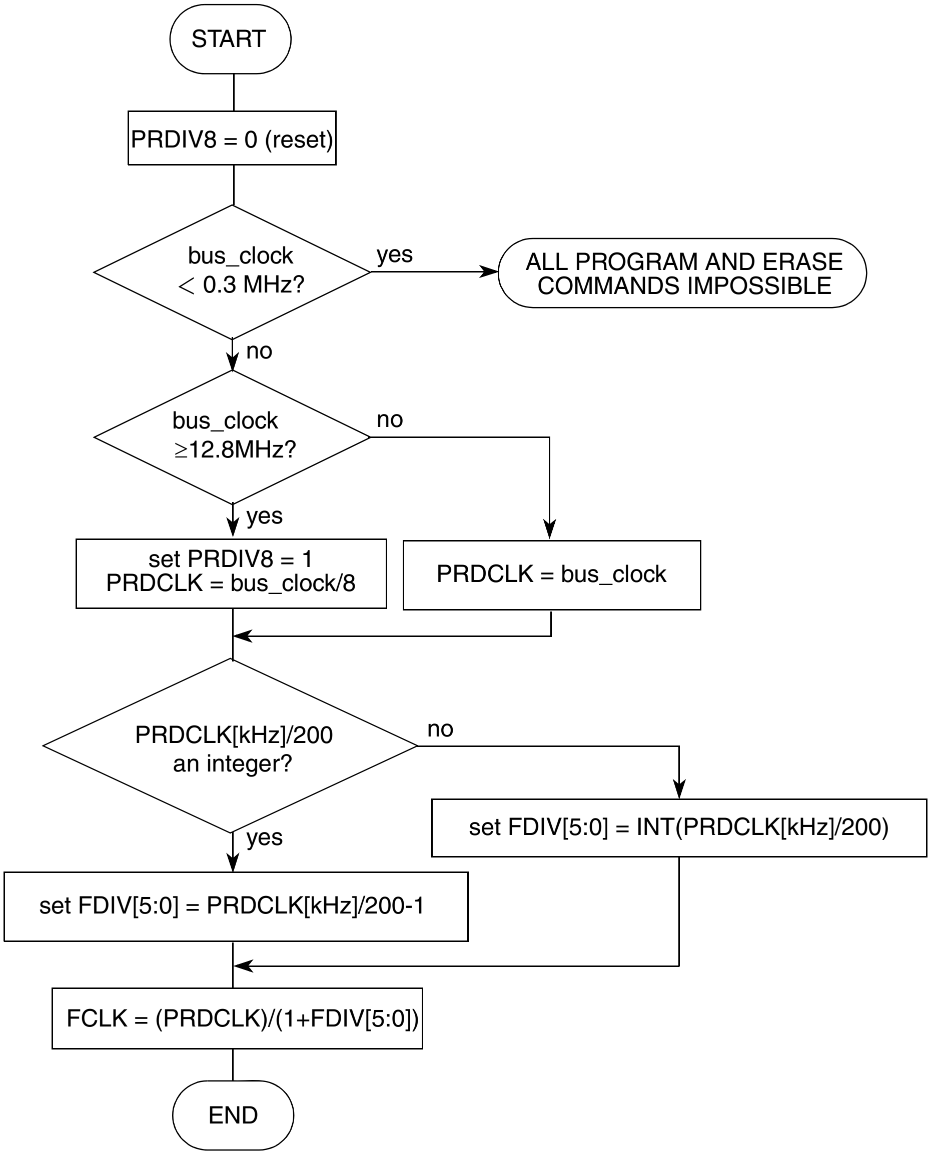


Figure 4-9. Determination Procedure for PRDIV8 and FDIV Bits

4.4.3.2 Command Write Sequence

The flash command controller is used to supervise 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 which 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 a 1 to FCBEF to launch the command.

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

NOTE

At least 3 bus cycles are required between writing any flash registers and performing an array-write operation. At least one bus cycle is required between two consecutive array write access, otherwise the ACCERR flag will be set.

4.4.4 Flash Commands

Table 4-14 summarizes the valid flash commands along with the effects of the commands on the flash block.

Table 4-14. 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 will set upon 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 will verify that the entire flash array memory is erased.

An example flow to execute the erase verify operation is shown in Figure 4-10. 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 will be ignored.
2. Write the erase verify command, 0x05, to the FCMD register.

3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the erase verify command.

After launching the erase verify command, the FCCF flag in the FSTAT register will set 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. Upon completion of the erase verify operation, the FBLANK flag in the FSTAT register will be set 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 will terminate and the FBLANK flag in the FSTAT register will remain clear.

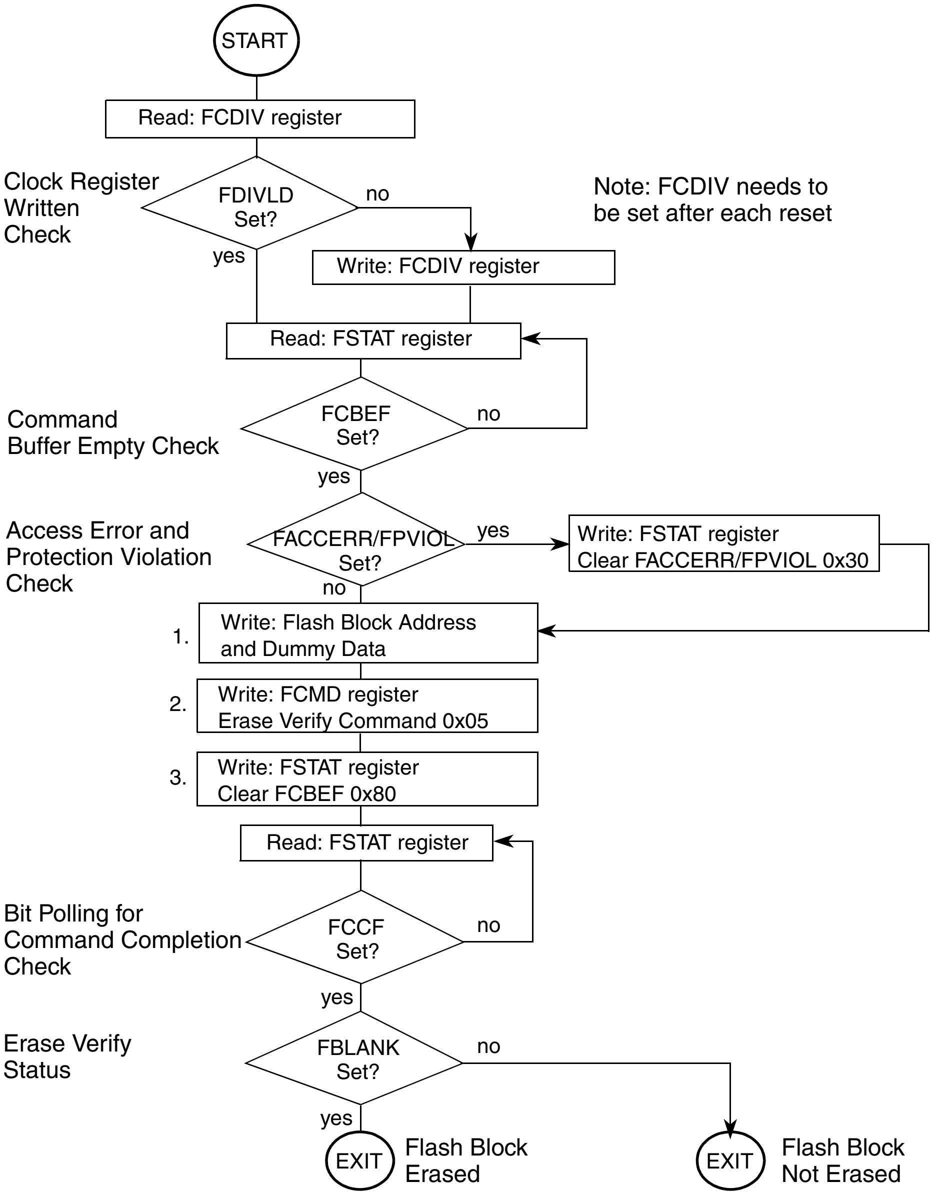


Figure 4-10. Example Erase Verify Command Flow

4.4.4.2 Program Command

The program operation will program a previously erased address in the flash memory using an embedded algorithm.

An example flow to execute the program operation is shown in Figure 4-11. 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 will be programmed to the address written.
2. Write the program command, 0x20, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 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 will set and the program command will not launch. Once the program command has successfully launched, the FCCF flag in the FSTAT register will set after the program operation has completed.

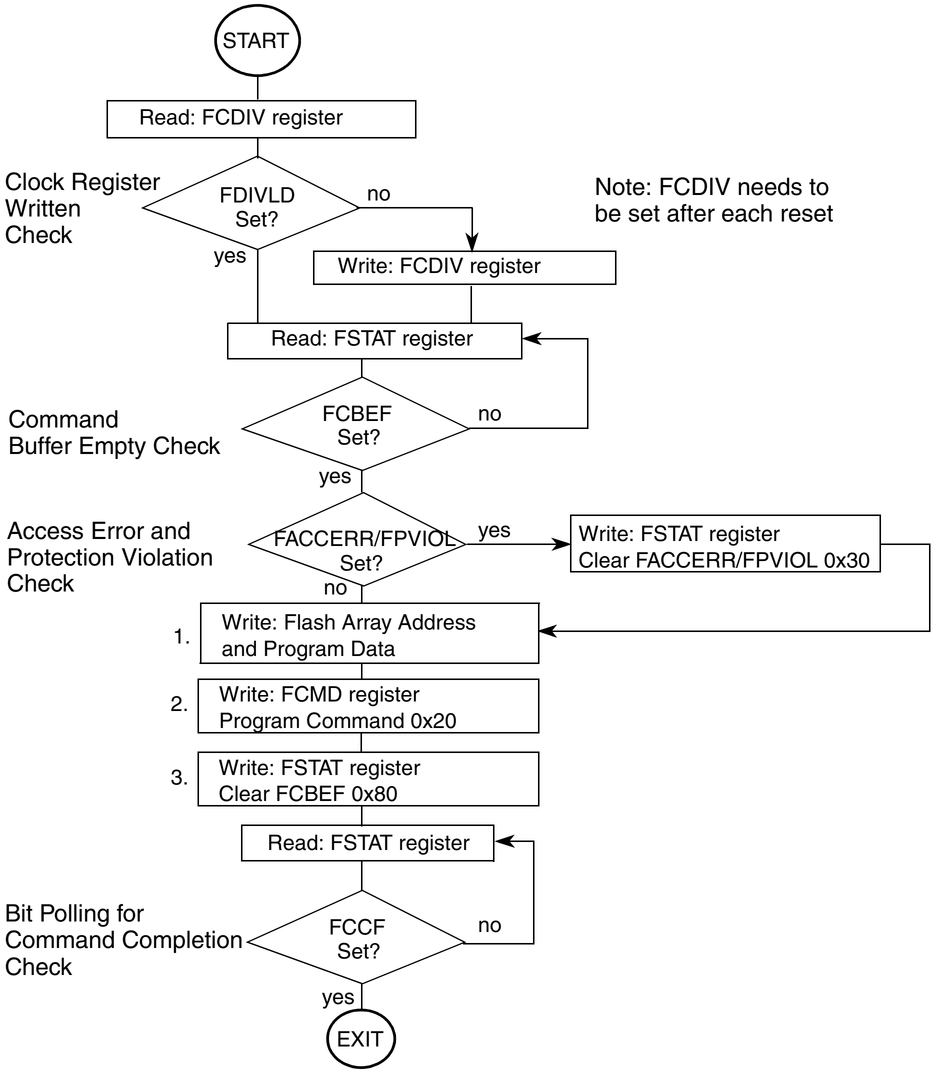


Figure 4-11. Example Program Command Flow

4.4.4.3 Burst Program Command

The burst program operation will program 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 as 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 4-12. 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 will be 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 a 1 to FCBEF to launch the program burst command.
4. After the FCBEF flag in the FSTAT register returns to a 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 will set and the burst program command will not launch. Once the burst program command has successfully launched, the FCCF flag in the FSTAT register will set 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.

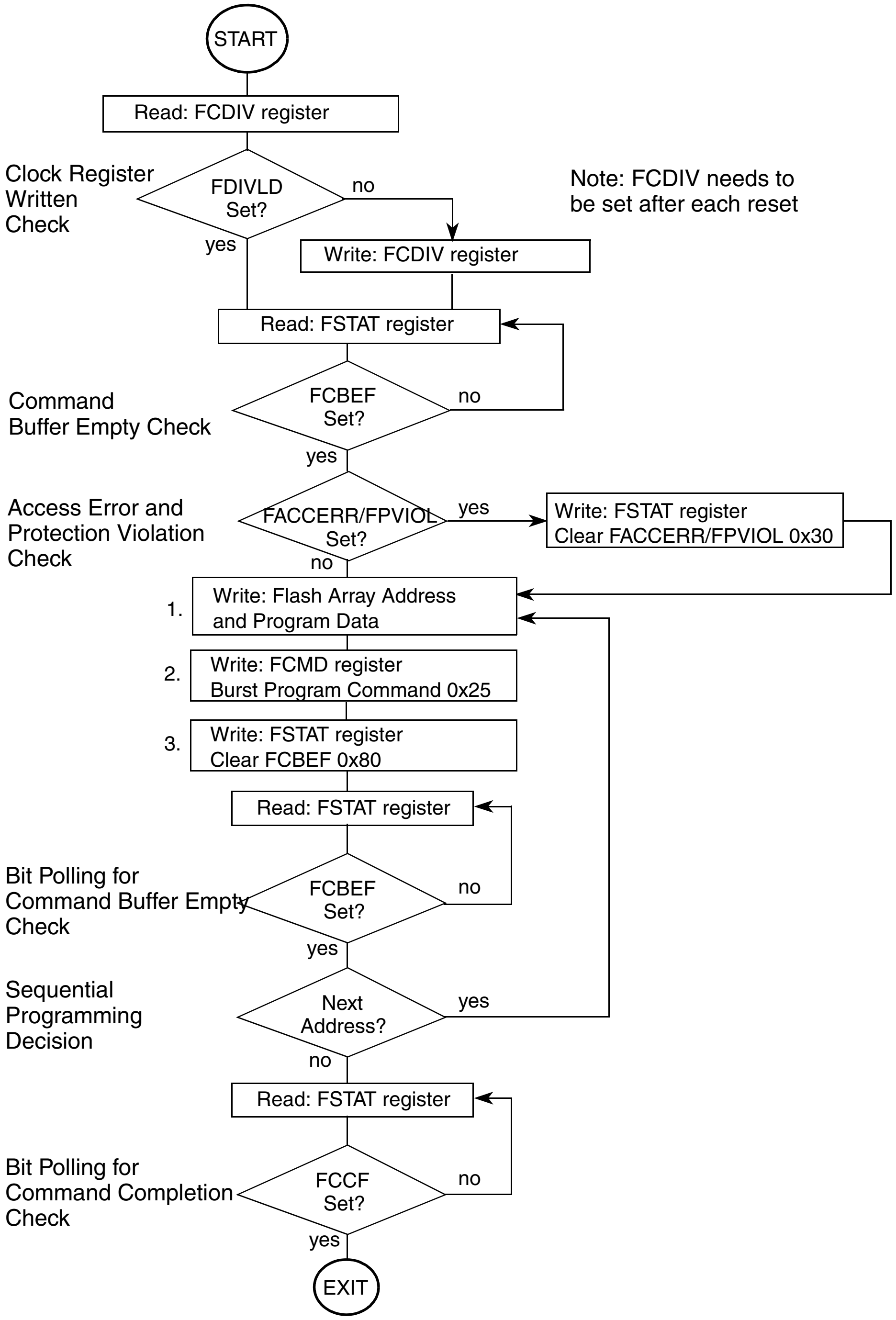

START
Read: FCDIV register
Clock Register Written Check
FDIVLD Set?
no
yes
Note: FCDIV needs to be set after each reset
Write: FCDIV register
Read: FSTAT register
Command Buffer Empty Check
FCBEF Set?
no
yes
Access Error and Protection Violation Check
FACCERR/FPVIOL Set?
yes
no
Write: FSTAT register Clear FACCERR/FPVIOL 0x30
1.
Write: Flash Array Address and Program Data
2.
Write: FCMD register Burst Program Command 0x25
3.
Write: FSTAT register Clear FCBEF 0x80
Read: FSTAT register
Bit Polling for Command Buffer Empty Check
FCBEF Set?
no
yes
Sequential Programming Decision
Next Address?
yes
no
Read: FSTAT register
Bit Polling for Command Completion Check
FCCF Set?
no
yes
EXIT

Figure 4-12. Example Burst Program Command Flow

4.4.4.4 Sector Erase Command

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

An example flow to execute the sector erase operation is shown in Figure 4-13. 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 a 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 will set and the sector erase command will not launch. Once the sector erase command has successfully launched, the FCCF flag in the FSTAT register will set after the sector erase operation has completed.

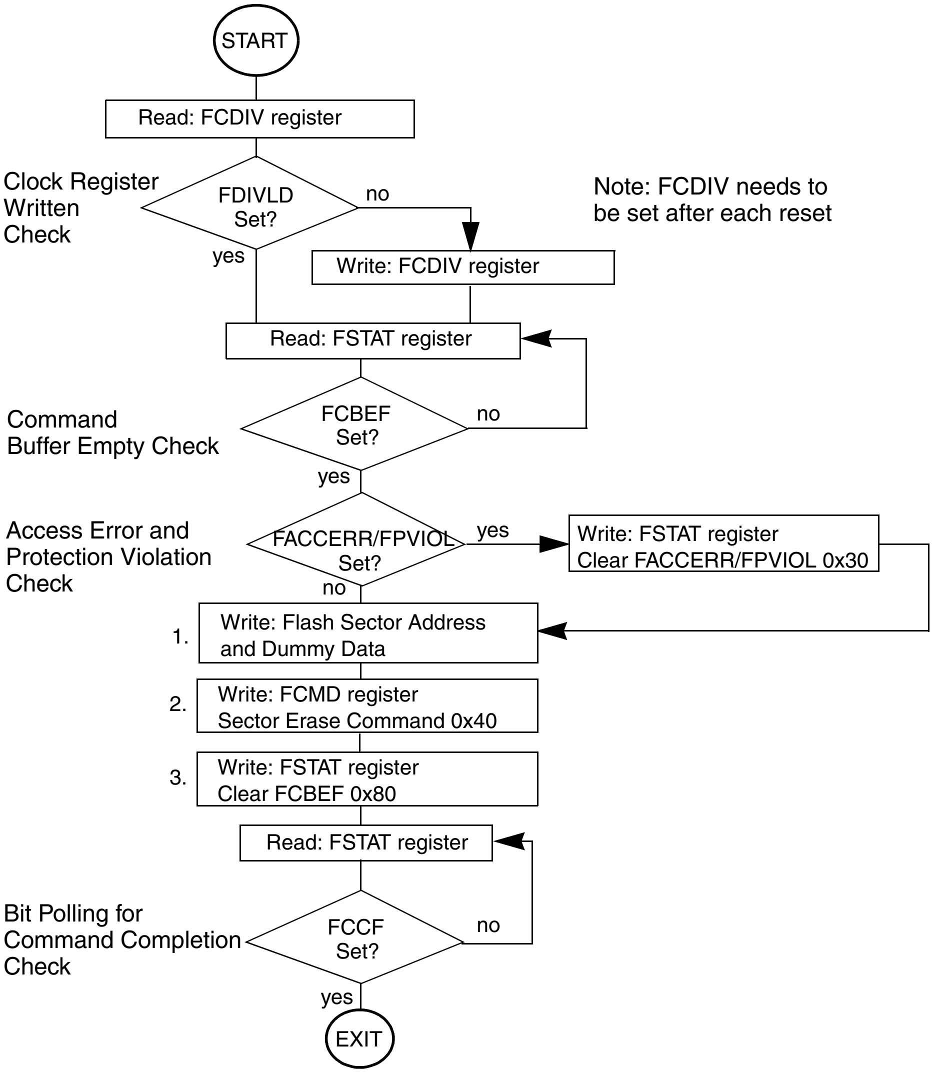

START
Read: FCDIV register
Clock Register Written Check
FDIVLD Set?
no
yes
Note: FCDIV needs to be set after each reset
Write: FCDIV register
Read: FSTAT register
Command Buffer Empty Check
FCBEF Set?
no
yes
Access Error and Protection Violation Check
FACCERR/FPVIOL Set?
yes
no
Write: FSTAT register
Clear FACCERR/FPVIOL 0x30
1.
Write: Flash Sector Address and Dummy Data
2.
Write: FCMD register
Sector Erase Command 0x40
3.
Write: FSTAT register
Clear FCBEF 0x80
Read: FSTAT register
Bit Polling for Command Completion Check
FCCF Set?
no
yes
EXIT

Figure 4-13. Example Sector Erase Command Flow

4.4.4.5 Mass Erase Command

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

An example flow to execute the mass erase operation is shown in Figure 4-14. 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 will be ignored.
2. Write the mass erase command, 0x41, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 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 will set and the mass erase command will not launch. Once the mass erase command has successfully launched, the FCCF flag in the FSTAT register will set after the mass erase operation has completed.

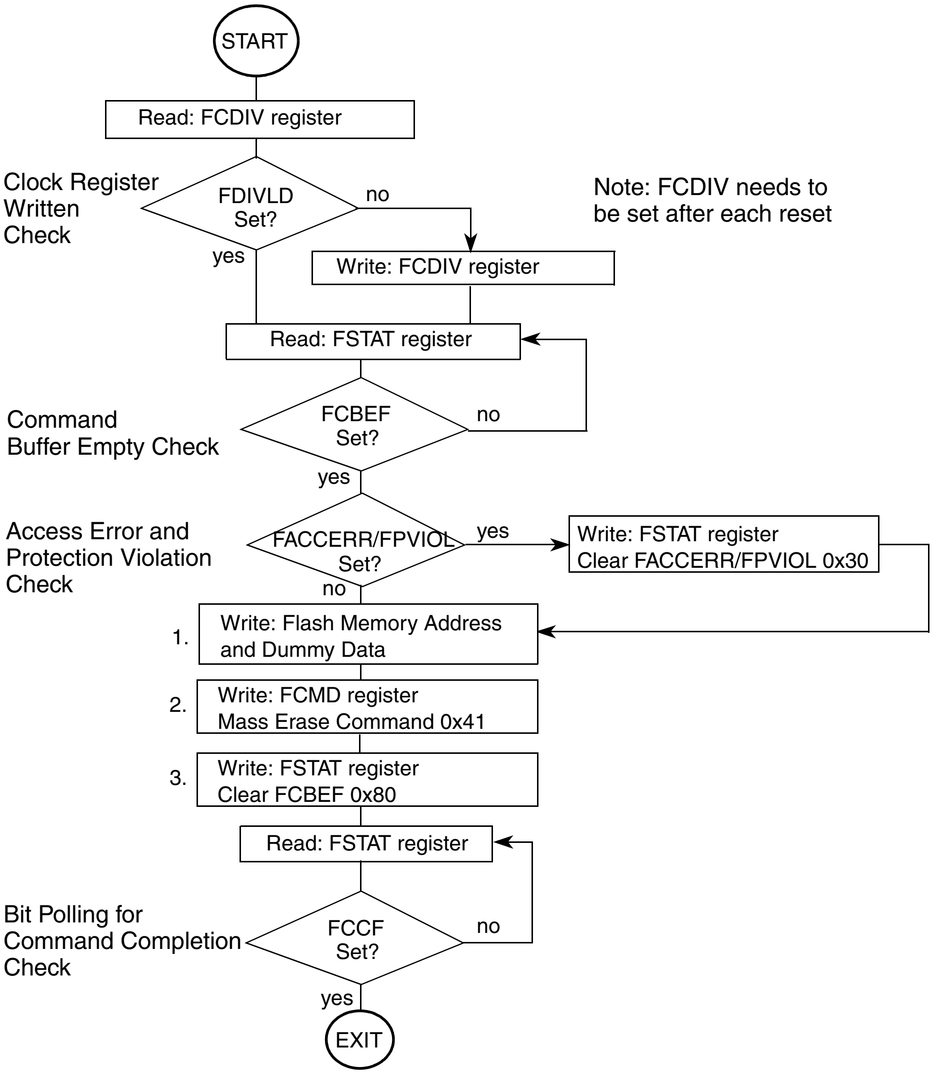

START
Read: FCDIV register
Clock Register Written Check
FDIVLD Set?
no
yes
Note: FCDIV needs to be set after each reset
Write: FCDIV register
Read: FSTAT register
Command Buffer Empty Check
FCBEF Set?
no
yes
Access Error and Protection Violation Check
FACCERR/FPVIOL Set?
yes
no
Write: FSTAT register Clear FACCERR/FPVIOL 0x30
1.
Write: Flash Memory Address and Dummy Data
2.
Write: FCMD register Mass Erase Command 0x41
3.
Write: FSTAT register Clear FCBEF 0x80
Read: FSTAT register
Bit Polling for Command Completion Check
FCCF Set?
no
yes
EXIT

Figure 4-14. Example Mass Erase Command Flow

4.4.5 Illegal Flash Operations

4.4.5.1 Flash Access Violations

The FACCERR flag will be set 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 clear.
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 a 0 to the FCBEF flag in the FSTAT register to abort a command write sequence.

The FACCERR flag will also be set 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 will not be 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 will return invalid data and the FACCERR flag will not be set.

If the FACCERR flag is set in the FSTAT register, the user 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 will be set 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, the user 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 will be completed.

4.4.6.2 Stop Mode

If a command is active (FCCF = 0) when the MCU enters stop mode, the operation will be aborted and, if the operation is program or erase, the flash array data being programmed or erased may be corrupted and the FCCF and FACCERR flags will be set. If active, the high voltage circuitry to the flash array will immediately be switched off when entering stop mode. Upon exit from stop mode, the FCBEF flag is set and any buffered command will not be launched. 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 the user does not use the STOP instruction during program or erase operations.

Active commands will 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 4-14 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 4.2.2, “Flash Module Reserved Memory Locations”.;ll;

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 will cause 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 4.2.2). 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 will be 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, reads of the flash memory will 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 will cause the security state machine to lock, leaving the MCU in the secured state. A reset of the MCU will cause 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 will lock 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 the keys written are all 0s or all 1s.
4. If the KEYACC bit does not remain set while the keys are written.
5. If any of the keys is 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 will be unsecured. Once the MCU is unsecured, the flash security byte can be programmed to the unsecure state, if desired.

In the unsecure state, the user has full control of the contents of the backdoor keys by programming the associated addresses in the flash configuration field (see Section 4.2.2).

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).

The MCU cannot be unsecured 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 4.2.2)
- Flash protection byte (see Section 4.2.2 and Section 4.4.2.4)
- Flash nonvolatile byte (see Section 4.2.2)
- Flash security byte (see Section 4.2.2 and Section 4.4.2.2)

4.4.8.2 Reset While Flash Command Active

If a reset occurs while any flash command is in progress, that command will be immediately aborted. 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 MCF51AC256 *Series Data Sheet*, order number MCF51AC256.

4.5 Security

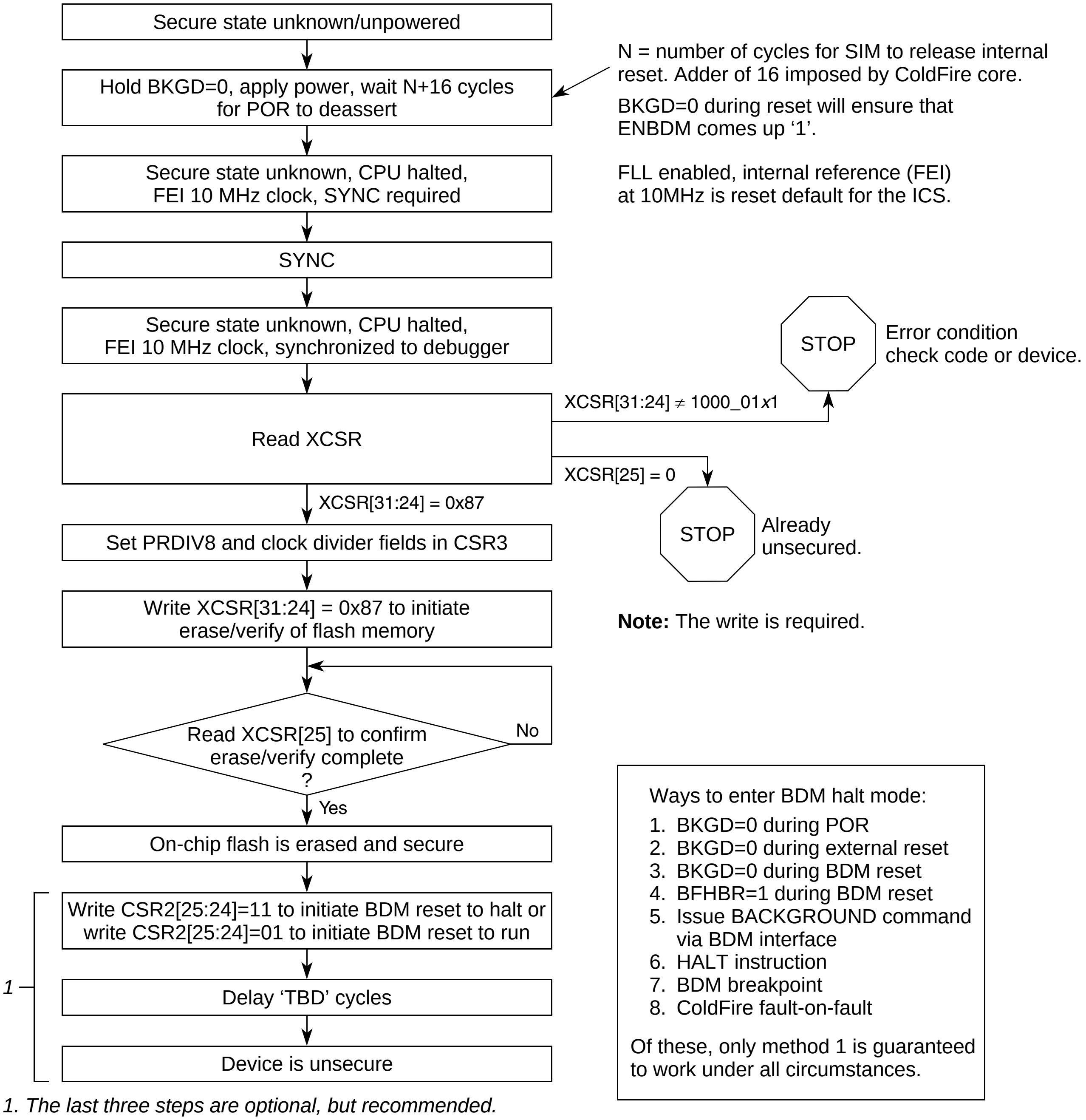
This device includes circuitry to prevent unauthorized access to the contents of flash and RAM memory. When security is engaged, BDM access is restricted to the upper byte of the ColdFire CSR, XCSR, and CSR2 registers. RAM, flash memory, peripheral registers and most of the CPU register set are not available 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 (SEC[1:0]) 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. A user engages security by programming the NVOPT location which can be done at the same time the flash memory is programmed. The 1:0 state engages security and the other three combinations disengage 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 initialized to one if the device is secured, zero otherwise.

A user 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 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 will unsecure devices via an alternate BDM-based methodology shown in Figure 4-15. 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 can also be used, but may not work under all circumstances.


Secure state unknown/unpowered
Hold BKGD=0, apply power, wait N+16 cycles for POR to deassert
Secure state unknown, CPU halted, FEI 10 MHz clock, SYNC required
SYNC
Secure state unknown, CPU halted, FEI 10 MHz clock, synchronized to debugger
Read XCSR
XCSR[31:24] ≠ 1000_01x1
STOP
Error condition check code or device.
XCSR[25] = 0
STOP
Already unsecured.
XCSR[31:24] = 0x87
Set PRDIV8 and clock divider fields in CSR3
Write XCSR[31:24] = 0x87 to initiate erase/verify of flash memory
Read XCSR[25] to confirm erase/verify complete ?
No
Yes
On-chip flash is erased and secure
Write CSR2[25:24]=11 to initiate BDM reset to halt or write CSR2[25:24]=01 to initiate BDM reset to run
Delay 'TBD' cycles
Device is unsecure
1

N = number of cycles for SIM to release internal reset. Adder of 16 imposed by ColdFire core.

BKGD=0 during reset will ensure that ENBDM comes up '1'.

FLL enabled, internal reference (FEI) at 10MHz is reset default for the ICS.

Note: The write is required.

Ways to enter BDM halt mode:

1. BKGD=0 during POR
2. BKGD=0 during external reset
3. BKGD=0 during BDM reset
4. BFHBR=1 during BDM reset
5. Issue BACKGROUND command via BDM interface
6. HALT instruction
7. BDM breakpoint
8. ColdFire fault-on-fault

Of these, only method 1 is guaranteed to work under all circumstances.

1. The last three steps are optional, but recommended.

Figure 4-15. Procedure for Clearing Security via the BDM Port

Chapter 5
Resets, Interrupts, and General System Control

5.1 Introduction

This chapter discusses basic reset and interrupt mechanisms and the various sources of reset and interrupt on MCF51AC256 series microcontrollers. Some interrupt sources from peripheral modules are discussed in greater detail within other sections of this document. This chapter gathers basic information about all reset and interrupt sources in one place for easy reference. A few reset and interrupt sources, including the computer operating properly (COP) watchdog are not part of on-chip peripheral systems with their own chapters.

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 most recent reset
- Separate interrupt vector for most modules (reduces polling overhead) (see Table 5-1)

5.3 Microcontroller Reset

Resetting the microcontroller provides a way to start processing from an 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 MCF51AC256 series microcontrollers have the following sources for reset:

- Power-on reset (POR)
- External pin reset (PIN)
- Computer operating properly (COP) timer
- Illegal opcode detect (ILOP)
- Illegal address detect (ILAD)
- Low-voltage detect (LVD)
- Clock generator (MCG) loss of clock reset (LOC)
- Background debug forced reset

Each of these sources, with the exception of the background debug forced reset, has an associated bit in the system reset status register (SRS).

5.3.1 Computer Operating Properly (COP) Watchdog

The COP watchdog forces a system reset when the application software fails to execute as expected. To prevent a system reset from the COP timer (when it is enabled), application software must reset the COP counter periodically. If the application program gets lost and fails to reset the COP counter before it times out, a system reset will be generated to force the system back to an known starting point.

After any reset, the SOPT[COPE] bit is set enabling the COP watchdog (see Section 5.9.3, "System Options (SOPT) Register," for additional information). If the COP watchdog is not used in an application, it can be disabled by clearing COPE. The COP counter is reset by writing any value to the address of SRS. This write does not affect the data in the read-only SRS. Instead, the act of writing to this address is decoded and sends a reset signal to the COP counter.

The SOPT2[COPCLKS] bit selects the clock source used for the COP timer (see Section 5.9.9, "System Options 2 (SOPT2) Register," for additional information). The clock source options are either the bus clock or an internal 1 kHz LPO clock source. With each clock source, there is an associated short and long time-out controlled by the SOPT[COPT] bit. Table 5-1 summaries the control functions of the COPCLKS and COPT bits. The COP watchdog defaults to operation from the bus clock source and the associated long time-out (2^8 cycles).

Table 5-1. COP Configuration Options

Control Bits		Clock Source	COP Overflow Count
COPCLKS	COPT		
0	0	~1 kHz LPO Clock	2^5 cycles (32 ms)[1]
0	1	~1 kHz LPO Clock	2^8 cycles (256 ms)[1]
1	0	Bus Clock	2^{13} cycles
1	1	Bus Clock	2^{18} cycles

[1] Values shown in this column are based on t_{LPO} = 1 ms.

Even if the application uses the default reset settings of COPE, COPCLKS, and COPT, the user must write to the write-once SOPT[1] and SOPT2 registers during reset initialization to lock in the settings. That way, they cannot be changed accidentally if the application program gets lost. The initial writes to SOPT and SOPT2 will reset the COP counter.

The COP counter is reset by writing any value to the address of SRS during the selected timeout period. Writes do not affect the data in the read-only SRS. As soon as the write sequence is done, the COP timeout period restarts. If the program fails to do this during the time-out period, the microcontroller will reset.

The write to SRS that services (clears) the COP counter must not be placed in an interrupt service routine (ISR) because the ISR could continue to be executed periodically even if the main application program fails.

In the CPU halt state, the COP counter will not increment.

[1] The WAITE bit in SOPT can be written multiple times. Other bits are write-once.

When the bus clock source is selected, the COP counter does not increment while the system is in stop mode. The COP counter resumes as soon as the microcontroller exits stop mode.

When the 1 kHz LPO clock source is selected, the COP counter is re-initialized to zero upon entry to stop mode. The COP counter begins from zero after the microcontroller exits stop mode.

5.3.2 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). The user may set CPUCR[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.

5.3.3 Illegal Address Detect (ILAD)

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

5.4 Interrupts & Exceptions

The interrupt architecture of ColdFire utilizes a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing 7 levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. The processor samples for active interrupt requests once per instruction by comparing the encoded priority level against a 3-bit interrupt mask value (I) contained in bits 10:8 of the processor's status register (SR). If the priority level is greater than the SR[I] field at the sample point, the processor suspends normal instruction execution and initiates interrupt exception processing.

Level 7 interrupts are treated as non-maskable and edge-sensitive within the processor, while levels 1-6 are treated as level-sensitive and may be masked depending on the value of the SR[I] field. For correct operation, the ColdFire processor requires that, once asserted, the interrupt source remain asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU does the following tasks in order:

1. enters supervisor mode,
2. disables trace mode,
3. uses the vector provided by the INTC when the interrupt was signaled (if CPUCR[IACK] = 0) or explicitly fetches an 8-bit vector from the INTC (if CPUCR[IACK] = 1).

This byte-sized operand fetch during exception processing is known as the interrupt acknowledge (IACK) cycle. The fetched data provides an index into the exception vector table which contains up to 256 addresses (depending upon the specific device), each pointing to the beginning of a specific exception service routine.

In particular, the first 64 exception vectors are reserved for the processor to handle reset, error conditions (access, address), arithmetic faults, system calls, etc. Vectors 64–255 are reserved for interrupt service routines. The MCF51JM128 series microcontrollers support 37 peripheral interrupt sources and additional seven software interrupt sources. These are mapped into the standard seven ColdFire interrupt levels, with up to 9 levels of prioritization within a given level by the V1 ColdFire interrupt controller. See Table 5-2 for details.

Once the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are two longwords in length, and contain 32 bits of vector and status register data, along with the 32-bit program counter value of the instruction that was interrupted. After the exception stack frame is stored in memory, the processor accesses the 32-bit pointer from the exception vector table using the vector number as the offset, and then jumps to that address to begin execution of the service routine. After the status register is stored in the exception stack frame, the SR[I] mask field is set to the level of the interrupt being acknowledged, effectively masking that level and all lower values while in the service routine.

All ColdFire processors guarantee that the first instruction of the service routine is executed before interrupt sampling is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled, if required. Optionally, the processor can be configured to automatically raise the mask level to 7 for any interrupt during exception processing by setting CPUCR[IME] = 1.

During the execution of the service routine, the appropriate actions must be performed on the peripheral to negate the interrupt request.

For more information on exception processing, see the *ColdFire Programmer's Reference Manual* and Chapter 13, "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 stop mode and system clocks are shut down, a separate asynchronous path will be used, so 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 in order for the IRQ pin to act as the interrupt request (IRQ) input. As an IRQ input, the user 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 which 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. If the user desires 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}.

NOTE

The voltage measured on the internally pullup IRQ pin will not be pulled to V_{DD}. The internal gates connected to this pin are pulled to V_{DD}. The IRQ pullup must not be used to pull up components external to the microcontroller.

5.4.2 Interrupt Vectors, Sources, and Local Masks

Table 5-2. Reset and Interrupt Vectors

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 (see Table 7-6)
64	0x100	7,mid	Next	IRQ_pin
65	0x104	7,3	Next	Low_voltage_detect
66	0x108	7,2	Next	MCG_lock
67	0x10C	5,6	Next	FTM1_ch0
68	0x110	5,5	Next	FTM1_ch1
69	0x114	5,4	Next	FTM1_ch2
70	0x118	5,mid	Next	FTM1_ch3
71	0x11C	5,3	Next	FTM1_ch4
72	0x120	5,2	Next	FTM1_ch5
73	0x124	5,1	Next	FTM1_ovfl
74	0x128	4,6	Next	FTM2_ch0
75	0x12C	4,5	Next	FTM2_ch1
76	0x130	4,4	Next	FTM2_ch2
77	0x134	4,mid	Next	FTM2_ch3
78	0x138	4,3	Next	FTM2_ch4

Table 5-2. Reset and Interrupt Vectors (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment
79	0x13C	4,2	Next	FTM2_ch5
80	0x140	4,1	Next	FTM2_ovfl
81	0x144	3,7	Next	SPI1
82	0x148	3,3	Next	SCI1_err
83	0x14C	3,2	Next	SCI1_rx
84	0x150	3,1	Next	SCI1_tx
85	0x154	2,7	Next	SCI2_err
86	0x158	2,6	Next	SCI2_rx
87	0x15C	2,5	Next	SCI2_tx
88	0x160	2,4	Next	KBI1
89	0x164	2,3	Next	ADC1
90	0x168	2,2	Next	IIC1
91	0x16C	2,1	Next	RTI
92	0x170	1,6	Next	TPM3_ch0
93	0x174	1,5	Next	TPM3_ch1
94–102	0x178–0x198		—	Reserved; unused for V1
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
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	1,4	Next	TPM3_ovfl
111	0x1BC	1,3	Next	SPI2
112	0x1C0	5,7	Next	FTM1_fault
113	0x1C4	4,7	Next	FTM2_fault
114	0x1C8	3,6	Next	MSCAN_wakeup
115	0x1CC	3,5	Next	MSCAN_errors
116	0x1D0	3,4	Next	MSCAN_Rx
117	0x1D4	3,mid	Next	MSCAN_Tx

Table 5-2. Reset and Interrupt Vectors (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment
118	0x1D8	1,2	Next	ACMP1
119	0x1DC	1,1	Next	ACMP2
120–255	0x1E0–0x3FC		—	Reserved; unused for V1

The CPU configuration register (CPUCR) within the supervisor programming model will allow the users to determine if specific ColdFire exception conditions are to generate a normal exception or a system reset. The default state of the CPUCR will force a system reset for any of the exception types listed in Table 5-3.

Table 5-3. ColdFire Exception Vector Table[1]

Vector	Exception	Reset Disabled via CPUCR	Reported using SRS
64-108	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	—
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

Table 5-3. ColdFire Exception Vector Table[1] (continued)

Vector	Exception	Reset Disabled via CPUCR	Reported using SRS
9	Trace	N/A	—
8	Privileged Violation	CPUCR[30]	ilop
4	Illegal instruction	CPUCR[30]	ilop[2]
3	Address error	CPUCR[31]	ilad
2	Access error	CPUCR[31]	ilad
n/a	Flt-on-Flt Halt	CPUCR[31]	ilad

[1] Exception vector numbers not appearing in this table are not applicable to the V1 core and are "reserved".

[2] 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 MCF51AC256 series microcontrollers include a system to protect memory contents against low voltage conditions and control microcontroller system states during supply voltage variations. The system is composed of a power-on reset (POR) circuit and a LVD circuit with a user-selectable trip voltage, either 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 mode unless the LVDSE bit is set. If both LVDE and LVDSE are set when the STOP instruction is processed, the device will enter 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 will cause a reset condition. As the supply voltage rises, the LVD circuit will hold the microcontroller in reset until the supply has risen above the LVD low threshold, V_{LVDL}. Both of the POR bit and the LVD bit 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 to 1. The low voltage detection threshold is determined by the LVDV bit. After an LVD reset has occurred, the LVD system will hold the microcontroller in reset until the supply voltage rises above the low voltage detection threshold. The LVD bit in the SRS register is set following an LVD reset or POR.

5.5.3 LVD Interrupt Operation

When a low voltage condition is detected and the LVD circuit is configured using SPMSC1 for interrupt operation (LVDE set, LVDIE set, and LVDRE clear), then LVDF in SPMSC1 will be set and an LVD interrupt request will occur. The LVDF bit is cleared by writing a 1 to the LVDACK bit in SPMSC1.

5.5.4 Low-Voltage Warning (LVW) Operation

The LVD system has a low voltage warning flag (LVWF) to indicate to the user that the supply voltage is approaching, but is above, the LVD voltage. LVWF is cleared by writing a 1 to the SPMSC2[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 Real-Time Interrupt (RTI)

The real-time interrupt function can be used to generate periodic interrupts. The RTI can accept two sources of clocks, the 1 kHz low power oscillator or the external oscillator if available. The 1 kHz internal clock source is completely independent of any bus clock source and is used only by the RTI module and, on some MCUs, the COP watchdog. The external clock source is the external reference of the MCG. To use the external oscillator as the RTI clock, it must be available and active. The RTICLKS bit in SRTISC is used to select the RTI clock source.

Either RTI clock source can be used when the MCU is in run, wait or stop3 mode. When using the external oscillator in stop3, it must be enabled in stop (OSCSTEN = 1). Only the internal 1 kHz clock source can be selected to wake the MCU from stop2 mode.

The SRTISC register includes a read-only status flag, a write-only acknowledge bit, and a 3-bit control value (RTIS2:RTIS1:RTIS0) used to disable the clock source to the real-time interrupt or select one of seven wakeup periods. The RTI has a local interrupt enable, RTIE, to allow masking of the real-time interrupt. The RTI can be disabled by writing each bit of RTIS to zeroes, and no interrupts will be generated. See Section 5.9.6, “System Real-Time Interrupt Status and Control Register (SRTISC),” for detailed information about this register.

5.7 MCLK Output

The PTC2 pin is shared with the MCLK clock output. Setting the pin enable bit, MPE, causes the PTC2 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 PTC2 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 will be driven low. The slew rate and drive strength for the pin are controlled by PTCSE2 and PTCDS2, respectively. The maximum clock output frequency is limited if slew rate control is enabled, see the data sheet for pin rise and fall times with slew rate enabled.

5.8 Peripheral Clock Gating

The MCF51AC256 series microcontrollers include a clock gating system to manage the bus clock sources to the individual peripherals. Using this system, the user can enable or disable the bus clock to each peripheral 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 will be enabled. For lowest possible run or wait currents, user software must disable the clock source to any peripheral not in use. The actual clock will be enabled or disabled immediately following the write to the clock gating control registers (SCGC1, SCGC2). 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 and SCGC2 registers.

5.9 Register Definition

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 4-2 and Table 4-3 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 SOPT and SPMSC2 registers are related to modes of operation. Although brief descriptions of these bits are provided here, the related functions are discussed in greater detail in Chapter 3, “Modes of Operation.”

5.9.1 Interrupt Request Status and Control Register (IRQSC)

This direct-page register includes status and control bits which are used to configure the IRQ function, report status, and acknowledge IRQ events.

	7	6	5	4	3	2	1	0
R	0	IRQPDD	IRQEDG	IRQPE	IRQF	0	IRQIE	IRQMOD
W						IRQACK		
Reset	0	0	0	0	0	0	0	0

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

Table 5-4. IRQSC Register Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 IRQPDD	**Interrupt Request (IRQ) Pull Device Disable**— This read/write control bit is used to disable 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 = 1. 1 IRQ pull device disabled if IRQPE = 1.

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

Field	Description
5 IRQEDG	**Interrupt Request (IRQ) Edge Select** — This read/write control bit is used to select the polarity of edges or levels on the IRQ pin that cause IRQF to be set. The IRQMOD control bit determines whether the IRQ pin is sensitive to both edges and levels or only edges. When IRQEDG = 1 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	**IRQ Pin Enable** — 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	**IRQ Flag** — This read-only status bit indicates whether an interrupt request event has occurred. 0 No IRQ request. 1 IRQ event detected.
2 IRQACK	**IRQ Acknowledge** — This write-only bit is used to acknowledge 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	**IRQ Interrupt Enable** — 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 = 1.
0 IRQMOD	**IRQ Detection Mode** — This read/write control bit selects either 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.

5.9.2 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 will be set. Writing any value to this register address clears the COP watchdog timer without affecting the contents of this register. The reset state of these bits depends on what caused the microcontroller to reset.

	7	6	5	4	3	2	1	0
R	POR	PIN	COP	ILOP	ILAD	LOC	LVD	0
W	Writing any value to SRS address clears COP watchdog timer.							
POR:	1	0	0	0	0	0	1	0
LVD:	u	0	0	0	0	0	1	0
Any other reset:	0	Note[1]	Note[1]	Note[1]	Note[1]	0	0	0

[1] Any of these reset sources that are active at the time of reset entry will cause the corresponding bit(s) to be set; bits corresponding to sources that are not active at the time of reset entry will be cleared.

Figure 5-2. System Reset Status (SRS)

Table 5-5. SRS Register Field Descriptions

Field	Description
7 POR	**Power-On Reset** — 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	**External Reset Pin** — 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.
5 COP	**Computer Operating Properly (COP) Watchdog** — Reset was caused by the COP watchdog timer timing out. This reset source can be blocked by SOPT[COPE] = 0. 0 Reset not caused by COP timeout. 1 Reset caused by COP timeout.
4 ILOP	**Illegal Opcode** — 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	**Illegal Address** — 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] = 0. When CPUCR[ARD] = 1, 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 LOC	**Loss-of-Clock Reset —** Reset was caused by a loss of external clock. 0 Reset not caused by a loss of external clock. 1 Reset caused by a loss of external clock.
1 LVD	**Low Voltage Detect** — If the LVDRE bit is set and the supply drops below the LVD trip voltage, an LVD reset will occur. This bit is also set by POR. 0 Reset not caused by LVD trip or POR. 1 Reset caused by LVD trip or POR.

5.9.3 System Options (SOPT) Register

This high-page register has three write-once bits and one write-anytime bit. For the write-once bits, only the first write after reset is honored. 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. SOPT must be written to 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	COPE[1]	COPT[1]	STOPE[1]	WAITE	0	0	0	0
W								
Reset:	1	1	0	1	0	0	0	0

[1] This bit can be written only one time after reset. Additional writes are ignored.

Figure 5-3. System Options (SOPT) Register

Table 5-6. SOPT Register Field Descriptions

Field	Description
7 COPE	**COP Watchdog Enable** — This write-once bit selects whether the COP watchdog is enabled. 0 COP watchdog timer disabled. 1 COP watchdog timer enabled (force reset on timeout).
6 COPT	**COP Watchdog Timeout** — This write-once bit selects the timeout period of the COP. COPT along with SOPT2[COPCLKS] defines the COP timeout period. 0 Short timeout period selected. 1 Long timeout period selected.
5 STOPE	**Stop Mode Enable** — This write-once bit is used to enable stop mode. If both 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 CPUCR[IRD].
4 WAITE	**WAIT Mode Enable** — This write-anytime bit is used to enable WAIT mode. If both 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 CPUCR[IRD].
3–0	Reserved, must be cleared.

5.9.4 System MCLK Control Register (SMCLK)

This register is used to control the MCLK clock output.

	7	6	5	4	3	2	1	0
R	0	0	0	MPE	0	MCSEL		
W								
Reset	0	0	0	0	0	0	0	0

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

Table 5-7. SMCLK Register Field Descriptions

Field	Description
4 MPE	**MCLK Pin Enable** — This bit is used to enable the MCLK function. 0 MCLK output disabled. 1 MCLK output enabled on PTC2 pin.
2–0 MCSEL	**MCLK Divide Select** — These bits are used to select the divide ratio for the MCLK output according to the formula below 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 Equation 5-1.

$$\text{MCLK frequency} = \text{Bus clock frequency} \div (2 * \text{MCSEL}) \qquad \textit{Eqn. 5-1}$$

5.9.5 System Device Identification Register (SDIDH, SDIDL)

These high-page read-only registers are included so host development systems can 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.

	7	6	5	4	3	2	1	0
R					ID11	ID10	ID9	ID8
W								
Reset:	—	—	—	—	1	1	0	0

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

Table 5-8. SDIDH Register Field Descriptions

Field	Description
7–4 Reserved	**Reserved. Reading these bits will result in an indeterminate value; writes have no effect.**
3–0 ID[11:8]	**Part Identification Number** — Each derivative in the ColdFire family has a unique identification number. The MCF51AC256 series microcontrollers are hard coded to the value 0xC1B. See also ID bits in Table 5-9.

	7	6	5	4	3	2	1	0
R	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
W								
Reset:	0	0	0	1	1	0	1	1

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

Table 5-9. SDIDL Register Field Descriptions

Field	Description
7–0 ID[7–0]	**Part Identification Number** — Each derivative in the ColdFire family has a unique identification number. The MCF51AC256 series microcontrollers are hard coded to the value 0xC1B. See also ID bits in Table 5-8.

5.9.6 System Real-Time Interrupt Status and Control Register (SRTISC)

This register contains one read-only status flag, one write-only acknowledge bit, three read/write delay selects, and one unimplemented bit, which always read 0.

	7	6	5	4	3	2	1	0
R	RTIF	0	RTICLKS	RTIE	0	RTIS2	RTIS1	RTIS0
W		RTIACK						
Reset	0	0	0	0	0	0	0	0

Figure 5-7. System RTI Status and Control Register (SRTISC)

Table 5-10. SRTISC Register Field Descriptions

Field	Description
7 RTIF	**Real-Time Interrupt Flag** — This read-only status bit indicates the periodic wakeup timer has timed out. 0 Periodic wakeup timer not timed out. 1 Periodic wakeup timer timed out.
6 RTIACK	**Real-Time Interrupt Acknowledge** — This write-only bit is used to acknowledge real-time interrupt request (write 1 to clear RTIF). Writing 0 has no meaning or effect. Reads always return logic 0.
5 RTICLKS	**Real-Time Interrupt Clock Select** — This read/write bit selects the clock source for the real-time interrupt. 0 Real-time interrupt request clock source is internal 1 kHz oscillator. 1 Real-time interrupt request clock source is external clock.
4 RTIE	**Real-Time Interrupt Enable** — This read/write bit enables real-time interrupts. 0 Real-time interrupts disabled. 1 Real-time interrupts enabled.
2–0 RTIS[2:0]	**Real-Time Interrupt Delay Selects** — These read/write bits select the wakeup delay for the RTI. The clock source for the real-time interrupt is a self-clocked source which oscillates at about 1 kHz, is independent of other MCU clock sources. Using external clock source the delays will be crystal frequency divided by value in RTIS2:RTIS1:RTIS0. See Table 5-11.

Table 5-11. Real-Time Interrupt Frequency

RTIS2:RTIS1:RTIS0	1 kHz LPO Clock Source Delay[1]	Using External Clock Source Delay (Crystal Frequency)
0:0:0	Disable periodic wakeup timer	Disable periodic wakeup timer
0:0:1	8 ms	divide by 256
0:1:0	32 ms	divide by 1024
0:1:1	64 ms	divide by 2048
1:0:0	128 ms	divide by 4096
1:0:1	256 ms	divide by 8192
1:1:0	512 ms	divide by 16384
1:1:1	1.024 s	divide by 32768

[1] Normal values shown in this column are based on f_{RTI} = 1 kHz.

5.9.7 System Power Management Status and Control 1 Register (SPMSC1)

This high-page register contains status and control bits to support the low voltage detection function, and to enable the bandgap voltage reference for use by the ADC module. To configure the low voltage detect trip voltage, see Table 5-13 for the SPMSC2[LVDV] bit description.

SPMSC1 is not reset when exiting from stop2.

	7	6	5	4	3	2	1[1]	0
R	LVDF	0	LVDIE	LVDRE[2]	LVDSE[2]	LVDE[2]	0	BGBE
W		LVDACK						
Reset:	0	0	0	1	1	1	0	0

[1] Bit 1 is a reserved bit that must always be written to 0.

[2] This bit can be written only once after reset. Additional writes are ignored.

Figure 5-8. System Power Management Status and Control 1 Register (SPMSC1)

Table 5-12. SPMSC1 Register Field Descriptions

Field	Description
7 LVDF	**Low-Voltage Detect Flag** — Provided LVDE = 1, this read-only status bit indicates a low-voltage detection event.
6 LVDACK	**Low-Voltage Detect Acknowledge** — This write-only bit is used to acknowledge low voltage detection errors (write 1 to clear LVDF). Reads always return 0.
5 LVDIE	**Low-Voltage Detect Interrupt Enable** — This bit enables hardware interrupt requests for LVDF. 0 Hardware interrupt disabled (use polling). 1 Request a hardware interrupt when LVDF = 1.
4 LVDRE	**Low-Voltage Detect Reset Enable** — This write-once bit enables LVDF events to generate a hardware reset (provided LVDE = 1). 0 LVDF does not generate hardware resets. 1 Force a microcontroller reset when LVDF = 1.
3 LVDSE	**Low-Voltage Detect Stop Enable** — Provided LVDE = 1, this read/write bit determines whether the low-voltage detection function operates when the microcontroller is in stop mode. 0 Low-voltage detection disabled during stop mode. 1 Low-voltage detection enabled during stop mode.
2 LVDE	**Low-Voltage Detect Enable** — This write-once bit enables low-voltage detection logic and qualifies the operation of other bits in this register. 0 LVD logic disabled. 1 LVD logic enabled.
0 BGBE	**Bandgap Buffer Enable** — This bit enables an internal buffer for the bandgap voltage reference for use by the ADC module on one of its internal channels or as a voltage reference for ACMP module. 0 Bandgap buffer disabled. 1 Bandgap buffer enabled.

5.9.8 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 as well as configure the stop mode behavior of the microcontroller. See Section 3.6, “Wait Mode,” and Section 3.7, “Stop Modes,” for more information.

SPMSC2 is not reset when exiting from stop2.

	7	6	5	4	3	2	1	0
R	LVWF	0	LVDV[1]	LVWV	PPDF	0	0	PPDC[2]
W	▨	LVWACK			▨	PPDACK	▨	
POR Reset:	0[3]	0	0	0	0	0	0	0
LVD Reset:	0[2]	0	U	U	0	0	0	0
Any other Reset:	0[2]	0	U	U	0	0	0	0

▨ = Unimplemented or Reserved U = Unaffected by reset

[1] This bit can be written only once after power-on reset. Additional writes are ignored.

[2] This bit can be written only once after reset. Additional writes are ignored.

[3] LVWF will be set in the case when V_{Supply} transitions below the trip point or after reset and V_{Supply} is already below V_{LVW}.

Figure 5-9. System Power Management Status and Control 2 Register (SPMSC2)

Table 5-13. SPMSC2 Register Field Descriptions

Field	Description
7 LVWF	**Low-Voltage Warning Flag** — 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	**Low-Voltage Warning Acknowledge** — The LVWACK bit is the low-voltage warning acknowledge. Writing a 1 to LVWACK clears LVWF to a 0 if a low voltage warning is not present.
5 LVDV	**Low-Voltage Detect Voltage Select** — The LVDV bit selects the LVD trip point voltage (V_{LVD}). 0 Low trip point selected ($V_{LVD} = V_{LVDL}$). 1 High trip point selected ($V_{LVD} = V_{LVDH}$).
4 LVWV	**Low-Voltage Warning Voltage Select** — 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	**Partial Power Down Flag** — The PPDF bit indicates that the MCU has exited the stop2 mode. 0 Not stop2 mode recovery. 1 Stop2 mode recovery.

Table 5-13. SPMSC2 Register Field Descriptions (continued)

Field	Description
2 PPDACK	**Partial Power Down Acknowledge** — Writing a 1 to PPDACK clears the PPDF bit.
0 PPDC	**Partial Power Down Control** — The write-once PPDC bit controls whether stop2 or stop3 mode is selected. 0 Stop3 mode enabled. 1 Stop2, partial power down, mode enabled.

5.9.9 System Options 2 (SOPT2) Register

This high-page register contains bits to configure MCU specific features on the MCF51AC256 series devices.

	7	6	5	4	3	2	1	0
R W	COPCLKS[1]	SPI2FE	SPI1FE	ACIC2	TPMCCFG	ACIC1	ADHWTS	
Reset:	1	0	0	0	1	0	0	0

▨ = Unimplemented or Reserved

[1] This bit can be written onlyonce after reset. Additional writes are ignored.

Figure 5-10. System Integration Module Options Register 2 (SOPT2)

Table 5-14. SOPT2 Register Field Descriptions

Field	Description
7 COPCLKS	**COP Watchdog Clock Select** — This write-once bit selects the clock source of the COP watchdog. 0 Internal 1 kHz LPO clock is source to COP. 1 Bus clock is source to COP.
6 SPI2FE	**SPI2 Ports Input Filter Enable** — This bit enables the input filter on the SPI2 port pins. 0 Input filter disabled (allows for higher maximum baud rate). 1 Input filter enabled (eliminates noise and restricts maximum baud rate).
5 SPI1FE	**SPI1 Ports Input Filter Enable** — This bit enables the input filter on the SPI1 port pins. 0 Input filter disabled (allows for higher maximum baud rate). 1 Input filter enabled (eliminates noise and restricts maximum baud rate).
4 ACIC2	**Analog Comparator 2 to Input Capture Enable**— This bit connects the output of ACMP2 to TPM3 input channel 0. See Chapter 8, "Analog Comparator (ACMPV3)," and Chapter 21, "Timer/PWM Module (TPMV3)," for more details on this feature. 0 ACMP2 output not connected to TPM3 input channel 0. 1 ACMP2 output connected to TPM3 input channel 0.
3 TPMCCFG	**TPM Clock Configuration** — 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 5-14. SOPT2 Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
ACIC1</td><td><b>Analog Comparator 1 to Input Capture Enable</b>— This bit connects the output of ACMP1 to FTM1 input channel 0. See Chapter 8, “Analog Comparator (ACMPV3),” and Chapter 11, “FlexTimer Module (FTMV1),” for more details on this feature.
0 ACMP output not connected to FTM1 input channel 0.
1 ACMP output connected to FTM1 input channel 0.</td></tr>
<tr><td>1–0
ADHWTS</td><td><b>ADC Hardware Trigger Select</b>— These bits select the source of the hardware trigger for the ADC module.
<table>
<tr><th>ADHWTS</th><th>Trigger Source</th></tr>
<tr><td><b>0:0</b></td><td>RTI</td></tr>
<tr><td><b>0:1</b></td><td>ACMP1</td></tr>
<tr><td>1:0</td><td>FTM1</td></tr>
<tr><td>1:1</td><td>Reserved</td></tr>
</table></td></tr>
</table>

5.9.10 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*x*, ADC, CAN, IIC, and SCI*x* modules. Gating off the clocks to unused peripherals is used to reduce the microcontroller’s run and wait currents. See Section 5.8, “Peripheral Clock Gating,” for more information.

	7	6	5	4	3	2	1	0
R W	TMP3	FTM2	FTM1	ADC	CAN	IIC	SCI2	SCI1
Reset:	1	1	1	1	1	1	1	1

Figure 5-11. System Clock Gating Control 1 Register (SCGC1)

Table 5-15. SCGC1 Register Field Descriptions

Field	Description
7 TPM3	**TPM3 Clock Gate Control** — 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	**FTM2 Clock Gate Control** — 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	**FTM1 Clock Gate Control** — 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	**ADC Clock Gate Control** — 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.

Table 5-15. SCGC1 Register Field Descriptions (continued)

Field	Description
3 CAN	**CAN Clock Gate Control** — This bit controls the clock gate to the CAN module. 0 Bus clock to the CAN module is disabled. 1 Bus clock to the CAN module is enabled.
2 IIC	**IIC Clock Gate Control** — 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	**SCI2 Clock Gate Control** — 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	**SCI1 Clock Gate Control** — 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.

5.9.11 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, KBI, ACMP, RTI, and SPI*x* modules. Gating off the clocks to unused peripherals is used to reduce the microcontroller’s run and wait currents. See Section 5.8, “Peripheral Clock Gating,” for more information.

	7	6	5	4	3	2	1	0
R W	CRC	FLS	IRQ	KBI*x*	ACMP*x*	RTI	SPI2	SPI1
Reset:	1	1	1	1	1	1	1	1

Figure 5-12. System Clock Gating Control 2 Register (SCGC2)

Table 5-16. SCGC2 Register Field Descriptions

Field	Description
7 CRC	**CRC Clock Gate Control** — 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	**FTSR Clock Gate Control** — 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	**IRQ Clock Gate Control** — 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 KBI*x*	**KBI Clock Gate Control** — This bit controls the clock gate to both of the KBI modules. 0 Bus clock to the KBI modules is disabled. 1 Bus clock to the KBI modules is enabled.

Table 5-16. SCGC2 Register Field Descriptions (continued)

Field	Description
3 ACMP*x*	**ACMP Clock Gate Control** — This bit controls the clock gate to both of the ACMP modules. 0 Bus clock to the ACMP modules is disabled. 1 Bus clock to the ACMP modules is enabled.
2 RTI	**RTI Clock Gate Control** — This bit controls the bus clock gate to the RTI module. Only the bus clock is gated; the clock resource is not controller by this bit. 0 Bus clock to the RTI module is disabled. 1 Bus clock to the RTI module is enabled.
1 SPI2	**SPI2 Clock Gate Control** — 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	**SPI1 Clock Gate Control** — 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.

Chapter 6
Parallel Input/Output Control

This chapter explains software controls related to parallel input/output (I/O) and pin control. The MCF51AC256 series 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/toggle functions which are integrated as part of the ColdFire core itself to improve edge resolution on those pins. See Section 6.3, “V1 ColdFire Rapid GPIO Functionality,” and Chapter 17, “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 1-1. The peripheral modules have priority over the general-purpose I/O functions so that 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 (PT*x*DD*n* = 0). The pin control functions for each pin are configured as follows: slew rate control enabled (PT*x*SE*n* = 1), low drive strength selected (PT*x*DSn = 0), and internal pullups disabled (PT*x*PE*n* = 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 Port Data and Data Direction

Reading and writing of parallel I/Os are performed through the port data registers. The direction, either input or output, is controlled through the port data direction registers. The parallel I/O port function for an individual pin is illustrated in the block diagram shown in Figure 6-1.

The data direction control bit (PT*x*DD*n*) determines whether the output buffer for the associated pin is enabled, and also controls the source for port data register reads. The input buffer for the associated pin is always enabled unless the pin is enabled as an analog function or is an output-only pin.

When a shared digital function is enabled for a pin, the output buffer is controlled by the shared function. However, the data direction register bit will continue to control the source for reads of the port data register.

When a shared analog function is enabled for a pin, both the input and output buffers are disabled. A value of 0 is read for any port data bit where the bit is an input (PT*x*DD*n* = 0) and the input buffer is disabled. In general, whenever a pin is shared with both an alternate digital function and an analog function, the analog function has priority such that if both the digital and analog functions are enabled, the analog function controls the pin.

Write to the port data register before changing the direction of a port pin to become an output. This ensures that the pin will not be driven momentarily with an old data value that happened to be in the port data register.

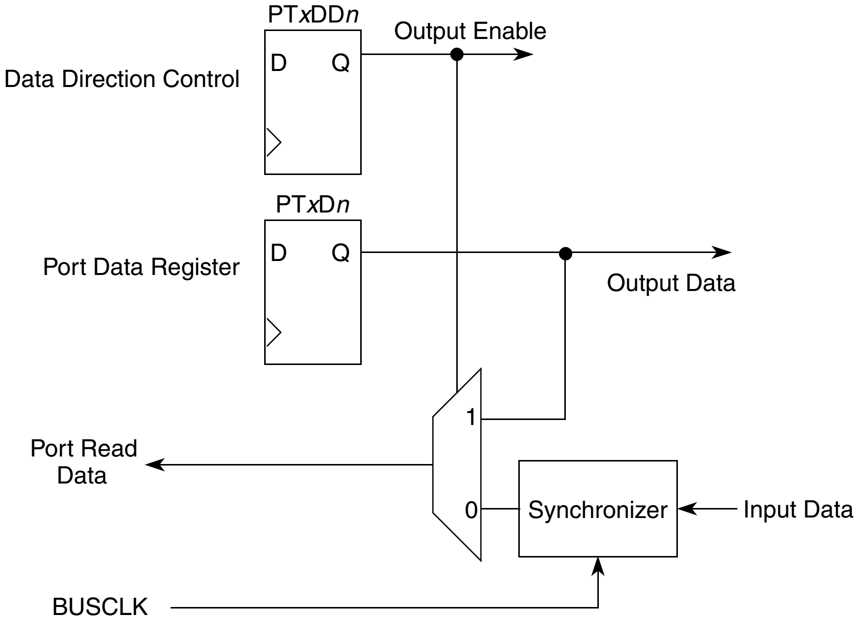


Figure 6-1. Classic Parallel I/O Block Diagram

6.2 Pullup, Slew Rate, and Drive Strength

A set of high-page registers are used to control pullups, slew rate, and drive strength for the pins. They may also be used in conjunction with the peripheral functions on these pins. These registers are associated with the parallel I/O ports, but operate independently of the parallel I/O registers.

6.2.1 Port Internal Pullup Enable

An internal pullup device can be enabled for each port pin by setting the corresponding bit in the pullup enable register (PT*x*PE*n*). The pullup device is disabled if the pin is configured as an output by the parallel I/O control logic or any shared peripheral function regardless of the state of the corresponding pullup enable register bit. The pullup device is also disabled if the pin is controlled by an analog function.

6.2.2 Port Slew Rate Enable

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate control register (PT*x*SE*n*). When enabled, slew control limits the rate at which an output can transition in order to reduce EMC emissions. Slew rate control has no effect on pins that are configured as inputs.

6.2.3 Port Drive Strength Select

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive strength select register (PT*x*DSn). When high drive is selected, a pin is capable of sourcing and sinking greater current. Even though every I/O pin can be selected as high drive, the user must ensure that the total current source and sink limits for the MCU are not exceeded. Drive strength selection affects the DC behavior of I/O pins. However, the AC behavior is also affected. High drive allows a pin to drive a greater load with the same switching speed as a low drive enabled pin into a smaller load. Because of this, the EMC emissions may be affected by enabling pins as high drive.

6.3 V1 ColdFire Rapid GPIO Functionality

The V1 ColdFire core is capable of performing higher speed I/O via its local bus, which does not have latency penalties associated with the on-chip peripheral bus bridge. The rapid GPIO module contains separate set/clear/data registers which are based at address 0x(00)C0_0000. This functionality can be programmed to take priority on ports E and F.

This functionality is further defined in Chapter 17, “Rapid GPIO (RGPIO).”

6.4 Pin Behavior in Stop Modes

Pin behavior following execution of a STOP instruction depends on the stop mode that is entered. An explanation of pin behavior for the various stop modes follows:

- Stop2 mode is a partial power-down mode, whereby I/O latches are maintained in their state as before the STOP instruction was executed (port states are lost and will need to be restored upon exiting stop2). CPU register status and the state of I/O registers must be saved in RAM before the STOP instruction is executed to place the MCU in stop2 mode.
  Upon recovery from stop2 mode, before accessing any I/O, the user must examine the state of the SPMSC2[PPDF] bit. If the PPDF bit is cleared, I/O must be initialized as if a power-on-reset had occurred. If the PPDF bit is set, I/O register states must be restored from the values saved in RAM before the STOP instruction was executed and peripherals may require initialization or restoration to their pre-stop condition. The user must then write a 1 to the SPMSC2[PPDACK] bit. Access to I/O is now permitted again in the user application program.
- In stop3 and stop4 modes, all I/O is maintained because internal logic circuity stays powered. Upon recovery, normal I/O function is available to the user.

6.5 Register Definition

This section provides information about the registers associated with the parallel I/O ports. The data and data direction registers and the keyboard interrupt registers are located in page zero of the memory map. The pullup, slew rate, drive strength, and interrupt control registers are located in the high page section of the memory map.

Refer to tables in Chapter 4, “Memory,” for the absolute address assignments for all parallel I/O and their pin control registers. This section refers to registers and control bits only by their names. A Freescale

Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

6.5.1 Port A Registers

Port A is an 8-bit input/output port and also shares its pins with the CAN, ACMP2, and ADC peripherals. On reset, the pins will default to the input state and under the I/O control. Enabling any of the CAN peripheral will take priority control over the pins. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTA0 shares its pin with TxCAN.
- PTA1 shares its pin with RxCAN.
- PTA2 does not share its pin with any other peripheral.
- PTA3 shares its pin with ACMP2O.
- PTA4 shares its pin with ACMP2–.
- PTA5 shares its pin with ACMP2+.
- PTA6 shares its pin with AD1P16.
- PTA7 shares its pin with AD1P17.

Port A is controlled by the registers listed below.

6.5.1.1 Port A Data Register (PTAD)

	7	6	5	4	3	2	1	0
R W	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-2. Port A Data Register (PTAD)

Table 6-1. PTAD Register Field Descriptions

Field	Description
7–0 PTAD*n*	**Port A Data Register Bits** — For port A pins that are inputs, reads return the logic level on the pin. For port A pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port A pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTAD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups/pulldowns disabled.

6.5.1.2 Port A Data Direction Register (PTADD)

	7	6	5	4	3	2	1	0
R W	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-3. Port A Data Direction Register (PTADD)

Table 6-2. PTADD Register Field Descriptions

Field	Description
7–0 PTADD*n*	**Data Direction for Port A Bits** — These read/write bits control the direction of port A pins and what is read for PTAD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port A bit *n* and PTAD reads return the contents of PTAD*n*.

6.5.1.3 Port A Pull Enable Register (PTAPE)

	7	6	5	4	3	2	1	0
R W	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-4. Internal Pull Enable for Port A Register (PTAPE)

Table 6-3. PTAPE Register Field Descriptions

Field	Description
7–0 PTAPE*n*	**Internal Pull Enable for Port A Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTA pin. For port A pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port A bit *n*. 1 Internal pullup device enabled for port A bit *n*.

6.5.1.4 Port A Slew Rate Enable Register (PTASE)

	7	6	5	4	3	2	1	0
R W	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-5. Slew Rate Enable for Port A Register (PTASE)

Table 6-4. PTASE Register Field Descriptions

Field	Description
7–0 PTASE*n*	**Output Slew Rate Enable for Port A Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port A bit *n*. 1 Output slew rate control enabled for port A bit *n*.

6.5.1.5 Port A Drive Strength Selection Register (PTADS)

	7	6	5	4	3	2	1	0
R W	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-6. Drive Strength Selection for Port A Register (PTADS)

Table 6-5. PTADS Register Field Descriptions

Field	Description
7–0 PTADS*n*	**Output Drive Strength Selection for Port A Bits** — Each of these control bits selects between low and high output drive for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port A bit *n*. 1 High output drive strength selected for port A bit *n*.

6.5.2 Port B Registers

Port B is an 8-bit input/output port and also shares its pins with the TPM3 and the ADC peripherals. On reset, the pins will default to the input state and under the I/O control. Enabling any of the shared peripheral will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTB0 shares its pin with TPM3CH0 and AD1P0.
- PTB1 shares its pin with TPM3CH1 and AD1P1.
- PTB2 shares its pin with AD1P2.
- PTB3 shares its pin with AD1P3.
- PTB4 shares its pin with AD1P4.
- PTB5 shares its pin with AD1P5.
- PTB6 shares its pin with AD1P6.
- PTB7 shares its pin with AD1P7.

Port B is controlled by the registers listed below.

6.5.2.1 Port B Data Register (PTBD)

	7	6	5	4	3	2	1	0
R W	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-7. Port B Data Register (PTBD)

Table 6-6. PTBD Register Field Descriptions

Field	Description
7–0 PTBD*n*	**Port B Data Register Bits** — For port B pins that are inputs, reads return the logic level on the pin. For port B pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port B pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTBD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups/pulldowns disabled.

6.5.2.2 Port B Data Direction Register (PTBDD)

	7	6	5	4	3	2	1	0
R W	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-8. Port B Data Direction Register (PTBDD)

Table 6-7. PTBDD Register Field Descriptions

Field	Description
7–0 PTBDD*n*	**Data Direction for Port B Bits** — These read/write bits control the direction of port B pins and what is read for PTBD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port B bit *n* and PTBD reads return the contents of PTBD*n*.

6.5.2.3 Port B Pull Enable Register (PTBPE)

	7	6	5	4	3	2	1	0
R W	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-9. Internal Pull Enable for Port B Register (PTBPE)

Table 6-8. PTBPE Register Field Descriptions

Field	Description
7–0 PTBPE*n*	**Internal Pull Enable for Port B Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTB pin. For port B pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port B bit *n*. 1 Internal pullup device enabled for port B bit *n*.

6.5.2.4 Port B Slew Rate Enable Register (PTBSE)

	7	6	5	4	3	2	1	0
R W	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-10. Slew Rate Enable for Port B Register (PTBSE)

Table 6-9. PTBSE Register Field Descriptions

Field	Description
7–0 PTBSE*n*	**Output Slew Rate Enable for Port B Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port B bit *n*. 1 Output slew rate control enabled for port B bit *n*.

6.5.2.5 Port B Drive Strength Selection Register (PTBDS)

	7	6	5	4	3	2	1	0
R W	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-11. Drive Strength Selection for Port B Register (PTBDS)

Table 6-10. PTBDS Register Field Descriptions

Field	Description
7–0 PTBDS*n*	**Output Drive Strength Selection for Port B Bits** — Each of these control bits selects between low and high output drive for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port B bit *n*. 1 High output drive strength selected for port B bit *n*.

6.5.3 Port C Registers

Port C is a 7-bit input/output port and also shares its pins with the IIC and SCI2 peripherals and the MCLK output. On reset the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTC0 shares its pin with SCL1.
- PTC1 shares its pin with SDA1.
- PTC2 shares its pin with MCLK.
- PTC3 shares its pin with TxD2.
- PTC4 shares its pin with $\overline{\text{SS2}}$.
- PTC5 shares its pin with RxD2.
- PTC6 shares its pin with FTM2FLT.

Port C is controlled by the registers listed below.

6.5.3.1 Port C Data Register (PTCD)

	7	6	5	4	3	2	1	0
R	0	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-12. Port C Data Register (PTCD)

Table 6-11. PTCD Register Field Descriptions

Field	Description
6–0 PTCD*n*	**Port C Data Register Bits** — For port C pins that are inputs, reads return the logic level on the pin. For port C pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port C pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTCD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.3.2 Port C Data Direction Register (PTCDD)

	7	6	5	4	3	2	1	0
R	0	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-13. Port C Data Direction Register (PTCDD)

Table 6-12. PTCDD Register Field Descriptions

Field	Description
6–0 PTCDD*n*	**Data Direction for Port C Bits** — These read/write bits control the direction of port C pins and what is read for PTCD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port C bit *n* and PTCD reads return the contents of PTCD*n.*

6.5.3.3 Port C Pull Enable Register (PTCPE)

	7	6	5	4	3	2	1	0
R	0	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-14. Internal Pull Enable for Port C Register (PTCPE)

Table 6-13. PTCPE Register Field Descriptions

Field	Description
6–0 PTCPE*n*	**Internal Pull Enable for Port C Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTC pin. For port C pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port C bit *n*. 1 Internal pullup device enabled for port C bit *n*.

6.5.3.4 Port C Slew Rate Enable Register (PTCSE)

	7	6	5	4	3	2	1	0
R	0	PTCSE6	PTCSE5	PTCSE4	PTCSE3	PTCSE2	PTCSE1	PTCSE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-15. Slew Rate Enable for Port C Register (PTCSE)

Table 6-14. PTCSE Register Field Descriptions

Field	Description
6–0 PTCSE*n*	**Output Slew Rate Enable for Port C Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port C bit *n*. 1 Output slew rate control enabled for port C bit *n*.

6.5.3.5 Port C Drive Strength Selection Register (PTCDS)

	7	6	5	4	3	2	1	0
R	0	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-16. Drive Strength Selection for Port C Register (PTCDS)

Table 6-15. PTCDS Register Field Descriptions

Field	Description
6–0 PTCDS*n*	**Output Drive Strength Selection for Port C Bits** — Each of these control bits selects between low and high output drive for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port C bit *n*. 1 High output drive strength selected for port C bit *n*.

6.5.4 Port D Registers

Port D is an 8-bit input/output port and also shares its pins with the FTM1, FTM2, keyboard interrupt, ACMP1 and the ADC peripherals. On reset, the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTD0 shares its pin with AD1P8 and ACMP1+.
- PTD1 shares its pin with AD1P9 and ACMP1–.
- PTD2 shares its pin with KBI1P5, AD1P10 and ACMP1O.
- PTD3 shares its pin with KBI1P6 and AD1P11.
- PTD4 shares its pin with FTM2CLK and AD1P12.
- PTD5 shares its pin with AD1P13.
- PTD6 shares its pin with FTM1CLK and AD1P14.
- PTD7 shares its pin with KBI1P7 and AD1P15.

Port D is controlled by the registers listed below.

6.5.4.1 Port D Data Register (PTDD)

	7	6	5	4	3	2	1	0
R W	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-17. Port D Data Register (PTDD)

Table 6-16. PTDD Register Field Descriptions

Field	Description
7–0 PTDD*n*	**Port D Data Register Bits** — For port D pins that are inputs, reads return the logic level on the pin. For port D pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port D pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTDD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups/pulldowns disabled.

6.5.4.2 Port D Data Direction Register (PTDDD)

	7	6	5	4	3	2	1	0
R W	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-18. Port D Data Direction Register (PTDDD)

Table 6-17. PTDDD Register Field Descriptions

Field	Description
7–0 PTDDD*n*	**Data Direction for Port D Bits** — These read/write bits control the direction of port D pins and what is read for PTDD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port D bit *n* and PTDD reads return the contents of PTDD*n*.

6.5.4.3 Port D Pull Enable Register (PTDPE)

	7	6	5	4	3	2	1	0
R W	PTDPE7	PTDPE6	PTDPE5	PTDPE4	PTDPE3	PTDPE2	PTDPE1	PTDPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-19. Internal Pull Enable for Port D Register (PTDPE)

Table 6-18. PTDPE Register Field Descriptions

Field	Description
7–0 PTDPE*n*	**Internal Pull Enable for Port D Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTD pin. For port D pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port D bit *n*. 1 Internal pullup device enabled for port D bit *n*.

6.5.4.4 Port D Slew Rate Enable Register (PTDSE)

	7	6	5	4	3	2	1	0
R W	PTDSE7	PTDSE6	PTDSE5	PTDSE4	PTDSE3	PTDSE2	PTDSE1	PTDSE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-20. Slew Rate Enable for Port D Register (PTDSE)

Table 6-19. PTDSE Register Field Descriptions

Field	Description
7–0 PTDSE*n*	**Output Slew Rate Enable for Port D Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTD pin. For port D pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port D bit *n*. 1 Output slew rate control enabled for port D bit *n*.

6.5.4.5 Port D Drive Strength Selection Register (PTDDS)

	7	6	5	4	3	2	1	0
R W	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-21. Drive Strength Selection for Port D Register (PTDDS)

Table 6-20. PTDDS Register Field Descriptions

Field	Description
7–0 PTDDS*n*	**Output Drive Strength Selection for Port D Bits** — Each of these control bits selects between low and high output drive for the associated PTD pin. For port D pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port D bit *n*. 1 High output drive strength selected for port D bit *n*.

6.5.5 Port E Registers

Port E is an 8-bit input/output port and also shares its pins with the RPGIO, SPI1, SCI1 and the FTM1 peripherals. On reset the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTE0 shares its pin with RGPIO0 and TxD1.
- PTE1 shares its pin with RGPIO1 and RxD1.
- PTE2 shares its pin with RGPIO2 and FTM1CH0.
- PTE3 shares its pin with RGPIO3 and FTM1CH1.
- PTE4 shares its pin with RGPIO4 and $\overline{\text{SS1}}$.
- PTE5 shares its pin with RGPIO5 and MISO1.
- PTE6 shares its pin with RGPIO6 and MOSI1.
- PTE7 shares its pin with RGPIO7 and SPSCK1.

Port E is controlled by the registers listed below.

6.5.5.1 Port E Data Register (PTED)

	7	6	5	4	3	2	1	0
R W	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
Reset:	0	0	0	0	0	0	0	0

Figure 6-22. Port E Data Register (PTED)

Table 6-21. PTED Register Field Descriptions

Field	Description
7–0 PTED*n*	**Port E Data Register Bits** — For port E pins that are inputs, reads return the logic level on the pin. For port E pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port E pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTED to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.5.2 Port E Data Direction Register (PTEDD)

	7	6	5	4	3	2	1	0
R W	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-23. Port E Data Direction Register (PTEDD)

Table 6-22. PTEDD Register Field Descriptions

Field	Description
7–0 PTEDD*n*	**Data Direction for Port E Bits** — These read/write bits control the direction of port E pins and what is read for PTED reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port E bit *n* and PTED reads return the contents of PTED*n*.

6.5.5.3 Port E Pull Enable Register (PTEPE)

	7	6	5	4	3	2	1	0
R W	PTEPE7	PTEPE6	PTEPE5	PTEPE4	PTEPE3	PTEPE2	PTEPE1	PTEPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-24. Internal Pull Enable for Port E Register (PTEPE)

Table 6-23. PTEPE Register Field Descriptions

Field	Description
7–0 PTEPE*n*	**Internal Pull Enable for Port E Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTE pin. For port E pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port E bit *n*. 1 Internal pullup device enabled for port E bit *n*.

6.5.5.4 Port E Slew Rate Enable Register (PTESE)

	7	6	5	4	3	2	1	0
R W	PTESE7	PTESE6	PTESE5	PTESE4	PTESE3	PTESE2	PTESE1	PTESE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-25. Slew Rate Enable for Port E Register (PTESE)

Table 6-24. PTESE Register Field Descriptions

Field	Description
7–0 PTESE*n*	**Output Slew Rate Enable for Port E Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTE pin. For port E pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port E bit *n*. 1 Output slew rate control enabled for port E bit *n*.

6.5.5.5 Port E Drive Strength Selection Register (PTEDS)

	7	6	5	4	3	2	1	0
R W	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-26. Drive Strength Selection for Port E Register (PTEDS)

Table 6-25. PTEDS Register Field Descriptions

Field	Description
7–0 PTEDS*n*	**Output Drive Strength Selection for Port E Bits** — Each of these control bits selects between low and high output drive for the associated PTE pin. For port E pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port E bit *n*. 1 High output drive strength selected for port E bit *n*.

6.5.6 Port F Registers

Port F is an 8-bit input/output port and also shares its pins with the RGPIO, FTM1 and FTM2 peripherals. On reset, the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTF0 shares its pin with RGPIO8 and FTM1CH2.
- PTF1 shares its pin with RGPIO9 and FTM1CH3.
- PTF2 shares its pin with RGPIO10 and FTM1CH4.
- PTF3 shares its pin with RGPIO11 and FTM1CH5.
- PTF4 shares its pin with RGPIO12 and FTM2CH0.
- PTF5 shares its pin with RGPIO13 and FTM2CH1.
- PTF6 shares its pin with RGPIO14 and FTM1FLT.
- PTF7 shares its pin with RGPIO15.

Port F is controlled by the registers listed below.

6.5.6.1 Port F Data Register (PTFD)

	7	6	5	4	3	2	1	0
R W	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-27. Port F Data Register (PTFD)

Table 6-26. PTFD Register Field Descriptions

Field	Description
7–0 PTFD*n*	**Port F Data Register Bits** — For port F pins that are inputs, reads return the logic level on the pin. For port F pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port F pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTFD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.6.2 Port F Data Direction Register (PTFDD)

	7	6	5	4	3	2	1	0
R W	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-28. Port F Data Direction Register (PTFDD)

Table 6-27. PTFDD Register Field Descriptions

Field	Description
7–0 PTFDD*n*	**Data Direction for Port F Bits** — These read/write bits control the direction of port F pins and what is read for PTFD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port F bit *n* and PTFD reads return the contents of PTFD*n*.

6.5.6.3 Port F Pull Enable Register (PTFPE)

	7	6	5	4	3	2	1	0
R W	PTFPE7	PTFPE6	PTFPE5	PTFPE4	PTFPE3	PTFPE2	PTFPE1	PTFPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-29. Internal Pull Enable for Port F Register (PTFPE)

Table 6-28. PTFPE Register Field Descriptions

Field	Description
7–0 PTFPE*n*	**Internal Pull Enable for Port F Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTF pin. For port F pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port F bit *n*. 1 Internal pullup device enabled for port F bit *n*.

6.5.6.4 Port F Slew Rate Enable Register (PTFSE)

	7	6	5	4	3	2	1	0
R W	PTFSE7	PTFSE6	PTFSE5	PTFSE4	PTFSE3	PTFSE2	PTFSE1	PTFSE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-30. Slew Rate Enable for Port F Register (PTFSE)

Table 6-29. PTFSE Register Field Descriptions

Field	Description
7–0 PTFSE*n*	**Output Slew Rate Enable for Port F Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTF pin. For port F pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port F bit *n*. 1 Output slew rate control enabled for port F bit *n*.

6.5.6.5 Port F Drive Strength Selection Register (PTFDS)

	7	6	5	4	3	2	1	0
R W	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-31. Drive Strength Selection for Port F Register (PTFDS)

Table 6-30. PTFDS Register Field Descriptions

Field	Description
7–0 PTFDS*n*	**Output Drive Strength Selection for Port F Bits** — Each of these control bits selects between low and high output drive for the associated PTF pin. For port F pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port F bit *n*. 1 High output drive strength selected for port F bit *n*.

6.5.7 Port G Registers

Port G is a 7-bit input/output port and also share its pins with the oscillator, keyboard interrupt, and ADC peripherals. On reset, the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTG0 shares its pin with KBIP0.
- PTG1 shares its pin with KBIP1.
- PTG2 shares its pin with KBIP2.
- PTG3 shares its pin with KBIP3 and AD1P18.
- PTG4 shares its pin with KBIP4 and AD1P19.

- PTG5 shares its pin with XTAL.
- PTG6 shares its pin with EXTAL.

Port G is controlled by the registers listed below.

6.5.7.1 Port G Data Register (PTGD)

	7	6	5	4	3	2	1	0
R	0	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-32. Port G Data Register (PTGD)

Table 6-31. PTGD Register Field Descriptions

Field	Description
6–0 PTGD*n*	**Port G Data Register Bits** — For port G pins that are inputs, reads return the logic level on the pin. For port G pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port G pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTGD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.7.2 Port G Data Direction Register (PTGDD)

	7	6	5	4	3	2	1	0
R	0	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-33. Port G Data Direction Register (PTGDD)

Table 6-32. PTGDD Register Field Descriptions

Field	Description
6–0 PTGDD*n*	**Data Direction for Port G Bits** — These read/write bits control the direction of port G pins and what is read for PTGD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port G bit *n* and PTGD reads return the contents of PTGD*n*.

6.5.7.3 Port G Pull Enable Register (PTGPE)

	7	6	5	4	3	2	1	0
R	0	PTGPE6	PTGPE5	PTGPE4	PTGPE3	PTGPE2	PTGPE1	PTGPE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-34. Internal Pull Enable for Port G Register (PTGPE)

Table 6-33. PTGPE Register Field Descriptions

Field	Description
6–0 PTGPE*n*	**Internal Pull Enable for Port G Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTG pin. For port G pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port G bit *n*. 1 Internal pullup device enabled for port G bit *n*.

6.5.7.4 Port G Slew Rate Enable Register (PTGSE)

	7	6	5	4	3	2	1	0
R	0	PTGSE6	PTGSE5	PTGSE4	PTGSE3	PTGSE2	PTGSE1	PTGSE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-35. Slew Rate Enable for Port G Register (PTGSE)

Table 6-34. PTGSE Register Field Descriptions

Field	Description
6–0 PTGSE*n*	**Output Slew Rate Enable for Port G Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTG pin. For port G pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port G bit *n*. 1 Output slew rate control enabled for port G bit *n*.

6.5.7.5 Port G Drive Strength Selection Register (PTGDS)

	7	6	5	4	3	2	1	0
R	0	PTGDS6	PTGDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-36. Drive Strength Selection for Port G Register (PTGDS)

Table 6-35. PTGDS Register Field Descriptions

Field	Description
6–0 PTGDS*n*	**Output Drive Strength Selection for Port G Bits** — Each of these control bits selects between low and high output drive for the associated PTG pin. For port G pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port G bit *n*. 1 High output drive strength selected for port G bit *n*.

6.5.8 Port H Registers

Port G is a 7-bit input/output port and also shares its pins with the FTM2, SPI2, VBUS, and ADC peripherals. On reset the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTH0 shares its pin with FTM2CH2 and AD1P20.
- PTH1 shares its pin with FTM2CH3, PSTCLK0, and AD1P21.
- PTH2 shares its pin with FTM2CH4, PSTCLK1, and AD1P22.
- PTH3 shares its pin with FTM2CH5, $\overline{\text{BKPT}}$, and AD1P23.
- PTH4 shares its pin with SPSCK2.
- PTH5 shares its pin with MOSI2.
- PTH6 shares its pin with MISO2.

Port H is controlled by the registers listed below.

6.5.8.1 Port H Data Register (PTHD)

	7	6	5	4	3	2	1	0
R	0	PTHD6	PTHD5	PTHD4	PTHD3	PTHD2	PTHD1	PTHD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-37. Port H Data Register (PTHD)

Table 6-36. PTHD Register Field Descriptions

Field	Description
6–0 PTHD*n*	**Port H Data Register Bits** — For port H pins that are inputs, reads return the logic level on the pin. For port H pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port H pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTHD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.8.2 Port H Data Direction Register (PTHDD)

	7	6	5	4	3	2	1	0
R	0	PTHDD6	PTHDD5	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-38. Port H Data Direction Register (PTHDD)

Table 6-37. PTHDD Register Field Descriptions

Field	Description
6–0 PTHDD*n*	**Data Direction for Port H Bits** — These read/write bits control the direction of port H pins and what is read for PTHD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port H bit *n* and PTHD reads return the contents of PTHD*n.*

6.5.8.3 Port H Pull Enable Register (PTHPE)

	7	6	5	4	3	2	1	0
R	0	PTHPE6	PTHPE5	PTHPE4	PTHPE3	PTHPE2	PTHPE1	PTHPE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-39. Internal Pull Enable for Port H Register (PTHPE)

Table 6-38. PTHPE Register Field Descriptions

Field	Description
6–0 PTHPE*n*	**Internal Pull Enable for Port H Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTH pin. For port H pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port H bit *n.* 1 Internal pullup device enabled for port H bit *n.*

6.5.8.4 Port H Slew Rate Enable Register (PTHSE)

	7	6	5	4	3	2	1	0
R	0	PTHSE6	PTHSE5	PTHSE4	PTHSE3	PTHSE2	PTHSE1	PTHSE0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-40. Slew Rate Enable for Port H Register (PTHSE)

Table 6-39. PTHSE Register Field Descriptions

Field	Description
6–0 PTHSE*n*	**Output Slew Rate Enable for Port H Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTH pin. For port H pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port H bit *n*. 1 Output slew rate control enabled for port H bit *n*.

6.5.8.5 Port H Drive Strength Selection Register (PTHDS)

	7	6	5	4	3	2	1	0
R	0	PTHDS6	PTHDS5	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 6-41. Drive Strength Selection for Port H Register (PTHDS)

Table 6-40. PTHDS Register Field Descriptions

Field	Description
6–0 PTHDS*n*	**Output Drive Strength Selection for Port H Bits** — Each of these control bits selects between low and high output drive for the associated PTH pin. For port H pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port H bit *n*. 1 High output drive strength selected for port H bit *n*.

6.5.9 Port J Registers

Port J is an 8-bit input/output port and shares its pins with the VBUS peripheral. On reset the pins will default to the input state and under the I/O control. Enabling any of the shared peripherals will take priority control over the pins. See Table 2-1 for details on pin function priorities.

- PTJ0 shares its pin with PST0.
- PTJ1 shares its pin with PST1.
- PTJ2 shares its pin with PST2.
- PTJ3 shares its pin with PST3.
- PTJ4 shares its pin with DDATA0.
- PTJ5 shares its pin with DDATA1.
- PTJ6 shares its pin with DDATA2.
- PTJ7 shares its pin with DDATA3.

Port J is controlled by the registers listed below.

6.5.9.1 Port J Data Register (PTJD)

	7	6	5	4	3	2	1	0
R W	PTJD7	PTJD6	PTJD5	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-42. Port J Data Register (PTJD)

Table 6-41. PTJD Register Field Descriptions

Field	Description
7–0 PTJD*n*	**Port J Data Register Bits** — For port J pins that are inputs, reads return the logic level on the pin. For port J pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port J pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTJD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pullups disabled.

6.5.9.2 Port J Data Direction Register (PTJDD)

	7	6	5	4	3	2	1	0
R W	PTJDD7	PTJDD6	PTJDD5	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0
Reset:	0	0	0	0	0	0	0	0

Figure 6-43. Port J Data Direction Register (PTJDD)

Table 6-42. PTJDD Register Field Descriptions

Field	Description
7–0 PTJDD*n*	**Data Direction for Port J Bits** — These read/write bits control the direction of port J pins and what is read for PTJD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port J bit *n* and PTJD reads return the contents of PTJD*n*.

6.5.9.3 Port J Pull Enable Register (PTJPE)

	7	6	5	4	3	2	1	0
R W	PTJPE7	PTJPE6	PTJPE5	PTJPE4	PTJPE3	PTJPE2	PTJPE1	PTJPE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-44. Internal Pull Enable for Port J Register (PTJPE)

Table 6-43. PTJPE Register Field Descriptions

Field	Description
7–0 PTJPE*n*	**Internal Pull Enable for Port J Bits** — Each of these control bits determines if the internal pullup device is enabled for the associated PTJ pin. For port J pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pullup device disabled for port J bit *n*. 1 Internal pullup device enabled for port J bit *n*.

6.5.9.4 Port J Slew Rate Enable Register (PTJSE)

	7	6	5	4	3	2	1	0
R W	PTJSE7	PTJSE6	PTJSE5	PTJSE4	PTJSE3	PTJSE2	PTJSE1	PTJSE0
Reset:	0	0	0	0	0	0	0	0

Figure 6-45. Slew Rate Enable for Port J Register (PTJSE)

Table 6-44. PTJSE Register Field Descriptions

Field	Description
7–0 PTJSE*n*	**Output Slew Rate Enable for Port J Bits** — Each of these control bits determines if the output slew rate control is enabled for the associated PTJ pin. For port J pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port J bit *n*. 1 Output slew rate control enabled for port J bit *n*.

6.5.9.5 Port J Drive Strength Selection Register (PTJDS)

	7	6	5	4	3	2	1	0
R W	PTJDS7	PTJDS6	PTJDS5	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0
Reset:	0	0	0	0	0	0	0	0

Figure 6-46. Drive Strength Selection for Port J Register (PTJDS)

Table 6-45. PTJDS Register Field Descriptions

Field	Description
7–0 PTJDS*n*	**Output Drive Strength Selection for Port J Bits** — Each of these control bits selects between low and high output drive for the associated PTJ pin. For port J pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port J bit *n*. 1 High output drive strength selected for port J bit *n*.

Chapter 7
ColdFire Core

7.1 Introduction

This section describes the organization of the Version 1 (V1) ColdFire® 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.

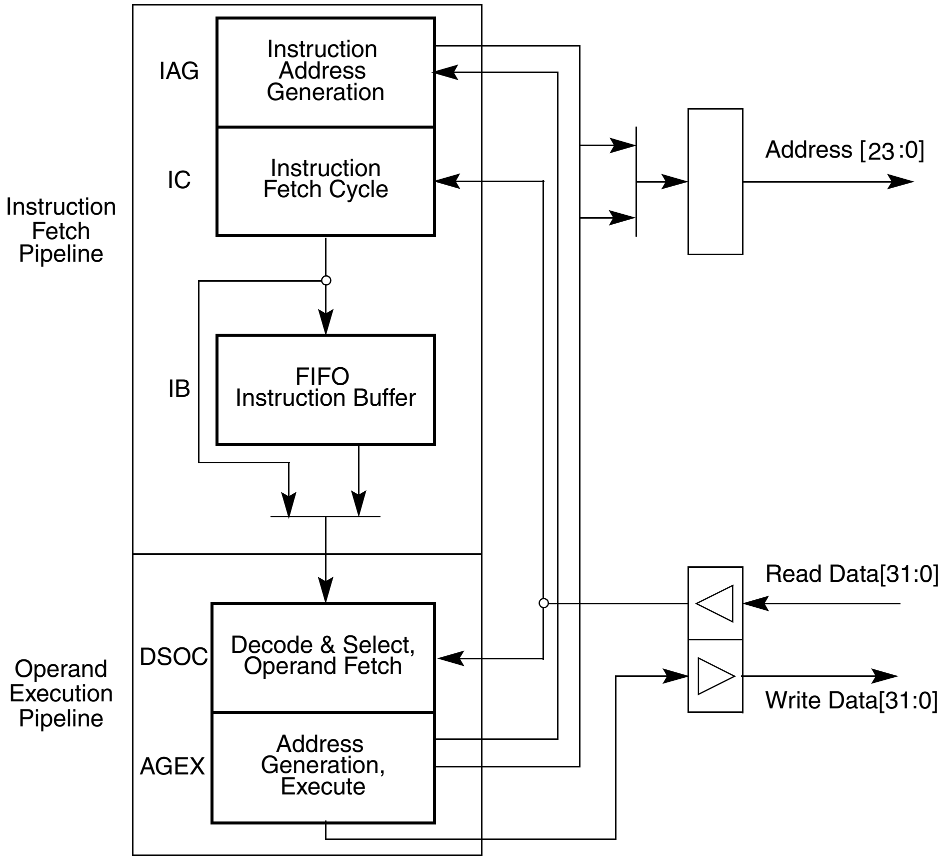

Instruction Fetch Pipeline
IAG
Instruction Address Generation
IC
Instruction Fetch Cycle
IB
FIFO Instruction Buffer
Address [23:0]
Read Data[31:0]
Write Data[31:0]
Operand Execution Pipeline
DSOC
Decode & Select, Operand Fetch
AGEX
Address Generation, Execute

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[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC[2]	Section/Page
Supervisor/User Access Registers						
Load: 0x60 Store: 0x40	Data Register 0 (D0)	32	R/W	See 7.3.3.14/7-20	No	7.2.1/7-3
Load: 0x61 Store: 0x41	Data Register 1 (D1)	32	R/W	See 7.3.3.14/7-20	No	7.2.1/7-3
Load: 0x6–7 Store: 0x4–7	Data Register –7 (D–D7)	32	R/W	POR: Undefined Else: Unaffected	No	7.2.1/7-3
Load: 0x68–E Store: 0x48–E	Address Register 0–6 (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No	7.2.2/7-4
Load: 0x6F Store: 0x4F	User A7 Stack Pointer (A7)	32	R/W	POR: Undefined Else: Unaffected	No	7.2.3/7-4
Load: 0xEE Store: 0xCE	Condition Code Register (CCR)	8	R/W	POR: Undefined Else: Unaffected	No	7.2.4/7-5
Load: 0xEF Store: 0xCF	Program Counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No	7.2.5/7-6
Supervisor Access Only Registers						
Load: 0xE0 Store: 0xC0	Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x(00)00_0000	No	7.2.3/7-4
Load: 0xE1 Store: 0xC1	Vector Base Register (VBR)	32	R/W	See section	Yes; Rc = 0x801	7.2.6/7-6
Load: 0xE2 Store: 0xC2	CPU Configuration Register (CPUCR)	32	W	See section	Yes; Rc = 0x802	7.2.7/7-7
Load: 0xEE Store: 0xCE	Status Register (SR)	16	R/W	0x27--	No	7.2.8/7-8

[1] 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 22, "Version 1 ColdFire Debug (CF1_DEBUG)." (These BDM commands are not similar to other ColdFire processors.)

[2] 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.

BDM: Load: 0x60 + *n; n* = 0-7 (D*n*)
Store: 0x40 + *n; n* = 0-7 (D*n*)

Access: User read/write
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data																															
Reset (D2-D7)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Reset (D0, D1)	See Section 7.3.3.14, “Reset Exception”																															

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.

BDM: Load: 0x68 + *n; n* = 0–6 (A*n*)
Store: 0x48 + *n; n* = 0–6 (A*n*)

Access: User read/write
BDM read/write

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

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.

BDM: Load: 0x6F (A7)
Store: 0x4F (A7)
Load: 0xE0 (OTHER_A7)
Store: 0xC0 (OTHER_A7)

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

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

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.

BDM: LSB of Status Register (SR)
Load: 0xEE (SR)
Store: 0xCE (SR)

Access: User read/write
BDM read/write

	7	6	5	4	3	2	1	0
R	0	0	0	X	N	Z	V	C
W								
Reset:	0	0	0	—	—	—	—	—

Figure 7-5. Condition Code Register (CCR)

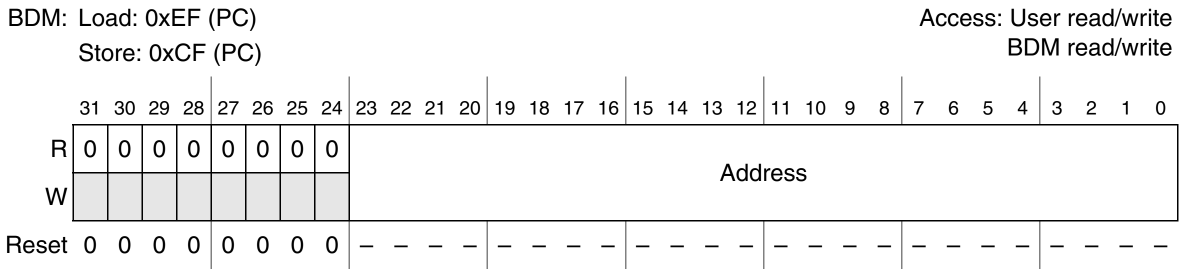
Table 7-2. CCR Field Descriptions

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

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.



BDM: Load: 0xEF (PC)
Store: 0xCF (PC)

Access: User read/write
BDM read/write

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

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.

BDM: 0x801 (VBR)
Load: 0xE1 (VBR)
Store: 0xC1 (VBR)

Access: Supervisor read/write
BDM read/write

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

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.

BDM: 0x802 (CPUCR)
Load: 0xE2 (CPUCR)
Store: 0xC2 (CPUCR)

Access: Supervisor read/write
BDM read/write

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

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. **Note:** 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.
24–0	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

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

Figure 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.
13 S	Supervisor/user state. 0 User mode 1 Supervisor mode

Table 7-4. SR Field Descriptions (continued)

Field	Description
12 M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	Reserved, must be cleared.
10–8 I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0 CCR	Refer to Section 7.2.4, “Condition Code Register (CCR)”.

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].

Table 7-5. Instruction Enhancements over Revision ISA_A (continued)

Instruction	Description
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 × 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 13, “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
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

Table 7-6. Exception Vector Assignments (continued)

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
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

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

All ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to disable interrupts effectively, if necessary, by raising the interrupt mask level contained in the status register. In addition, the ISA_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.

	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
SSP →	Format				FS[3:2]		Vector								FS[1:0]		Status Register															
+ 0x4	Program Counter																															

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
00*xx*	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.

7.3.2.2 S08 and ColdFire Exception Processing Comparison

This section presents a brief summary comparing the exception processing differences between the S08 and V1 ColdFire processor families.

Table 7-9. Exception Processing Comparison

Attribute	S08	V1 ColdFire
Exception Vector Table	32, 2-byte entries, fixed location at upper end of memory	103, 4-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 + 39 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
Software IACK	No	Yes
Exit Instruction from ISR	RTI	RTE

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 performance noticeably.

Emulation of the S08's 1-level IRQ processing can easily be managed by software convention within the ColdFire interrupt service routines. For this type of operation, only two of the seven interrupt levels are used:

- SR[I] equals 0 indicates interrupts are enabled
- SR[I] equals 7 indicates interrupts are disabled

Recall that ColdFire treats true level 7 interrupts as edge-sensitive, non-maskable requests. Typically, only the IRQ input pin and a low-voltage detect are assigned as level 7 requests. All the remaining interrupt requests (levels 1-6) are masked when SR[I] equals 7. In any case, all ColdFire processors guarantee that the first instruction of any exception handler is executed before interrupt sampling resumes. By making the first instruction of the ISR a store/load status register (`STLDSR #0x2700`) or a move-to-SR (`MOVE.W #2700,SR`) instruction, interrupts can be safely disabled until the service routine is exited with an RTE instruction that lowers the SR[I] back to level 0. The same functionality can also be provided without an explicit instruction by setting CPUCR[IME] because this forces the processor to load SR[I] with 7 on each interrupt exception.

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 CPUCR[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 CPUCR[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 7-10.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Line				OpMode						Effective Address					
										Mode			Register		

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

Table 7-10. 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 opwords in these two lines.

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any 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 22, “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 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 CPUCR[IAE]. See Chapter 13, "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 CPUCR[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 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 7-11. 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. (This is the value used for this device.) 1 MAC execute engine is present in core.
14 DIV	Divide present. This bit signals if the hardware divider (DIV) is present in the processor core. 0 Divide execute engine not present in core. (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					0	0	0
W																
Reset	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0

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

Figure 7-13. D1 Hardware Configuration Info

Table 7-12. 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 10011 96 KB flash 10100 256 KB flash (This is the value used for this device) 10110 512 KB flash Else Reserved for future use
18–16	Reserved
15–12	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 7-12. 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 (This is the value used for this device) 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 7-13.

Table 7-13. 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 7-14 lists execution times for MOVE.{B,W} instructions; Table 7-15 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 7-14. 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 7-14. 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 7-15. 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 7-16. 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 7-16. 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 7-17. 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 7-17. 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)	—	—	—	—	—	—	—
MULS.W	<ea>,Dx	9(0/0)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULU.W	<ea>,Dx	9(0/0)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULS.L	<ea>,Dx	≤18(0/0)	≤20(1/0)	≤20(1/0)	≤20(1/0)	≤20(1/0)	—	—	—
MULU.L	<ea>,Dx	≤18(0/0)	≤20(1/0)	≤20(1/0)	≤20(1/0)	≤20(1/0)	—	—	—

7.3.4.5 Miscellaneous Instruction Execution Times

Table 7-18. 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)	—	—	—	—	—	—	—

Table 7-18. Miscellaneous Instruction Execution Times (continued)

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

[1] The n is the number of registers moved by the MOVEM opcode.

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

[3] The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

[4] PEA execution times are the same for (d16,PC).

[5] PEA execution times are the same for (d8,PC,Xn*SF).

7.3.4.6 Branch Instruction Execution Times

Table 7-19. General Branch Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	2(0/1)	—	—	—
BSR		—	—	—	—	3(0/1)	—	—	—
JMP	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
JSR	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
RTE		—	—	7(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

Table 7-20. 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
Analog Comparator (ACMPV3)

8.1 Introduction

This device includes two independent analog comparators (ACMPs), named ACMP1 and ACMP2.

The analog comparator module (ACMP) provides a circuit for comparing two analog input voltages or for comparing one analog input voltage to an internal reference voltage. The comparator circuit can operate across the full range of the supply voltage (rail-to-rail operation).

NOTE

> Ignore any references to stop1 low-power mode in this chapter, because this device does not support it.

8.1.1 ACMP Configuration Information

When using the bandgap reference voltage for input to ACMP1+ and/or ACMP2+, the user must enable the bandgap buffer by setting SPMSC1[BGBE] (see Section 5.9.7, “System Power Management Status and Control 1 Register (SPMSC1)”). See data sheet for the value of the bandgap voltage.

8.1.2 ACMP/FTM/TPM Configuration Information

The ACMP modules can be configured to connect the output of the analog comparator to a TPM input capture channel 0 by setting the corresponding ACIC1 bit in the SOPT2 register (see Section 5.9.9, “System Options 2 (SOPT2) Register. With ACIC1 set, the TPMxCH0 pin is not available externally regardless of the configuration of the TPMx module.

The ACMP1 and ACMP2 outputs can be connected to FTM1CH0 and TPM3CH0, respectively.

In addition, the ACMP1 and ACMP2 outputs can be connected to FTM1 and FTM2, respectively, to be used as a synchronization signal. See Section 11.3.11, “FTM Synchronization Register (FTMxSYNC)”, for details.

8.1.3 ACMP Clock Gating

The bus clock to both ACMP modules can be gated on and off using the SCGC2[ACMPx] bit (see Section 5.9.11, “System Clock Gating Control 2 Register (SCGC2)”). This bit is set after any reset, which enables the bus clock to this module. To conserve power, the SCGC2[ACMPx] bit can be cleared to disable the clock to this module when not in use. See Section 5.8, “Peripheral Clock Gating,” for details.

8.1.4 Interrupt Vectors

ACMP1 and ACMP2 share a single interrupt vector, located at address 0xFFD6:0xFFD7. When interrupts are enabled for both ACMPs, the ACMP1SC[ACF] and ACMP2SC[ACF] bits must be polled to determine which ACMP caused the interrupt.

8.1.5 ACMPx Module-to-Module Interconnects

8.1.5.1 ACMPx to FTMx Synchronization Trigger

The ACMPxSC[ACF] bit of the ACMPx modules can be used as one of the three FTM hardware synchronization triggers. The ACMPx input is the second highest priority of the three hardware sync triggers. ACMP1SC[ACF] is connected to FTM1 and ACMP2SC[ACF] is connected to FTM2.

The ACF bit can be configured to set on rising voltage inputs, falling voltage inputs or both rising and falling inputs.

8.1.5.2 ACMP1 to FTM1 input capture

The ACMP1 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.

8.1.5.3 ACMP2 to TPM3 input capture

The ACMP2 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.

8.1.5.4 ACMP1 to ADC Hardware Trigger

The ACMP1 module has the capability to send a hardware trigger to the ADC using the ACMP1SC[ACF] bit. The ADC hardware trigger is enabled by the ADCSC2[ADTRG] bit. In addition, the hardware trigger source must be selected with the SOPT2[ADHWTS1:ADHWTS0] bits. ADHWTS1:ADHWTS0 = 0b01 selects ACMP1 as the ADC hardware trigger.

Typical usage is for the ACMP to be constantly monitoring a voltage level and the ADC to be off or monitoring a different signal until a voltage of interest is observed on the ACMP1 input. The ADC can then measure the same signal for a more accurate reading. Since ACMP1– and AD1P9 share the same pin as do ACMP1+ and AD1P8, taking an ADC reading on the ACMP1 inputs is trivial and doesn’t require separate pins for ACMP1 and ADC.

8.1.6 Features

The ACMP has the following features:

- Full rail to rail supply operation.
- Selectable interrupt on rising edge, falling edge, or either rising or falling edges of comparator output.
- Option to compare to fixed internal bandgap reference voltage.
- Option to allow comparator output to be visible on a pin, ACMPxO.

8.1.7 Modes of Operation

This section defines the ACMP operation in wait, stop and background debug modes.

8.1.7.1 ACMP in Wait Mode

The ACMP continues to run in wait mode if enabled before executing the WAIT instruction. Therefore, the ACMP can be used to bring the MCU out of wait mode if the ACMP interrupt, ACIE is enabled. For lowest possible current consumption, the ACMP should be disabled by software if not required as an interrupt source during wait mode.

8.1.7.2 ACMP in Stop Modes

8.1.7.2.1 Stop3 Mode Operation

The ACMP continues to operate in Stop3 mode if enabled and compare operation remains active. If ACOPE is enabled, comparator output operates as in the normal operating mode and comparator output is placed onto the external pin. The MCU is brought out of stop when a compare event occurs and ACIE is enabled; ACF flag sets accordingly.

If stop is exited with a reset, the ACMP will be put into its reset state.

8.1.7.2.2 Stop2 and Stop1 Mode Operation

During either Stop2 and Stop1 mode, the ACMP module will be fully powered down. Upon wake-up from Stop2 or Stop1 mode, the ACMP module will be in the reset state.

8.1.7.3 ACMP in Active Background Mode

When the microcontroller is in active background mode, the ACMP will continue to operate normally.

8.1.8 Block Diagram

The block diagram for the Analog Comparator module is shown Figure 8-1.

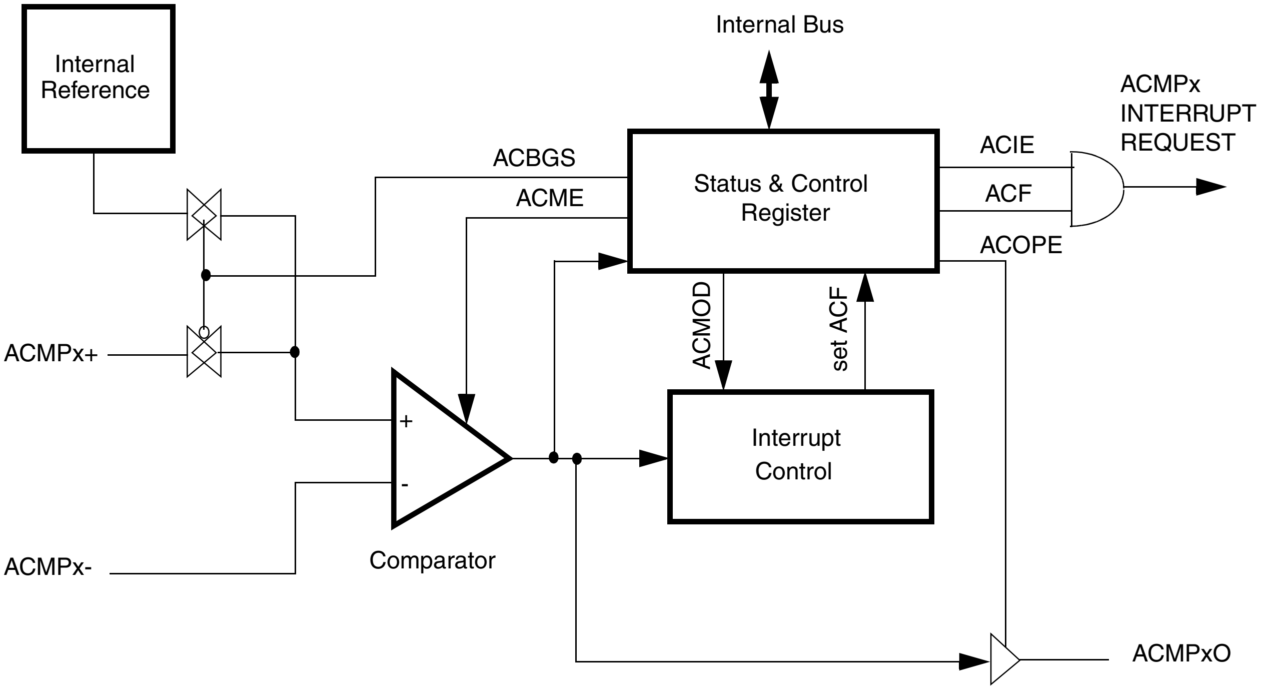


Figure 8-1. Analog Comparator 5 V Block Diagram

8.2 External Signal Description

The ACMP has two analog input pins, ACMPx+ and ACMPx- and one digital output pin ACMPxO. Each of these pins can accept an input voltage that varies across the full operating voltage range of the MCU. As shown in Figure 8-1, the ACMPx- pin is connected to the inverting input of the comparator, and the ACMPx+ pin is connected to the comparator non-inverting input if ACBGS is a 0. As shown in Figure 8-1, the ACMPxO pin can be enabled to drive an external pin.

The signal properties of ACMP are shown in Table 8-1.

Table 8-1. Signal Properties

Signal	Function	I/O
ACMPx-	Inverting analog input to the ACMP. (Minus input)	I
ACMPx+	Non-inverting analog input to the ACMP. (Positive input)	I
ACMPxO	Digital output of the ACMP.	O

8.3 Memory Map and Register Definition

8.3.1 Memory Map (Register Summary)

Table 8-2. ACMP Register Summary

Name		7	6	5	4	3	2	1	0
ACMPxSC	R	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD	
	W					▨			

8.3.2 Register Descriptions

The ACMP includes one register:

- An 8-bit status and control register

Refer to the direct-page register summary in the memory section of this data sheet for the absolute address assignments for all ACMP registers.This section refers to registers and control bits only by their names .

Some MCUs may have more than one ACMP, so register names include placeholder characters to identify which ACMP is being referenced.

8.3.2.1 ACMPx Status and Control Register (ACMPxSC)

ACMPxSC contains the status flag and control bits which are used to enable and configure the ACMP.

	7	6	5	4	3	2	1	0
R	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD	
W					▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 8-2. ACMPx Status and Control Register

.

Table 8-3. ACMPx Status and Control Register Field Descriptions

Field	Description
7 ACME	**Analog Comparator Module Enable** — ACME enables the ACMP module. 0 ACMP not enabled 1 ACMP is enabled
6 ACBGS	**Analog Comparator Bandgap Select** — ACBGS is used to select between the bandgap reference voltage or the ACMPx+ pin as the input to the non-inverting input of the analog comparatorr. 0 External pin ACMPx+ selected as non-inverting input to comparator 1 Internal reference select as non-inverting input to comparator Note: refer to this chapter introduction to verify if any other config bits are necessary to enable the bandgap reference in the chip level.
5 ACF	**Analog Comparator Flag** — ACF is set when a compare event occurs. Compare events are defined by ACMOD. ACF is cleared by writing a one to ACF. 0 Compare event has not occured 1 Compare event has occured
4 ACIE	**Analog Comparator Interrupt Enable** — ACIE enables the interrupt from the ACMP. When ACIE is set, an interupt will be asserted when ACF is set. 0 Interrupt disabled 1 Interrupt enabled
3 ACO	**Analog Comparator Output** — Reading ACO will return the current value of the analog comparator output. ACO is reset to a 0 and will read as a 0 when the ACMP is disabled (ACME = 0).
2 ACOPE	**Analog Comparator Output Pin Enable** — ACOPE is used to enable the comparator output to be placed onto the external pin, ACMPxO. 0 Analog comparator output not available on ACMPxO 1 Analog comparator output is driven out on ACMPxO
1:0 ACMOD	**Analog Comparator Mode** — ACMOD selects the type of compare event which sets ACF. 00 Encoding 0 — Comparator output falling edge 01 Encoding 1 — Comparator output rising edge 10 Encoding 2 — Comparator output falling edge 11 Encoding 3 — Comparator output rising or falling edge

8.4 Functional Description

The analog comparator can be used to compare two analog input voltages applied to ACMPx+ and ACMPx-; or it can be used to compare an analog input voltage applied to ACMPx- with an internal bandgap reference voltage. ACBGS is used to select between the bandgap reference voltage or the ACMPx+ pin as the input to the non-inverting input of the analog comparator. The comparator output 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. ACMOD is used to select the condition which will cause ACF to be set. ACF can be set on a rising edge of the comparator output, a falling edge of the comparator output, or either a rising or a falling edge (toggle). The comparator output can be read directly through ACO. The comparator output can be driven onto the ACMPxO pin using ACOPE.

Chapter 9
Analog-to-Digital Converter (ADC12V1)

9.1 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

> Ignore any references to stop1 low-power mode in this chapter, because this device does not support it.

9.1.1 ADC Clock Gating

The bus clock to the ADC can be gated on and off using the SCGC1[ADC] bit (see Section 5.9.8, "System Power Management Status and Control 2 Register (SPMSC2)"). This bit is set after any reset, which enables the bus clock to this module. To conserve power, the SCGC1[ADC] bit can be cleared to disable the clock to this module when not in use. See Section 5.8, "Peripheral Clock Gating," for details.

9.1.2 Module Configurations

This section provides information for configuring the ADC on this device.

9.1.2.1 Channel Assignments

The ADC channel assignments for this device are shown in Table 9-1. Reserved channels convert to an unknown value. Channels which are connected to an I/O pin have an associated pin control bit as shown.

Table 9-1. ADC Channel Assignment

ADCH	Channel	Input	Pin Control
00000	AD0	PTB0/AD1P0	ADPC0
00001	AD1	PTB1/AD1P1	ADPC1
00010	AD2	PTB2/AD1P2	ADPC2
00011	AD3	PTB3/AD1P3	ADPC3
00100	AD4	PTB4/AD1P4	ADPC4
00101	AD5	PTB5/AD1P5	ADPC5
00110	AD6	PTB6/AD1P6	ADPC6
00111	AD7	PTB7/AD1P7	ADPC7
01000	AD8	PTD0/AD1P8	ADPC8
10000	AD16	PTA6/AD1P16	ADPC16
10001	AD17	PTA7/AD1P17	ADPC17
10010	AD18	PTG3/AD1P18	ADPC18
10011	AD19	PTG4/AD1P19	ADPC19
10100	AD20	PTH0/AD1P20	ADPC20
10101	AD21	PTH1/AD1P21	ADPC21
10110	AD22	PTH2/AD1P22	ADPC22
10111	AD23	PTH3/AD1P23	ADPC23
11000	AD24	Reserved	N/A

Table 9-1. ADC Channel Assignment (continued)

ADCH	Channel	Input	Pin Control	ADCH	Channel	Input	Pin Control
01001	AD9	PTD1/AD1P9	ADPC9	11001	AD25	Reserved	N/A
01010	AD10	PTD2/AD1P10	ADPC10	11010	AD26	Temperature Sensor[1]	N/A
01011	AD11	PTD3/AD1P11	ADPC11	11011	AD27	Internal Bandgap	N/A
01100	AD12	PTD4/AD1P12	ADPC12	11100	—	Reserved	N/A
01101	AD13	PTD5/AD1P13	ADPC13	11101	V_{REFH}	V_{REFH}	N/A
01110	AD14	PTD6/AD1P14	ADPC14	11110	V_{REFL}	V_{REFL}	N/A
01111	AD15	PTD7/AD1P15	ADPC15	11111	Module Disabled	None	N/A

[1] For information, see Section 9.1.2.3, "Temperature Sensor."

NOTE

Selecting the internal bandgap channel requires SPMSC1[BGBE] to be set (see Section 5.9.7, "System Power Management Status and Control 1 Register (SPMSC1)"). See data sheet for the value of the bandgap voltage.

9.1.2.2 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 MCGERCLK. See Chapter 16, "Multipurpose Clock Generator (MCGV3)," for more information.

9.1.2.3 Temperature Sensor

The ADC module includes a temperature sensor whose output is connected to one of the ADC analog channel inputs. Equation 9-1 provides an approximate transfer function of the temperature sensor.

$$\text{Temp} = 25 - ((V_{TEMP} - V_{TEMP25}) \div m) \qquad \textit{Eqn. 9-1}$$

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 in the data sheet.

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 9-1. If V_{TEMP} is less than V_{TEMP25} the hot slope value is applied in Equation 9-1.

9.1.3 ADC Module Interconnects

9.1.3.1 ADC to FTMx synchronization triggers

The ADCSC1[COCO] bit of the ADC module can be used as one of the three FTM hardware synchronization triggers. This ADC input is the highest priority of the three hardware sync triggers.

9.1.3.2 ACMP1 to ADC Hardware Trigger

The ACMP1 module has the capability to send a hardware trigger to the ADC using the ACMP1SC[ACF] bit. The ADC hardware trigger is enabled by the ADCSC2[ADTRG] bit. In addition, the hardware trigger source must be selected with the SOPT2[ADHWTS1:ADHWTS0] bits. ADHWTS1:ADHWTS0 = 0b01 selects ACMP1 as the ADC hardware trigger.

Typical usage is for the ACMP to be constantly monitoring a voltage level and the ADC to be off or monitoring a different signal until a voltage of interest is observed on the ACMP1 input. The ADC can then measure the same signal for a more accurate reading. Since ACMP1– and AD1P9 share the same pin as do ACMP1+ and AD1P8, taking an ADC reading on the ACMP1 inputs is trivial and doesn’t require separate pins for ACMP1 and ADC.

9.1.3.3 FTM1 to ADC Hardware Trigger

The FTM1 module has the capability to send a hardware trigger to the ADC using the FTM1_ADC_TRIGGER register. FTM1 can be configured to send a trigger on any channel value match from channels 2–5 (channels 0–1 cannot be used). Multiple channels can be selected at once to generate multiple triggers to the ADC within one period of the FTM counter.

The ADC hardware trigger is enabled by the ADCSC2[ADTRG] bit. In addition, the hardware trigger source must be selected with the SOPT2[ADHWTS] bits. ADHWTS = 0b10 selects the FTM1 as the ADC hardware trigger.

9.1.3.4 RTI to ADC Hardware Trigger

The RTI module has the capability to send a hardware trigger to the ADC using the RTISC[RTIF] bit. The ADC hardware trigger is enabled by the ADCSC2[ADTRG] bit. In addition, the hardware trigger source must be selected with the SOPT2[ADHWTS] bits. ADHWTS = 0b00 selects the RTI as the ADC hardware trigger.

Typical usage is to use the RTI to send a trigger to the ADC to provide a constant periodic monitoring of the ADC input when configuring the ADC for continuos conversions is not desired. Since both the RTI and ADC modules can run in stop and wait modes and by using the automatic compare function of the ADC, periodic measurements can be made without waking the MCU from the low power mode until the ADC result meets the compare condition.

9.1.4 Features

Features of the ADC module include:

- Linear successive approximation algorithm with 12-bit resolution
- Up to 28 analog inputs
- Output formatted in 12-, 10-, or 8-bit right-justified unsigned format
- Single or continuous conversion (automatic return to idle after single conversion)
- Configurable sample time and conversion speed/power
- Conversion complete flag and interrupt
- Input clock selectable from up to four sources
- Operation in wait or stop3 modes for lower noise operation
- Asynchronous clock source for lower noise operation
- Selectable asynchronous hardware conversion trigger
- Automatic compare with interrupt for less-than, or greater-than or equal-to, programmable value
- Temperature sensor

9.1.5 ADC Module Block Diagram

Figure 9-1 provides a block diagram of the ADC module.

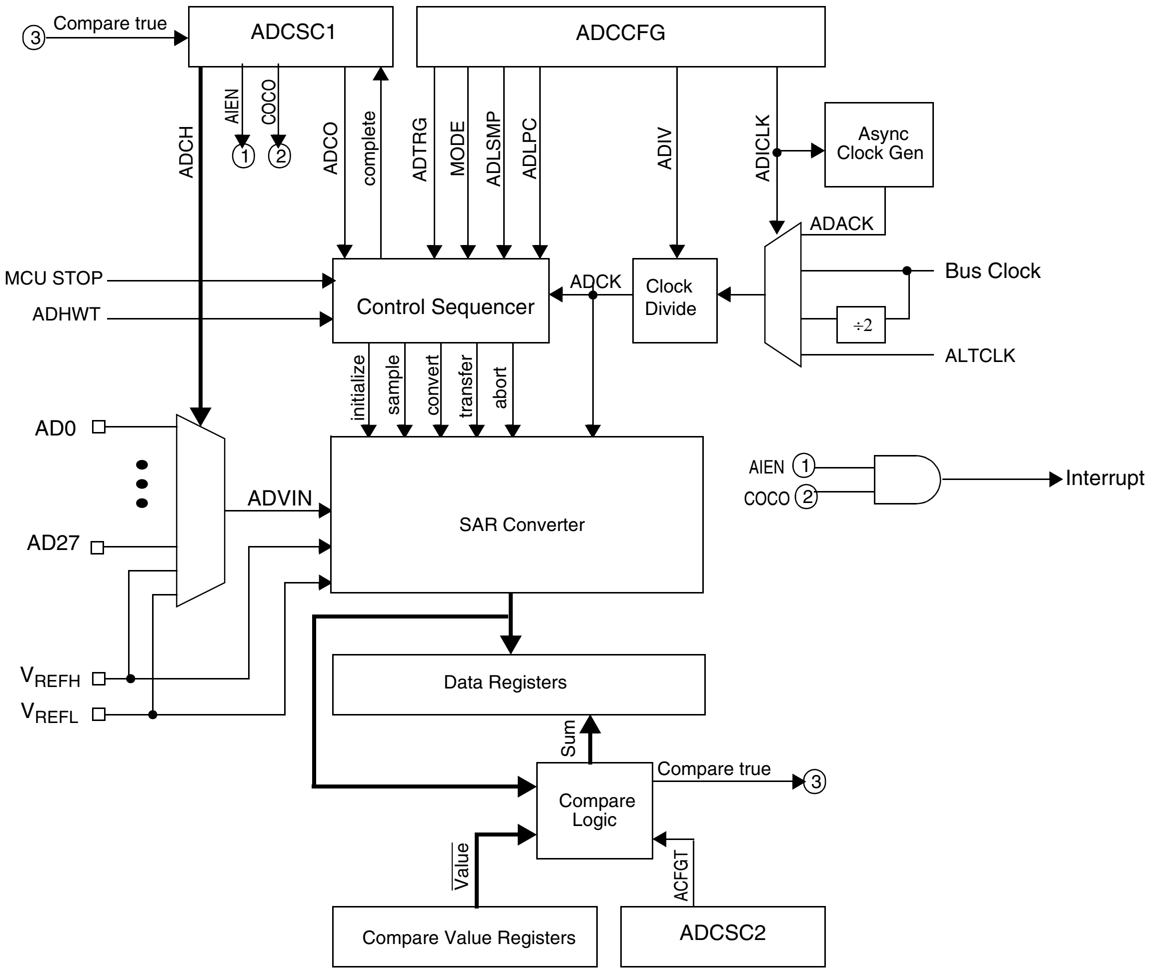


Figure 9-1. ADC Block Diagram

9.2 External Signal Description

The ADC module supports up to 28 separate analog inputs. It also requires four supply/reference/ground connections.

Table 9-2. Signal Properties

Name	Function
AD27–AD0	Analog Channel inputs
V_{REFH}	High reference voltage
V_{REFL}	Low reference voltage
V_{DDA}	Analog power supply
V_{SSA}	Analog ground

9.2.1 Analog Power (V_{DDA})

The ADC analog portion uses V_{DDA} as its power connection. In some packages, V_{DDA} is connected internally to V_{DD}. If externally available, connect the V_{DDA} pin to the same voltage potential as V_{DD}. External filtering may be necessary to ensure clean V_{DDA} for good results.

9.2.2 Analog Ground (V_{SSA})

The ADC analog portion uses V_{SSA} as its ground connection. In some packages, V_{SSA} is connected internally to V_{SS}. If externally available, connect the V_{SSA} pin to the same voltage potential as V_{SS}.

9.2.3 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_{DDA}. If externally available, V_{REFH} may be connected to the same potential as V_{DDA} or may be driven by an external source between the minimum V_{DDA} spec and the V_{DDA} potential (V_{REFH} must never exceed V_{DDA}).

9.2.4 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_{SSA}. If externally available, connect the V_{REFL} pin to the same voltage potential as V_{SSA}.

9.2.5 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.

9.3 Register Definition

These memory-mapped registers control and monitor operation of the ADC:

- Status and control register, ADCSC1
- Status and control register, ADCSC2
- Data result registers, ADCRH and ADCRL
- Compare value registers, ADCCVH and ADCCVL
- Configuration register, ADCCFG
- Pin control registers, APCTL1, APCTL2, APCTL3

9.3.1 Status and Control Register 1 (ADCSC1)

This section describes the function of the ADC status and control register (ADCSC1). Writing ADCSC1 aborts the current conversion and initiates a new conversion (if the ADCH bits are equal to a value other than all 1s).

	7	6	5	4	3	2	1	0
R	COCO	AIEN	ADCO	ADCH				
W								
Reset:	0	0	0	1	1	1	1	1

Figure 9-2. Status and Control Register (ADCSC1)

Table 9-3. ADCSC1 Field Descriptions

Field	Description
7 COCO	Conversion Complete Flag. The COCO flag is a read-only bit 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 when ADCSC1 is written or when ADCRL is read. 0 Conversion not completed 1 Conversion completed
6 AIEN	Interrupt Enable AIEN enables conversion complete interrupts. When COCO becomes set while AIEN is high, an interrupt is asserted. 0 Conversion complete interrupt disabled 1 Conversion complete interrupt enabled
5 ADCO	Continuous Conversion Enable. ADCO enables 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. 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	Input Channel Select. The ADCH bits form a 5-bit field that selects one of the input channels. The input channels are detailed in Table 9-4. The successive approximation converter subsystem is turned off when the channel select bits are all set. 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 ones 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.

Table 9-4. Input Channel Select

ADCH	Input Select
00000–01111	AD0–15
10000–11011	AD16–27
11100	Reserved
11101	V_{REFH}
11110	V_{REFL}
11111	Module disabled

9.3.2 Status and Control Register 2 (ADCSC2)

The ADCSC2 register controls the compare function, conversion trigger, and conversion active of the ADC module.

	7	6	5	4	3	2	1	0
R	ADACT	ADTRG	ACFE	ACFGT	0	0	R[1]	R[1]
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-3. Status and Control Register 2 (ADCSC2)

[1] Bits 1 and 0 are reserved bits that must always be written to 0.

Table 9-5. ADCSC2 Register Field Descriptions

Field	Description
7 ADACT	Conversion Active. 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
6 ADTRG	Conversion Trigger Select. Selects the type of trigger used for initiating a conversion. Two types of triggers 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 ACFE	Compare Function Enable. Enables the compare function. 0 Compare function disabled 1 Compare function enabled
4 ACFGT	Compare Function Greater Than Enable. Configures the compare function to trigger when the result of the conversion of the input being monitored is greater than or equal to the compare value. The compare function defaults to triggering when the result of the compare of the input being monitored is less than the compare value. 0 Compare triggers when input is less than compare value 1 Compare triggers when input is greater than or equal to compare value

9.3.3 Data Result High Register (ADCRH)

In 12-bit operation, ADCRH contains the upper four bits of 12-bit conversion data. In 10-bit operation, ADCRH contains the upper two bits of 10-bit conversion data. In 12-bit and 10-bit mode, ADCRH is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met. When configured for 10-bit mode, ADR[11:10] are cleared. When configured for 8-bit mode, ADR[11:8] are cleared.

When automatic compare is not enabled, the value stored in ADCRH are the upper bits of the conversion result. When automatic compare is enabled, the conversion result is manipulated as described in Section 9.4.5, “Automatic Compare Function” prior to storage in ADCRH:ADCRL registers.

In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion data into the result registers until ADCRL is read. If ADCRL is not read until after the next conversion is completed, the intermediate conversion data is lost. In 8-bit mode, there is no interlocking with ADCRL. If the MODE bits are changed, any data in ADCRH becomes invalid.

	7	6	5	4	3	2	1	0
R	0	0	0	0	ADR11	ADR10	ADR9	ADR8
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-4. Data Result High Register (ADCRH)

9.3.4 Data Result Low Register (ADCRL)

ADCRL contains the lower eight bits of a 12-bit or 10-bit conversion data, and all eight bits of 8-bit conversion data. ADCRL is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met.

When automatic compare is not enabled, the value stored in ADCRL is the lower eight bits of the conversion result. When automatic compare is enabled, the conversion result is manipulated as described in Section 9.4.5, “Automatic Compare Function” prior to storage in ADCRH:ADCRL registers.

In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion data into the result registers until ADCRL is read. If ADCRL is not read until the after next conversion is completed, the intermediate conversion data is lost. In 8-bit mode, there is no interlocking with ADCRH. If the MODE bits are changed, any data in ADCRL becomes invalid.

	7	6	5	4	3	2	1	0
R	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-5. Data Result Low Register (ADCRL)

9.3.5 Compare Value High Register (ADCCVH)

In 12-bit mode, the ADCCVH register holds the upper four bits of the 12-bit compare value. When the compare function is enabled, these bits are compared to the upper four bits of the result following a conversion in 12-bit mode.

	7	6	5	4	3	2	1	0
R	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-6. Compare Value High Register (ADCCVH)

In 10-bit mode, the ADCCVH register holds the upper two bits of the 10-bit compare value (ADCV[9:8]). These bits are compared to the upper two bits of the result following a conversion in 10-bit mode when the compare function is enabled.

In 8-bit mode, ADCCVH is not used during compare.

9.3.6 Compare Value Low Register (ADCCVL)

This register holds the lower eight bits of the 12-bit or 10-bit compare value or all eight bits of the 8-bit compare value. When the compare function is enabled, bits ADCV[7:0] are compared to the lower eight bits of the result following a conversion in 12-bit, 10-bit or 8-bit mode.

	7	6	5	4	3	2	1	0
R	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-7. Compare Value Low Register(ADCCVL)

9.3.7 Configuration Register (ADCCFG)

ADCCFG selects the mode of operation, clock source, clock divide, and configures for low power and long sample time.

	7	6	5	4	3	2	1	0
R	ADLPC	ADIV		ADLSMP	MODE		ADICLK	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-8. Configuration Register (ADCCFG)

Table 9-6. ADCCFG Register Field Descriptions

Field	Description
7 ADLPC	Low-Power Configuration. ADLPC controls the speed and power configuration of the successive approximation converter. This optimizes 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	Clock Divide Select. ADIV selects the divide ratio used by the ADC to generate the internal clock ADCK. Table 9-7 shows the available clock configurations.
4 ADLSMP	Long Sample Time Configuration. 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	Conversion Mode Selection. MODE bits are used to select between 12-, 10-, or 8-bit operation. See Table 9-8.
1:0 ADICLK	Input Clock Select. ADICLK bits select the input clock source to generate the internal clock ADCK. See Table 9-9.

Table 9-7. 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 9-8. 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 9-9. Input Clock Select

ADICLK	Selected Clock Source
00	Bus clock
01	Bus clock divided by 2
10	Alternate clock (ALTCLK)
11	Asynchronous clock (ADACK)

9.3.8 Pin Control 1 Register (APCTL1)

The pin control registers disable the I/O port control of MCU pins used as analog inputs. APCTL1 is used to control the pins associated with channels 0–7 of the ADC module.

	7	6	5	4	3	2	1	0
R W	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
Reset:	0	0	0	0	0	0	0	0

Figure 9-9. Pin Control 1 Register (APCTL1)

Table 9-10. APCTL1 Register Field Descriptions

Field	Description
7 ADPC7	ADC Pin Control 7. ADPC7 controls the pin associated with channel AD7. 0 AD7 pin I/O control enabled 1 AD7 pin I/O control disabled
6 ADPC6	ADC Pin Control 6. ADPC6 controls the pin associated with channel AD6. 0 AD6 pin I/O control enabled 1 AD6 pin I/O control disabled
5 ADPC5	ADC Pin Control 5. ADPC5 controls the pin associated with channel AD5. 0 AD5 pin I/O control enabled 1 AD5 pin I/O control disabled
4 ADPC4	ADC Pin Control 4. ADPC4 controls the pin associated with channel AD4. 0 AD4 pin I/O control enabled 1 AD4 pin I/O control disabled
3 ADPC3	ADC Pin Control 3. ADPC3 controls the pin associated with channel AD3. 0 AD3 pin I/O control enabled 1 AD3 pin I/O control disabled
2 ADPC2	ADC Pin Control 2. ADPC2 controls the pin associated with channel AD2. 0 AD2 pin I/O control enabled 1 AD2 pin I/O control disabled

Table 9-10. APCTL1 Register Field Descriptions (continued)

Field	Description
1 ADPC1	ADC Pin Control 1. ADPC1 controls the pin associated with channel AD1. 0 AD1 pin I/O control enabled 1 AD1 pin I/O control disabled
0 ADPC0	ADC Pin Control 0. ADPC0 controls the pin associated with channel AD0. 0 AD0 pin I/O control enabled 1 AD0 pin I/O control disabled

9.3.9 Pin Control 2 Register (APCTL2)

APCTL2 controls channels 8–15 of the ADC module.

	7	6	5	4	3	2	1	0
R W	ADPC15	ADPC14	ADPC13	ADPC12	ADPC11	ADPC10	ADPC9	ADPC8
Reset:	0	0	0	0	0	0	0	0

Figure 9-10. Pin Control 2 Register (APCTL2)

Table 9-11. APCTL2 Register Field Descriptions

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

Table 9-11. APCTL2 Register Field Descriptions (continued)

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

9.3.10 Pin Control 3 Register (APCTL3)

APCTL3 controls channels 16–23 of the ADC module.

	7	6	5	4	3	2	1	0
R W	ADPC23	ADPC22	ADPC21	ADPC20	ADPC19	ADPC18	ADPC17	ADPC16
Reset:	0	0	0	0	0	0	0	0

Figure 9-11. Pin Control 3 Register (APCTL3)

Table 9-12. APCTL3 Register Field Descriptions

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

Table 9-12. APCTL3 Register Field Descriptions (continued)

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

9.4 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 registers (ADCRH and ADCRL). In 10-bit mode, the result is rounded to 10 bits and placed in the data registers (ADCRH and ADCRL). In 8-bit mode, the result is rounded to 8 bits and placed in ADCRL. The conversion complete flag (COCO) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (AIEN = 1).

The ADC module has the capability of automatically comparing the result of a conversion with the contents of its compare registers. The compare function is enabled by setting the ACFE bit and operates with any of the conversion modes and configurations.

9.4.1 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 two. 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 do not perform according to specifications. If the available clocks are too fast, 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.

9.4.2 Input Select and Pin Control

The pin control registers (APCTL3, APCTL2, and APCTL1) disable the I/O port control of the pins used as analog inputs.When a pin control register bit is set, the following conditions are forced for the associated MCU pin:

- The output buffer is forced to its high impedance state.
- The input buffer is disabled. A read of the I/O port returns a zero for any pin with its input buffer disabled.
- The pullup is disabled.

9.4.3 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 convert configuration, only the initial rising edge to launch continuous conversions is observed. The hardware trigger function operates in conjunction with any of the conversion modes and configurations.

9.4.4 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 a software or hardware trigger. In addition, the ADC module can be configured for low power operation, long sample time, continuous conversion, and automatic compare of the conversion result to a software determined compare value.

9.4.4.1 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 conversion is enabled.

If continuous conversions are enabled, a new conversion is automatically initiated after the completion of the current conversion. In software triggered operation, continuous conversions begin after ADCSC1 is written and continue until aborted. In hardware triggered operation, continuous conversions begin after a hardware trigger event and continue until aborted.

9.4.4.2 Completing Conversions

A conversion is completed when the result of the conversion is transferred into the data result registers, ADCRH and ADCRL. This is indicated by the setting of COCO. An interrupt is generated if AIEN is high at the time that COCO is set.

A blocking mechanism prevents a new result from overwriting previous data in ADCRH and ADCRL if the previous data is in the process of being read while in 12-bit or 10-bit MODE (the ADCRH register has been read but the ADCRL register has not). When blocking is active, the data transfer is blocked, COCO is not set, and the new result is lost. In the case of single conversions with the compare function enabled and the compare condition false, blocking has no effect and ADC operation is terminated. In all other cases of operation, when a data transfer is blocked, another conversion is initiated regardless of the state of ADCO (single or continuous conversions enabled).

If single conversions are enabled, the blocking mechanism could result in several discarded conversions and excess power consumption. To avoid this issue, the data registers must not be read after initiating a single conversion until the conversion completes.

9.4.4.3 Aborting Conversions

Any conversion in progress is 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, ADCCFG, ADCCVH, or ADCCVL occurs. This indicates a mode of operation change has occurred and the current conversion is therefore invalid.
- The MCU is reset.
- The MCU enters stop mode with ADACK not enabled.

When a conversion is aborted, the contents of the data registers, ADCRH and ADCRL, are not altered. However, they continue to be the values transferred after the completion of the last successful conversion. If the conversion was aborted by a reset, ADCRH and ADCRL return to their reset states.

9.4.4.4 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} (see the electrical specifications).

9.4.4.5 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 selects 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 ADCRH and ADCRL 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 9-13.

Table 9-13. 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 μs + 20 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	0	5 μs + 23 ADCK + 5 bus clock cycles
Single or first continuous 8-bit	11	1	5 μs + 40 ADCK + 5 bus clock cycles
Single or first continuous 10-bit or 12-bit	11	1	5 μs + 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:

$$\text{Conversion time} = \frac{\text{23 ADCK Cyc}}{\text{8 MHz/1}} + \frac{\text{5 bus Cyc}}{\text{8 MHz}} = 3.5\ \mu\text{s}$$

$$\text{Number of bus cycles} = 3.5\ \mu\text{s x 8 MHz} = 28\text{ cycles}$$

NOTE

The ADCK frequency must be between f_{ADCK} minimum and f_{ADCK} maximum to meet ADC specifications.

9.4.5 Automatic Compare Function

The compare function is enabled by the ACFE bit. The compare function can be configured to check for an upper or lower limit. After the input is sampled and converted, the compare value (ADCCVH and ADCCVL) is subtracted from the conversion result. When comparing to an upper limit (ACFGT = 1), if the conversion result is greater-than or equal-to the compare value, COCO is set. When comparing to a lower limit (ACFGT = 0), if the result is less than the compare value, COCO is set. An ADC interrupt is generated upon the setting of COCO if the ADC interrupt is enabled (AIEN = 1).

The subtract operation of two positive values (the conversion result less the compare value) results in a signed value that is 1-bit wider than the bit-width of the two terms. The final value transferred to the ADCRH and ADCRL registers is the result of the subtraction operation, excluding the sign bit. The value of the sign bit can be derived based on ACFGT control setting. When ACFGT=1, the sign bit of any value stored in ADCRH and ADCRL is always 0, indicating a positive result for the subtract operation. When ACFGT = 1, the sign bit of any result is always 1, indicating a negative result for the subtract operation.

Upon completion of a conversion while the compare function is enabled, if the compare condition is not true, COCO is not set and no data is transferred to the result registers.

NOTE

> The compare function can monitor the voltage on a channel while the MCU is in wait or stop3 mode. The ADC interrupt wakes the MCU when the compare condition is met.

An example of compare operation eases understanding of the compare feature. If the ADC is configured for 10-bit operation, ACFGT=0, and ADCCVH:ADCCVL= 0x200, then a conversion result of 0x080 causes the compare condition to be met and the COCO bit is set. A value of 0x280 is stored in ADCRH:ADCRL. This is signed data without the sign bit and must be combined with a derived sign bit to have meaning. The value stored in ADCRH:ADCRL is calculated as follows.

The value to interpret from the data is (Result – Compare Value) = (0x080 – 0x200) = –0x180. A standard method for handling subtraction is to convert the second term to its 2's complement, and then add the two terms. First calculate the 2's complement of 0x200 by complementing each bit and adding 1. Note that prior to complementing, a sign bit of 0 is added so that the 10-bit compare value becomes a 11-bit signed value that is always positive.

```
  %101 1111 1111        <= 1’s complement of 0x200 compare value
+             %1
  ---------------
  %110 0000 0000        <= 2’s complement of 0x200 compare value
```

Then the conversion result of 0x080 is added to 2’s complement of 0x200:

```
  %000 1000 0000
+ %110 0000 0000
  ---------------
   %110 1000 0000       <= Subtraction result is –0x180 in signed 11-bit data
```

The subtraction result is an 11-bit signed value. The lower 10 bits (0x280) are stored in ADCRH:ADCRL. The sign bit is known to be 1 (negative) because the ACFGT=0, the COCO bit was set, and conversion data was updated in ADCRH:ADCRL.

A simpler way to use the data stored in ADCRH:ADCRL is to apply the following rules. When comparing for upper limit (ACFGT=1), the value in ADCRH:ADCRL is a positive value and does not need to be manipulated. This value is the difference between the conversion result and the compare value. When comparing for lower limit (ACFGT=0), ADCRH:ADCRL is a negative value without the sign bit. If the value from these registers is complemented and then a value of 1 is added, then the calculated value is the unsigned (i.e., absolute) difference between the conversion result and the compare value. In the previous example, 0x280 is stored in ADCRH:ADCRL. The following example shows how the absolute value of the difference is calculated.

```
%01 0111 1111 <= Complement of 10-bit value stored in ADCRH:ADCRL
+          %1
---------------
 %01 1000 0000<= Unsigned value 0x180 is the absolute value of (Result - Compare Value)
```

9.4.6 MCU Wait Mode Operation

Wait mode is a lower power-consumption standby mode from which recovery is 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 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).

9.4.7 MCU Stop3 Mode Operation

Stop mode is a low power-consumption standby mode during which most or all clock sources on the MCU are disabled.

9.4.7.1 Stop3 Mode With ADACK Disabled

If the asynchronous clock, ADACK, is not selected as the conversion clock, executing a stop instruction aborts the current conversion and places the ADC in its idle state. The contents of ADCRH and ADCRL are unaffected by stop3 mode. After exiting from stop3 mode, a software or hardware trigger is required to resume conversions.

9.4.7.2 Stop3 Mode With ADACK Enabled

If ADACK is selected as the conversion clock, the ADC continues operation during stop3 mode. For guaranteed ADC operation, the MCU's voltage regulator must remain active during stop3 mode. Consult the module introduction for configuration information for this MCU.

If a conversion is in progress when the MCU enters stop3 mode, it continues until completion. Conversions can be initiated while the MCU is in stop3 mode by means of the hardware trigger or if continuous conversions are enabled.

A conversion complete event sets the COCO and generates an ADC interrupt to wake the MCU from stop3 mode if the ADC interrupt is enabled (AIEN = 1).

NOTE

> The ADC module can wake the system from low-power stop and cause the MCU to begin consuming run-level currents without generating a system level interrupt. To prevent this scenario, software should ensure the data transfer blocking mechanism (discussed in Section 9.4.4.2, "Completing Conversions) is cleared when entering stop3 and continuing ADC conversions.

9.4.8 MCU Stop2 Mode Operation

The ADC module is automatically disabled when the MCU enters stop2 mode. All module registers contain their reset values following exit from stop2. Therefore, the module must be re-enabled and re-configured following exit from stop2.

9.5 Initialization Information

This section gives an example that provides some basic direction on how to initialize and configure the ADC module. You can configure 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 9-7, Table 9-8, and Table 9-9 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.

9.5.1 ADC Module Initialization Example

9.5.1.1 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) and compare function options, if enabled.
3. Update status and control register 1 (ADCSC1) to select whether conversions will be continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions will be performed is also selected here.

9.5.1.2 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.

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 ÷ 1
Bit 4    ADLSMP   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    ACFE     0        Compare function disabled
Bit 4    ACFGT    0        Not used in this example
Bit 3:2           00       Reserved, always reads zero
Bit 1:0           00       Reserved for Freescale’s internal use; always write zero
```

ADCSC1 = 0x41 (%01000001)

```
Bit 7    COCO     0        Read-only flag which 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
```

ADCRH/L = 0xxx

```
Holds results of conversion. Read high byte (ADCRH) before low byte (ADCRL) so that
conversion data cannot be overwritten with data from the next conversion.
```

ADCCVH/L = 0xxx

```
Holds compare value when compare function enabled
```

APCTL1=0x02

```
AD1 pin I/O control disabled. All other AD pins remain general purpose I/O pins
```

APCTL2=0x00

```
All other AD pins remain general purpose I/O pins
```

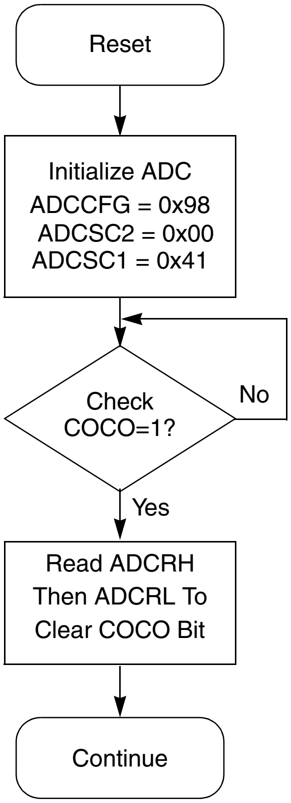


Figure 9-12. Initialization Flowchart for Example

9.6 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.

9.6.1 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.

9.6.1.1 Analog Supply Pins

The ADC module has analog power and ground supplies (V_{DDA} and V_{SSA}) available as separate pins on some devices. V_{SSA} is shared on the same pin as the MCU digital V_{SS} on some devices. On other devices, V_{SSA} and V_{DDA} are shared with the MCU digital supply pins. In these cases, there are separate pads for the analog supplies 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_{DDA} and V_{SSA} 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.

If separate power supplies are used for analog and digital power, the ground connection between these supplies must be at the V_{SSA} pin. This should be the only ground connection between these supplies if possible. The V_{SSA} pin makes a good single point ground location.

9.6.1.2 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_{DDA} on some devices. The low reference is V_{REFL}, which may be shared on the same pin as V_{SSA} on some devices.

When available on a separate pin, V_{REFH} may be connected to the same potential as V_{DDA}, or may be driven by an external source between the minimum V_{DDA} spec and the V_{DDA} potential (V_{REFH} must never exceed V_{DDA}). When available on a separate pin, V_{REFL} must be connected to the same voltage potential as V_{SSA}. 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 μF 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).

9.6.1.3 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 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 μF 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 should not be transitioning during conversions.

9.6.2 Sources of Error

Several sources of error exist for A/D conversions. These are discussed in the following sections.

9.6.2.1 Sampling Error

For proper conversions, the input must be sampled long enough to achieve the proper accuracy. Given the maximum input resistance of approximately 7kΩ 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 kΩ.

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.

9.6.2.2 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_{DDA} / (2^N*I_{LEAK}) for less than 1/4LSB leakage error (N = 8 in 8-bit, 10 in 10-bit or 12 in 12-bit mode).

9.6.2.3 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 μF low-ESR capacitor from V_{REFH} to V_{REFL}.
- There is a 0.1 μF low-ESR capacitor from V_{DDA} to V_{SSA}.
- If inductive isolation is used from the primary supply, an additional 1 μF capacitor is placed from V_{DDA} to V_{SSA}.
- V_{SSA} (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 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 μF capacitor (C_{AS}) on the selected input channel to V_{REFL} or V_{SSA} (this improves noise issues, but affects the 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.

9.6.2.4 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:

$$1 \text{ lsb} = (V_{REFH} - V_{REFL}) / 2^N \qquad \textit{Eqn. 9-2}$$

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 will be ± 1/2 lsb in 8- or 10-bit mode. As a consequence, however, the code width of the first (0x000) conversion is only 1/2 lsb and the code width of the last (0xFF or 0x3FF) is 1.5 lsb.

For 12-bit conversions the code transitions only after the full code width is present, so the quantization error is –1 lsb to 0 lsb and the code width of each step is 1 lsb.

9.6.2.5 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_{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 lsb in 8-bit or 10-bit modes and 1 lsb in 12-bit mode). If the first conversion is 0x001, the difference between the actual 0x001 code width and its ideal (1 lsb) is used.
- Full-scale error (E_{FS}) — This error is defined as the difference between the actual code width of the last conversion and the ideal code width (1.5 lsb in 8-bit or 10-bit modes and 1LSB in 12-bit mode). If the last conversion is 0x3FE, the difference between the actual 0x3FE code width and its ideal (1LSB) 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 includes all forms of error.

9.6.2.6 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 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 ±1/2 lsb in 8-bit or 10-bit mode, or around 2 lsb in 12-bit mode, and increases with noise.

This error may be reduced by repeatedly sampling the input and averaging the result. Additionally the techniques discussed in Section 9.6.2.3 reduces 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 never converted for any input value.

In 8-bit or 10-bit mode, the ADC is guaranteed to be monotonic and have no missing codes.

Chapter 10
Cyclic Redundancy Check Generator (CRCV2)

10.1 Introduction

This section describes the Cyclic Redundancy Check (CRC) generator module which uses the 16-bit CRC-CCITT polynomial, $x^{16} + x^{12} + x^5 + 1$, to generate a CRC code for error detection. The 16-bit code is calculated for 8-bits of data at a time, and provides a simple check for all accessible memory locations, whether they be in FLASH or RAM.

10.1.1 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

10.1.2 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.

10.1.3 Block Diagram

Figure 10-1 provides a block diagram of the CRC module.

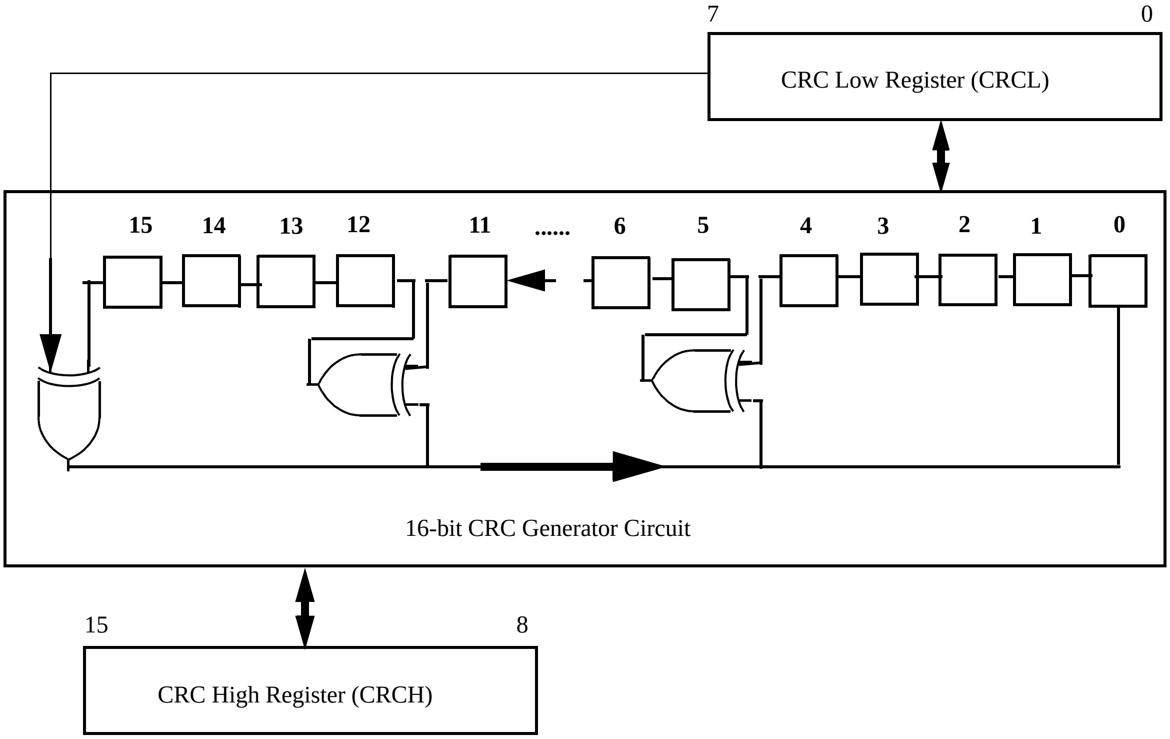


Figure 10-1. Cyclic Redundancy Check (CRC) Module Block Diagram

10.2 External Signal Description

There are no CRC signals that connect off chip.

10.3 Register Definition

10.3.1 Memory Map

Table 10-1. CRC Register Summary

Name		7	6	5	4	3	2	1	0
CRCH (offset=0)	R W	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
CRCL (offset=1)	R W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

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 must be ignored for HCS08 cores. See Section 10.4.2, “Programming Model Extension for CF1Core for more details.

10.3.2 Register Description

The CRC module includes:

- A 16-bit CRC result and seed register (CRCH:CRCL)

Refer to the direct-page register summary in the Memory chapter of this data sheet 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.

10.3.2.1 CRC High Register (CRCH)

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

Figure 10-2. CRC High Register (CRCH)

Table 10-2. Register Field Descriptions

Field	Description
7–0 CRCH	**CRCH** — 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.

10.3.2.2 CRC Low Register (CRCL)

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

Figure 10-3. CRC High Register (CRCH)

Table 10-3. Register Field Descriptions

Field	Description
7–0 CRCL	**CRCL —** 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.

10.4 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 must 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. After all 8 bits have been shifted into the CRC generator (in the next bus cycle after the data write to CRCL), 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.

10.4.1 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 10-1, but it recommends a SEED = 0x0000.
- ITU-T T.30 and ITU-T X.25 implement the same circuit shown in Figure 10-1, 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 10-4 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 10-4. Expected CRC results

ASCII String Message	SEED (initial CRC value)	CRC result
“A”	0x0000	0x58e5
“A”	0xffff	0xb915
“A”	0x1d0f[1]	0x9479
“123456789”	0x0000	0x31c3
“123456789”	0xffff	0x29b1
“123456789”	0x1d0f[1]	0xe5cc
A string of 256 upper case “A” characters with no line breaks	0x0000	0xabe3
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[1]	0xe938

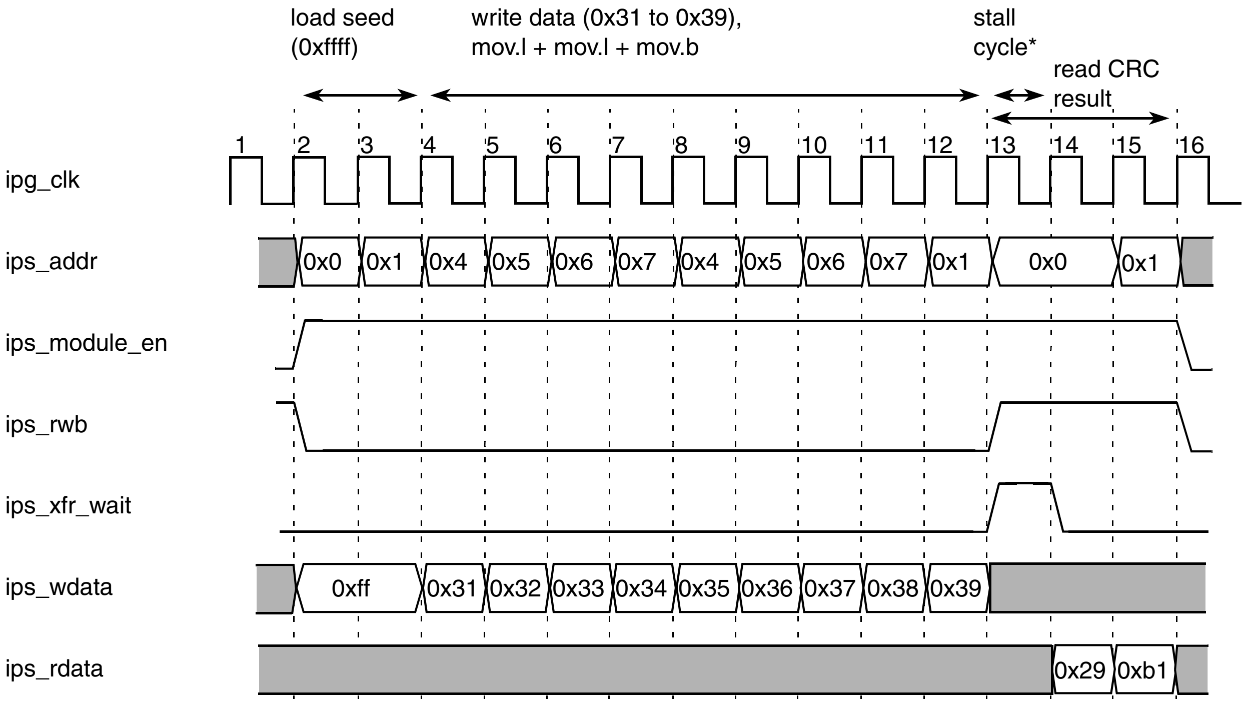
[1] One common variation of CRC-CCITT require the message to be augmented with zeros and a SEED=0xFFFF. The CRC module will give the same results of this alternative implementation when SEED=0x1D0F and no message augmentation.

10.4.2 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 must 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. Reads from 0x2-0x3 (unused/reserved) return 0x00. Writes to 0x2-0x3 have no effect.

Due to internal CF1Core characteristics, this approach provides a greater data transfer rate than the original programming model used on HCS08 (single byte writes to CRCL position). Figure 10-4 illustrates a message calculation on CF1Core.



* On cycle 13, there is a read-after-write hazard, since calculation of 0x39 data is underway. *ips_xfr_wait* is asserted to signalize a stall cycle (the IPS master must wait until cycle 14 to read the CRC result).

Figure 10-4. CRC calculation of ASCII message “123456789” (0x31 to 0x39) on CF1Core

10.5 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 10.4.2, “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 11
FlexTimer Module (FTMV1)

11.1 Introduction

The FTM is a six-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.

11.1.1 FTM Module-to-Module Interconnects

11.1.1.1 FTMx Synchronization Triggers

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.

11.1.1.1.1 ADC to FTMx Synchronization triggers

The ADCSC1[COCO] bit of the ADC module can be used as one of the three FTM hardware synchronization triggers. This ADC input is the highest priority of the three hardware sync triggers.

Typical usage is to setup the ADC using the automatic compare function (ADCSC2[ACFE]=1). In this configuration, the COCO bit is only set when the ADC result is either less than or greater than/equal to the compare value.

11.1.1.1.2 ACMPx to FTMx Synchronization Trigger

The ACMPxSC[ACF] bit of the ACMPx modules can be used as one of the three FTM hardware synchronization triggers. The ACMPx input is the second highest priority of the three hardware sync triggers. ACMP1SC[ACF] is connected to FTM1 and ACMP2SC[ACF] is connected to FTM2.

The ACF bit can be configured to set on rising voltage inputs, falling voltage inputs or both rising and falling inputs.

11.1.1.1.3 TPM3 to FTMx Synchronization triggers

The TPM3CHn output of the TPM3 module can be used as one of the three FTM hardware synchronization triggers. TPM3CH0 is connected to FTM1 and TPM3CH1 is connected to FTM2.

Typical usage is to setup the TPM3 channel as an output compare. The sync trigger occurs on the rising edge of the TPM3 channel, so the “set output on compare” option (TPM3CnSC[ELSnB:ELSnA]=0b11) must be used.

11.1.1.2 ACMP1 to FTM1 input capture

The ACMP1 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.

11.1.1.3 FTM1 to ADC Hardware Trigger

The FTM1 module has the capability to send a hardware trigger to the ADC using the FTM1_ADC_TRIGGER register. FTM1 can be configured to send a trigger on any channel value match from channels 2-5 (channels 0-1 cannot be used). Multiple channels can be selected at once to generate multiple triggers to the ADC within one period of the FTM counter.

The ADC hardware trigger is enabled by the ADCSC2[ADTRG] bit. In addition, the hardware trigger source must be selected with the SOPT2[ADHWTS1:ADHWTS0] bits. ADHWTS1:ADHWTS0 = 0b10 selects the FTM1 as the ADC hardware trigger.

11.1.2 Bits in External Trigger Registers (FTMxEXTTRIG)

The External Trigger Register (EXTTRIG) is not available in FTM2. The INITTRIGEN bit in FTM1EXTTRIG is not available.

11.1.3 Features

The FTM features include:

- FTM source clock is selectable
  - Source clock can be the system clock, the fixed system clock or a clock from an external pin
  - Fixed system clock source is synchronized to the system clock by an external synchronization circuit to FTM
  - External clock pin may be shared with any FTM channel pin or a separated input pin
  - External clock source is synchronized to the system clock by FTM
- 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 to ADC (hardware trigger)
- Software control of PWM outputs
- A fault input 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

11.1.4 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. Provided the FTM does not need to produce a real time reference or provide the interrupt source(s) needed to wake the MCU from wait mode, then power can be saved by disabling FTM functions before entering wait mode.

11.1.5 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). The FTM 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 11-1 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.

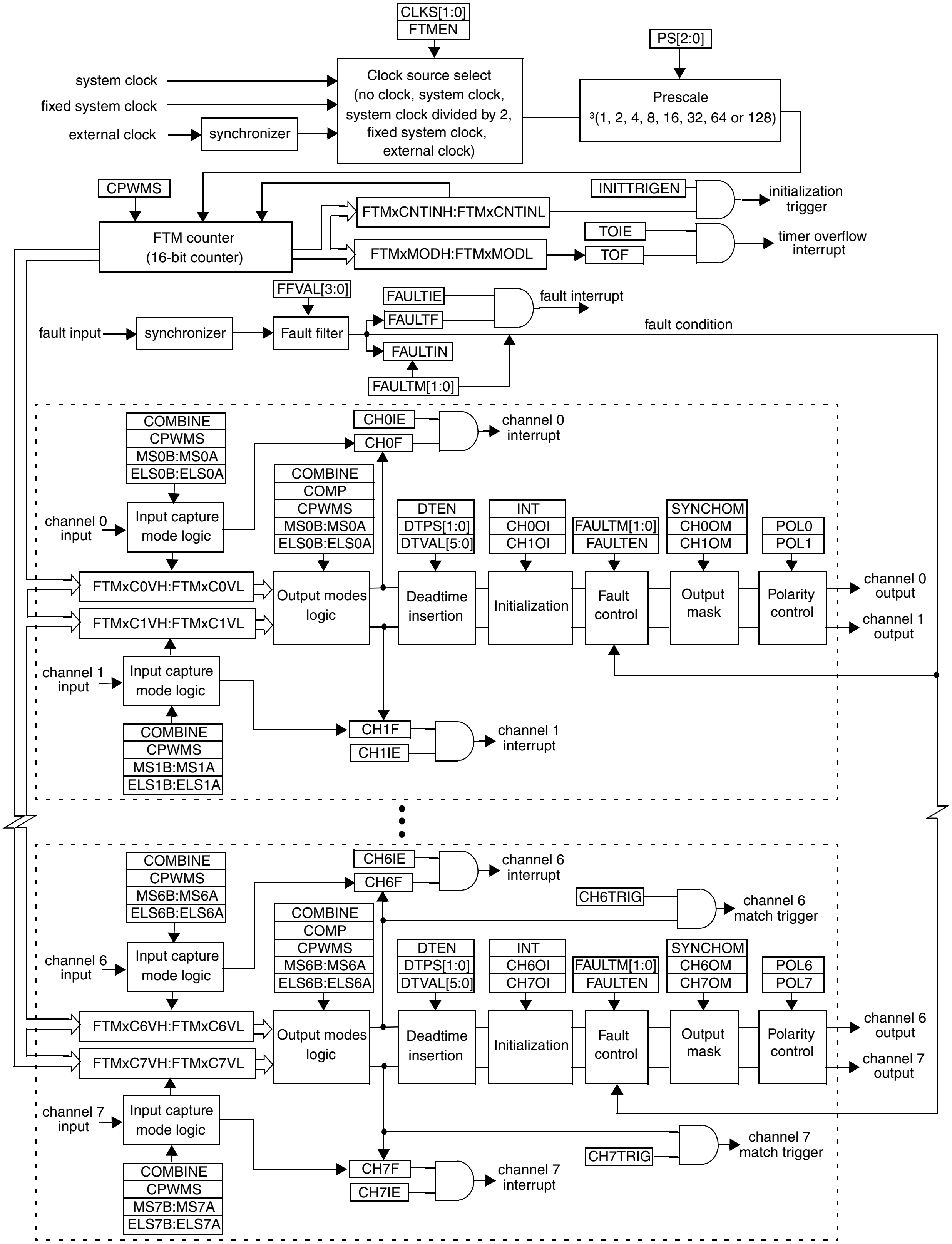


Figure 11-1. FTM Block Diagram

11.2 Signal Description

Table 11-1 shows the user-accessible signals for the FTM.

Table 11-1. Signal Properties

Name	Function
FTMxCLK[1]	FTM x External Clock - FTM external clock which may be selected to drive the FTM counter
FTMxCHn[2]	FTM x channel (n) - I/O pin associated with FTM channel (n)

[1] When preset, this signal can share any channel pin; however depending upon full-chip implementation, this signal could be connected to a separate external pin.

[2] n=channel number (0 to 7)

11.2.1 FTMxCLK — FTM External Clock

When an external clock is included, it can be shared with the same pin as any FTM channel or it can be connected to a separate input pin. Refer to the I/O pin descriptions in full-chip specification for the specific chip implementation.

If the external clock signal shares the same pin as a channel I/O pin, then this channel pin will not be available for channel I/O functions when the external clock source is selected. It is the user’s responsibility to avoid conflicting settings such as this.

The external clock source is synchronized in the FTM. The system clock clocks the synchronizer, so the frequency of the external source must be no more than one-fourth the frequency of the system clock, to meet Nyquist criteria and allow for jitter.

11.2.2 FTMxCHn — FTM Channel (n) I/O Pin(s)

Each FTM channel is associated with an I/O pin on the MCU. The function of this pin depends on the channel configuration. The FTM pins are shared with general purpose I/O pins, where each pin has a port data register bit, and a data direction control bit, and the port has optional passive pullups which may be enabled whenever a port pin is acting as an input.

Immediately after reset, no FTM functions are enabled, so all FTM channels pins revert to general purpose I/O control. The FTM channels pins also revert to general purpose I/O control when (CLKS = 00) or (ELSnB:ELSnA = 0:0).

11.3 Memory Map and Register Definition

This section provides a detailed description of all FTM registers accessible to the end user.

11.3.1 Module Memory Map

This section presents a high-level view of the FTM registers and how they are mapped. Table 11-2 shows the registers for an 8-channel FTM. Register names include a timer number (x) because it is common for MCU systems to include more than one FTM.

Table 11-2. 8-channel FTM Module Memory Map

Address	Use	Access	Section/Page
Base+0000	FTM Status and Control (FTMxSC)	R/W	11.3.3/-9
Base+0001	FTM Counter Register High (FTMxCNTH)	R[1]	11.3.4/-10
Base+0002	FTM Counter Register Low (FTMxCNTL)	R[1]	11.3.4/-10
Base+0003	FTM Modulo Register High (FTMxMODH)	R/W	11.3.5/-11
Base+0004	FTM Modulo Register Low (FTMxMODL)	R/W	11.3.5/-11
Base+0005	FTM Channel 0 Status and Control (FTMxC0SC)	R/W	11.3.6/-12
Base+0006	FTM Channel 0 Value High (FTMxC0VH)	R/W	11.3.7/-14
Base+0007	FTM Channel 0 Value Low (FTMxC0VL)	R/W	11.3.7/-14
Base+0008	FTM Channel 1 Status and Control (FTMxC1SC)	R/W	11.3.6/-12
Base+0009	FTM Channel 1 Value High (FTMxC1VH)	R/W	11.3.7/-14
Base+000A	FTM Channel 1 Value Low (FTMxC1VL)	R/W	11.3.7/-14
Base+000B	FTM Channel 2 Status and Control (FTMxC2SC)	R/W	11.3.6/-12
Base+000C	FTM Channel 2 Value High (FTMxC2VH)	R/W	11.3.7/-14
Base+000D	FTM Channel 2 Value Low (FTMxC2VL)	R/W	11.3.7/-14
Base+000E	FTM Channel 3 Status and Control (FTMxC3SC)	R/W	11.3.6/-12
Base+000F	FTM Channel 3 Value High (FTMxC3VH)	R/W	11.3.7/-14
Base+0010	FTM Channel 3 Value Low (FTMxC3VL)	R/W	11.3.7/-14
Base+0011	FTM Channel 4 Status and Control (FTMxC4SC)	R/W	11.3.6/-12
Base+0012	FTM Channel 4 Value High (FTMxC4VH)	R/W	11.3.7/-14
Base+0013	FTM Channel 4 Value Low (FTMxC4VL)	R/W	11.3.7/-14
Base+0014	FTM Channel 5 Status and Control (FTMxC5SC)	R/W	11.3.6/-12
Base+0015	FTM Channel 5 Value High (FTMxC5VH)	R/W	11.3.7/-14
Base+0016	FTM Channel 5 Value Low (FTMxC5VL)	R/W	11.3.7/-14
Base+0017	FTM Channel 6 Status and Control (FTMxC6SC)	R/W	11.3.6/-12
Base+0018	FTM Channel 6 Value High (FTMxC6VH)	R/W	11.3.7/-14
Base+0019	FTM Channel 6 Value Low (FTMxC6VL)	R/W	11.3.7/-14

Table 11-2. 8-channel FTM Module Memory Map

Base+001A	FTM Channel 7 Status and Control (FTMxC7SC)	R/W	11.3.6/-12
Base+001B	FTM Channel 7 Value High (FTMxC7VH)	R/W	11.3.7/-14
Base+001C	FTM Channel 7 Value Low (FTMxC7VL)	R/W	11.3.7/-14
Base+001D	FTM Counter Initial Value High (FTMxCNTINH)	R/W	11.3.8/-15
Base+001E	FTM Counter Initial Value Low (FTMxCNTINL)	R/W	11.3.8/-15
Base+001F	FTM Capture and Compare Status Register (FTMxSTATUS)	R/W	11.3.9/-16
Base+0020	FTM Features Mode Selection (FTMxMODE)	R/W	11.3.10/-16
Base+0021	FTM Synchronization (FTMxSYNC)	R/W	11.3.11/-18
Base+0022	FTM Initial State for Channels Output (FTMxOUTINIT)	R/W	11.3.12/-19
Base+0023	FTM Output Mask (FTMxOUTMASK)	R/W	11.3.13/-20
Base+0024	FTM Function For Linked Channels (FTMxCOMBINE0)	R/W	11.3.14/-20
Base+0025	FTM Function For Linked Channels (FTMxCOMBINE1)	R/W	11.3.14/-20
Base+0026	FTM Function For Linked Channels (FTMxCOMBINE2)	R/W	11.3.14/-20
Base+0027	FTM Function For Linked Channels (FTMxCOMBINE3)	R/W	11.3.14/-20
Base+0028	FTM Deadtime Insertion Control (FTMxDEADTIME)	R/W	11.3.15/-21
Base+0029	FTM External Trigger (FTMxEXTTRIG)	R/W	11.3.16/-24
Base+002A	FTM Channels Polarity (FTMxPOL)	R/W	11.3.17/-25
Base+002B	FTM Fault Mode Status (FTMxFMS)	R/W	11.3.18/-25
Base+002C	FTM Input Capture Filter Control (FTMxFILTER0)	R/W	11.3.19/-26
Base+002D	FTM Input Capture Filter Control (FTMxFILTER1)	R/W	11.3.19/-26
Base+002E	FTM Fault Input Filter Control (FTMxFLTFILTER)	R/W	11.3.20/-27
Base+002F	FTM Register for Future Use (FTMxRFU)	R	11.3.21/-27

[1] 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.

NOTE

Although it is possible to write to the FTM specific registers (FTM registers between the FTMxCNTINH and FTMxRFU registers) when FTMEN = 0, it is not recommended and the results can be not guaranteed.

11.3.2 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 would be the status and control register for channel 2 of FTM1.

11.3.3 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 prescale factor. These controls relate to all channels within this module.

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKS		PS		
W	0							
Reset	0	0	0	0	0	0	0	0

Figure 11-2. FTM Status and Control Register (FTMxSC)

Table 11-3. FTMxSC Field Descriptions

Field	Description
7 TOF	Timer overflow flag. This read/write flag is set when the FTM counter resets to the value of FTMxCNTINH:FTMxCNTINL registers after reaching the modulo value programmed in the FTMxMODH:FTMxMODL registers. Clear TOF by reading the FTM status and control register when TOF is set and then writing a logic 0 to TOF. If another FTM overflow occurs before the clearing sequence is complete, the sequence is reset so TOF would remain 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. When the FTM is configured for CPWM mode (up-down counter), TOF is set after the FTM counter has reached the value in the FTMxMODH:FTMxMODL registers, at the transition to the next lower count value. 0 FTM counter has not reached the modulo value or overflowed. 1 FTM counter has overflowed.
6 TOIE	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. 0 TOF interrupts inhibited (use software polling). 1 TOF interrupts enabled.
5 CPWMS	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. 0 FTM counter operates in up counting mode. 1 FTM counter operates in up-down counting mode.
4–3 CLKS	Clock source select. As shown in Table 11-4, this 2-bit field is used to disable the FTM counter or select one of four clock sources to drive the counter prescaler. The fixed system clock source is only meaningful in systems with a PLL or FLL-based system clock. When there is no PLL and no FLL, the fixed-system clock source is the same as the system clock. The external source is synchronized to the system clock by FTM, and the fixed system clock source (when a PLL or a FLL is present) is synchronized to the system clock by an on-chip synchronization circuit. When a PLL or a FLL is present but not enabled, the fixed-system clock source is the same as the system clock.
2–0 PS	Prescale factor select. This 3-bit field selects one of 8 division factors for the FTM clock input as shown in Table 11-5. This prescaler is located after any clock source synchronization or clock source selection so it affects the clock source selected to drive the FTM system. The new prescaler factor will affect the clock source on the next system clock cycle after the new value is updated into the register bits.

Table 11-4. 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 (for TPM compatibility)
	1	System clock
10	X	Fixed system clock
11	X	External source

Table 11-5. 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

11.3.4 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 11-3. 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 11-4. FTM Counter Register Low (FTMxCNTL)

When BDM is active, the FTM counter is frozen (this is the value that will be read by user); 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 the user was in the middle of reading a 16-bit register when BDM became active, it will read the appropriate value from the other half of the 16-bit value after returning to normal execution.

11.3.5 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 11.4.3, “Counter).

Reset sets the FTM counter modulo registers to 0x0000.

Writing to FTMxMODH or FTMxMODL inhibits the TOF bit and overflow interrupts until these registers are updated with the value of their write buffer (11.4.10, “Load of the registers with write buffers).

NOTE

> The update of the TOF bit and the generation of the overflow interrupt are disabled when there is a write to FTMxMODH or FTMxMODL. This condition will remain until the FTMxMODH:L registers are updated with the contents of their write buffers. This could be several timer count periods.

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 11.4.10, “Load of the registers with write buffers.

If FTMEN = 0, then this write coherency mechanism may be manually reset by writing to the 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.

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

Figure 11-5. FTM Counter Modulo Register High (FTMxMODH)

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

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.

11.3.6 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.

	7	6	5	4	3	2	1	0
R	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
W	0						▨	▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-7. FTM Channel (n) Status and Control Register (FTMxCnSC)

Table 11-6. FTMxCnSC Field Descriptions

Field	Description
7 CHnF	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) pin. When channel (n) is an output compare or edge-aligned PWM channel, CHnF is set when the value in the FTM counter matches the value in the FTM channel (n) value registers. When channel (n) is center-aligned PWM channel, CHnF is set twice during each CPWM cycle (one at the start and another at the end of the active duty cycle period) when the value in the FTM counter matches the value in the FTM 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 will not be set even when the value in the FTM counter registers matches the value in the FTM channel (n) value registers. When channel (n) is in combine mode, CHnF is always set when the value in the FTM counter matches the value in the FTM channel. A corresponding interrupt is requested when CHnF is set and interrupts are enabled (CHnIE = 1). Clear CHnF by reading FTMxCnSC while CHnF is set and then writing a logic 0 to CHnF. If another interrupt request occurs before the clearing sequence is complete, the sequence is reset so CHnF remains set after the clear sequence is completed for the earlier CHnF. This is done so a CHnF interrupt request cannot be lost due to clearing a previous CHnF. Reset clears the CHnF bit. Writing a logic 1 to CHnF has no effect. 0 No input capture or match event occurred on channel (n) 1 Input capture or match event on channel (n)
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 for FTM channel (n). Refer to the summary of channel mode and setup controls in Table 11-7. MSnB bit is write protected, this bit can only be written if WPDIS = 1.

Table 11-6. FTMxCnSC Field Descriptions (continued)

Field	Description
4 MSnA	Mode select A for FTM channel (n). Refer to the summary of channel mode and setup controls in Table 11-7. MSnA bit is write protected, this bit can only be written if WPDIS = 1. **Note:** If the associated port pin is not stable for at least two system 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 COMBINE:CPWMS:MSnB:MSnA bits and shown in Table 11-7, these bits can select which edge of the channel input signal is an input capture event or the level that will be driven in response to the channel or initial match according to the selected output mode. ELSnB and ELSnA bits are write protected, these bits can only be written if WPDIS = 1. Setting ELSnB:ELSnA to 0:0 configures the related FTM pin as a general purpose I/O pin not related to any FTM functions. This function is typically used to temporarily disable an input capture channel or to make the FTM pin available as a general purpose I/O pin. When ELSnB:ELSnA are set to 0:0 the associated FTM channel physical output is disabled however, compare and match events will continue to set the appropriate flags.

Table 11-7. 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	High-true pulses (clear output on match-up)
			X1		Low-true pulses (set output on match-up)

Table 11-7. Mode, Edge, and Level Selection (continued)

COMBINE	CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
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)

11.3.7 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.

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

Figure 11-8. FTM Channel Value Register High (FTMxCnVH)

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

Figure 11-9. 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 will be 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 the user was in the middle of reading a 16-bit register when BDM became active, it will read 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 11.4.10, “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 written (while BDM was not active) by the user.

11.3.8 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.

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

Figure 11-10. FTM Counter Initial Value Register High (FTMxCNTINH)

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

Figure 11-11. 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 will start 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).

11.3.9 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.

	7	6	5	4	3	2	1	0
R	CH7F	CH6F	CH5F	CH4F	CH3F	CH2F	CH1F	CH0F
W	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-12. FTM Capture and Compare Status Register (FTMxSTATUS)

Table 11-8. FTMxSTATUS Field Descriptions

Field	Description
7-0 CHnF	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) pin. When channel (n) is an output compare or edge-aligned/center-aligned PWM channel, CHnF is set when the value in the FTM counter registers matches the value in the FTM 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 will not be set even when the value in the FTM counter registers matches the value in the FTM channel (n) value registers. When channel (n) is in combine mode, CHnF is always set when the value in the FTM counter matches the value in the FTM channel (n) value. Clear CHnF by reading FTMxSTATUS while CHnF is set and then writing a logic 0 to the selected CHnF. If another request to set the CHnF bit occurs before the clearing sequence is complete, the sequence is reset so CHnF remains set after the clear sequence is completed for the earlier CHnF. Reset clears the CHnF bit. Writing a logic 1 to CHnF has no effect. 0 No input capture or match event occurred on channel (n) 1 Input capture or match event on channel (n) **Note:** Each CHnF bit in FTMxSTATUS register is a copy of CHnF bit in FTMxCnSC register. All CHnF bits can be checked using only one read of FTMxSTATUS register. And all CHnF bits can be cleared by a read of FTMxSTATUS register followed by a write 0x00 to FTMxSTATUS register.

NOTE

The use of FTMxSTATUS register is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). The use of this register with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.3.10 FTM Features Mode Selection Register (FTMxMODE)

This read/write register contains the control bits used to configure the fault interrupt and fault control mode, write protection, channel output initialization and enable the enhanced features of the FTM. These controls relate to all channels within this module.

	7	6	5	4	3	2	1	0
R	FAULTIE	FAULTM		0	0	WPDIS	0	FTMEN
W				▨	▨		INIT	
Reset	0	0	0	0	0	1	0	0

▨ = Unimplemented or Reserved

Figure 11-13. FTM Features Mode Selection Register (FTMxMODE)

Table 11-9. FTMxMODE Field Descriptions

Field	Description
7 FAULTIE	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. 0 Fault control interrupt is disabled. 1 Fault control interrupt is enabled. FAULTIE bit is write protected, this bit can only be written if WPDIS = 1.
6-5 FAULTM	Fault control mode. These bits define the FTM fault control mode. The fault control mode is selected according to the Table 11-10. The FAULTM[1:0] bits are write protected, these bits can only be written if WPDIS = 1.
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 (FTMxCNTINH through to FTMxRFU) are available for use with no restrictions. When FTMEN = 0 only the TPM compatible registers (FTMxSC through to FTMxC7VL) 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. 0 Only TPM compatible registers (FTMxSC through to FTMxC7VL) are available. 1 All FTM registers are available.

Table 11-10. Fault Control Mode Selection

FAULTM	Fault Control Mode
00	Fault control mode is disabled for all channels
01	Fault control mode is enabled for even channels only (channels 0, 2, 4 and 6) and the selected mode is the manual fault clearing
10	Fault control mode is enabled for all channels and the selected mode is the manual fault clearing
11	Fault control mode is enabled for all channels and the selected mode is the automatic fault clearing

11.3.11 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.

	7	6	5	4	3	2	1	0
R W	SWSYNC	TRIG2	TRIG1	TRIG0	SYNCHOM	REINIT	CNTMAX	CNTMIN
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-14. FTM Synchronization Register (FTMxSYNC)

Table 11-11. 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.

Table 11-11. FTMxSYNC Field Descriptions (continued)

Field	Description
2 REINIT	Reinitialization of the FTM by PWM synchronization (Section 11.4.11, “PWM synchronization). REINIT bit determines if the FTM is initialized when the selected trigger for the PWM synchronization is detected. When REINIT = 1, the FTM counter is loaded with its initial value (FTMxCNTINH:L) and the registers FTMxMODH:L and FTMxCnVH:L are updated with their write buffer contents when the synchronization trigger occurs. The FTM reinitialization is independent of the boundary cycles. When REINIT = 0, registers FTMxMODH:L and FTMxCnVH:L are updated with their write buffer contents at the selected boundary cycle after the synchronization trigger occurs. The FTM counter is not affected and continues to count normally. 0 FTM counter countinues to count normally. FTMxMODH:L and FTMxCnVH:L are updated with their write buffer contents on the selected boundary cycle after the selected trigger is detected. 1 FTM counter is updated with its initial value and FTMxMODH:L and FTMxCnVH:L are updated with their write buffer contents 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. 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 TRIG0, TRIG1 or TRIG2 bits is likely to result in an unpredictable behavior.

11.3.12 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).

	7	6	5	4	3	2	1	0
R W	CH7OI	CH6OI	CH5OI	CH4OI	CH3OI	CH2OI	CH1OI	CH0OI
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-15. FTM Initial State for Channels Output Register (FTMxOUTINIT)

Table 11-12. 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.

11.3.13 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 will respond (that is, it will be 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 11.4.11, “PWM synchronization.

	7	6	5	4	3	2	1	0
R W	CH7OM	CH6OM	CH5OM	CH4OM	CH3OM	CH2OM	CH1OM	CH0OM
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-16. FTM Output Mask Register (FTMxOUTMASK)

Table 11-13. 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.

11.3.14 FTM Function For Linked Channels Register (FTMxCOMBINEm)

This read/write register contains the control bits used to configure the fault control, synchronization, deadtime, 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.

	7	6	5	4	3	2	1	0
R	0	FAULTEN	SYNCEN	DTEN	0	0	COMP	COMBINE
W	▨				▨	▨		
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-17. FTM Function For Linked Channels Register (FTMxCOMBINEm)

Table 11-14. 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.
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.

11.3.15 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.

	7	6	5	4	3	2	1	0
R W	DTPS		DTVAL					
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-18. FTM Deadtime Insertion Control Register (FTMxDEADTIME)

Table 11-15. 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 Table 11-16). 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). 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 11-16. Deadtime Prescale Factor Selection

DTPS	System Clock Divided-by
0X	1
10	4
11	16

Table 11-17. 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

Table 11-17. Deadtime Value (number of system clock periods) (continued)

DTVAL	DTPS = 0X	DTPS = 10	DTPS = 11
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
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

Table 11-17. Deadtime Value (number of system clock periods) (continued)

DTVAL	DTPS = 0X	DTPS = 10	DTPS = 11
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

11.3.16 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 0 and 1 are not used to generate channel triggers.

	7	6	5	4	3	2	1	0
R	TRIGF	INITTRIGEN	CH7TRIG	CH6TRIG	CH5TRIG	CH4TRIG	CH3TRIG	CH2TRIG
W	0							
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-19. FTM External Trigger Register (FTMxEXTTRIG)

Table 11-18. 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 complete, 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 = 7, 6, 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.

11.3.17 FTM Channels Polarity Register (FTMxPOL)

This read/write register defines the output polarity of the FTM channels.

	7	6	5	4	3	2	1	0
R W	POL7	POL6	POL5	POL4	POL3	POL2	POL1	POL0
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-20. FTM Channels Polarity Register (FTMxPOL)

Table 11-19. 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.

11.3.18 FTM Fault Mode Status Register (FTMxFMS)

This read/write register contains the fault detection flag, write protection enable and the fault input bits.

	7	6	5	4	3	2	1	0
R	FAULTF	WPEN	FAULTIN	0	0	0	0	0
W	0		▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-21. FTM Fault Mode Status Register (FTMxFMS)

Table 11-20. FTMxFMS Field Descriptions

Field	Description
7 FAULTF	Fault detection flag. The FAULTF bit is set when fault control is enabled and a fault condition is detected. 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 before the clearing sequence is complete, the sequence is reset so FAULTF remains set after the clearing sequence is completed for the earlier fault condition. 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 input. The FAULTIN bit is the value of the fault input after the fault filter 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.

11.3.19 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 and 5 do not have an input filter.

	7	6	5	4	3	2	1	0
R W	CH(n+1)FVAL				CH(n)FVAL			
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-22. FTM Input Capture Filter Control Register (FTMxFILTERm)

Table 11-21. 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 filer. 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 should only be done when the input capture modes of channels (n) and (n+1) are disabled. Failure to do so could result in a missing valid signal.

11.3.20 FTM Fault Input Filter Control Register (FTMxFLTFILTER)

This read/write register selects the filter value for the fault input.

	7	6	5	4	3	2	1	0
R	0	0	0	0	FFVAL			
W	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 11-23. FTM Fault Input Filter Control Register (FTMxFLTFILTER)

Table 11-22. FTMxFLTFILTER Field Descriptions

Field	Description
3-0 FFVAL	Fault input filter. FFVAL[3:0] bits select the filter value for the fault input. The fault filter is disabled when FFVAL[3:0] is zero.

NOTE

Writing to this register has immediate effect and should only be done when the fault control is disabled. Failure to do so could result in a missing fault detection.

11.3.21 FTM Register for Future Use (FTMxRFU)

This register is reserved, however it will be used in the next versions of FTM. FTMxRFU register is always read as 0x00.

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

▨ = Unimplemented or Reserved

Figure 11-24. FTM Register for Future Use (FTMxRFU)

11.4 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 11-25.

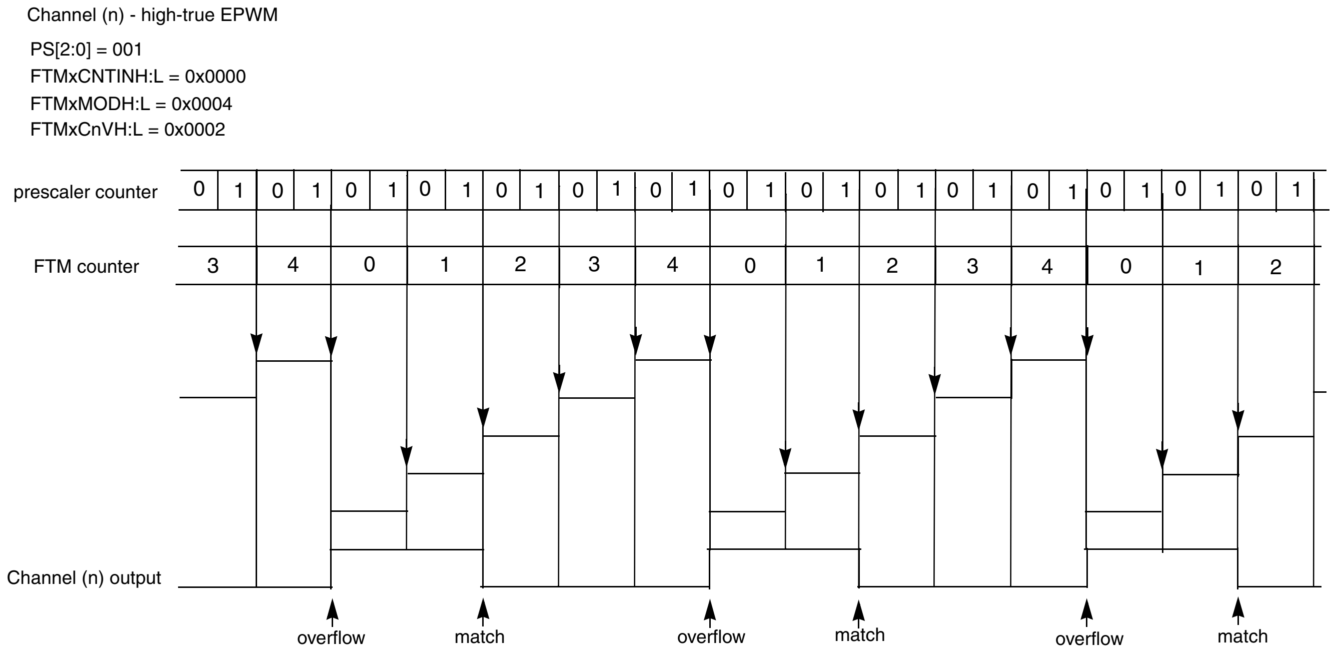


Figure 11-25. Notation used in the FTM Block Guide

11.4.1 Clock Source

FTM module has only one clock domain which is the system clock.

11.4.1.1 Counter Clock Source

The CLKS[1:0] bits in the FTMxSC register select one of four possible clock sources for the FTM counter or disables the FTM counter, see Table 11-4. After any MCU reset, CLKS[1:0] = 00 so no clock source is selected, and the FTM is in a very low power state.

The CLKS[1:0] bits may be read or written at any time and disabling the FTM counter (writing 00 to the CLKS[1:0]) does not affect the FTM counter value or other registers.

The system clock and system clock divided by 2 require no synchronization. The system clock divided by 2 is provided for TPM compatibility.

The fixed system clock requires no synchronization in FTM module. In fact, a synchronization circuit is used at chip level to synchronize the crystal-related source clock to the system clock.

The external clock source may be connected to any FTM channel pin. This clock source always has to pass through a synchronizer to assure that counter transitions are properly aligned to system clock transitions. The system clock drives the synchronizer; therefore, to meet Nyquist criteria even with jitter, the frequency of the external clock source must not exceed 1/4 of the system clock frequency. With ideal clocks, the external clock can be as fast as system clock divided by four.

When the external clock source is shared with an FTM channel pin, this pin should not be used for other channel timing functions. For example, it would be ambiguous to configure channel 0 for input capture when the FTM channel 0 pin was also being used as the external clock source. It is the user’s responsibility to avoid such settings.

The FTM channel can still be used. When ELSnB:ELSnA are set to 0:0 the associated FTM channel physical output is disabled however, compare and match events will continue to set the appropriate flags.

11.4.2 Prescaler

The selected counter clock source passes through a prescaler which is a 7-bit counter. The value of the prescaler is selected by the PS[2:0] bits (Table 11-5). Figure 11-26 shows an example of the prescaler counter and FTM counter.

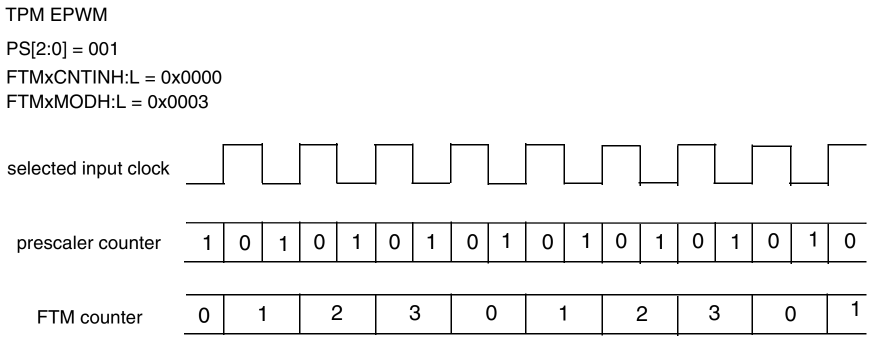


Figure 11-26. Example of the prescaler counter

11.4.3 Counter

The FTM has a 16-bit counter which is used to produce the channels output. The FTM counter clock is the clock generated using the prescaler counter (Section 11.4.2, “Prescaler”).

The FTM counter can be an up or up-down counter according to the CPWMS bit.

11.4.3.1 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 11-27). The value of FTMxCNTINH:FTMxCNTINL is loaded into the FTM counter, and the counter incremented 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 (FTMxMODH:FTMxMODL - FTMxCNTINH:FTMxCNTINL + 0x0001) x period of the FTM counter clock.

The TOF bit is set when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxCNTINH:FTMxCNTINL.

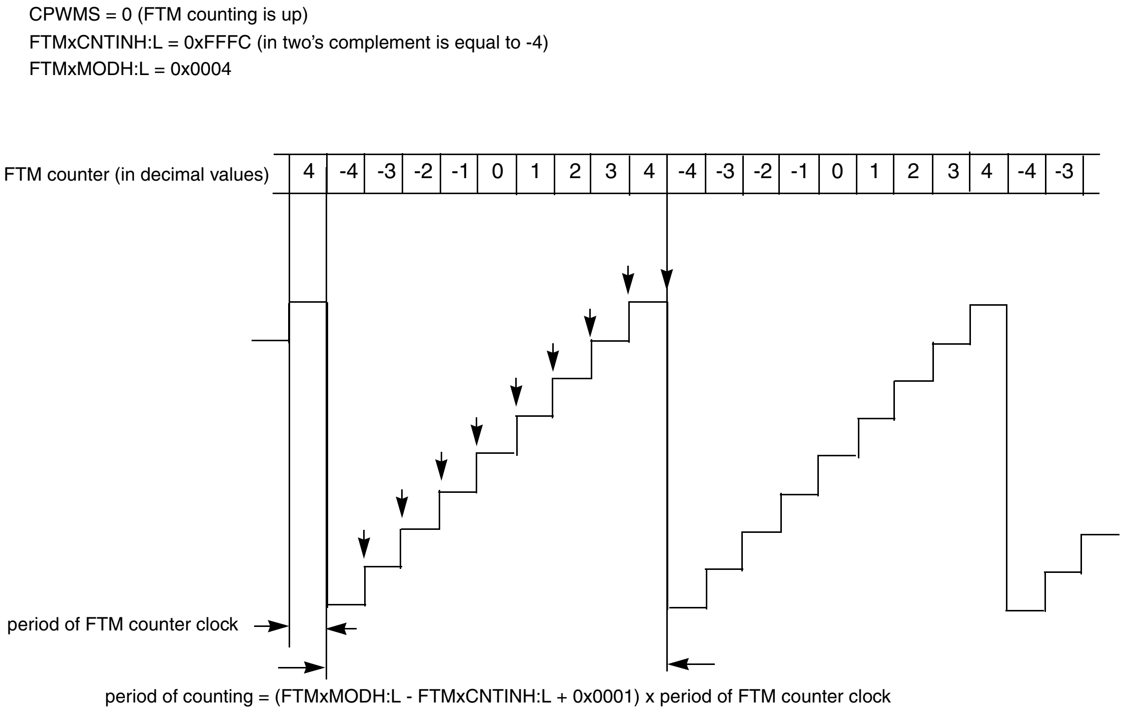

CPWMS = 0 (FTM counting is up)
FTMxCNTINH:L = 0xFFFC (in two's complement is equal to -4)
FTMxMODH:L = 0x0004
FTM counter (in decimal values)
4 -4 -3 -2 -1 0 1 2 3 4 -4 -3 -2 -1 0 1 2 3 4 -4 -3
period of FTM counter clock
period of counting = (FTMxMODH:L - FTMxCNTINH:L + 0x0001) x period of FTM counter clock

Figure 11-27. 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 11-28). 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 not = 0x0000), then the initial value of the FTM counter is a positive number, so the FTM counting is up and unsigned.

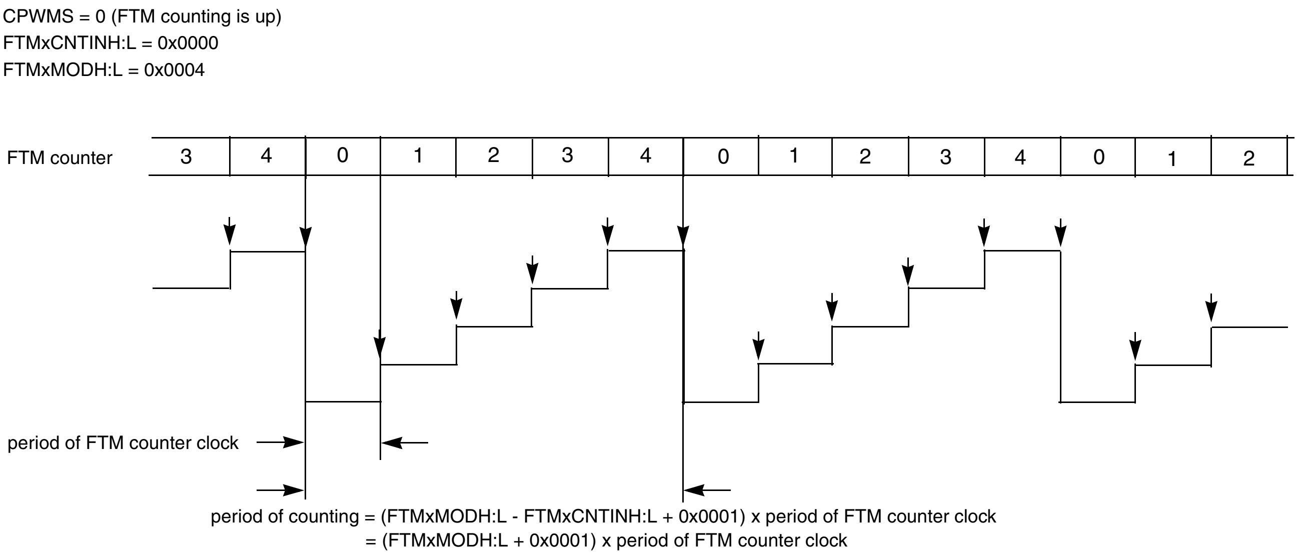

CPWMS = 0 (FTM counting is up)
FTMxCNTINH:L = 0x0000
FTMxMODH:L = 0x0004
FTM counter
3
4
0
1
2
3
4
0
1
2
3
4
0
1
2
period of FTM counter clock
period of counting = (FTMxMODH:L - FTMxCNTINH:L + 0x0001) x period of FTM counter clock
= (FTMxMODH:L + 0x0001) x period of FTM counter clock

Figure 11-28. 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 will be set in each rising edge of the FTM counter clock.

When FTMxMODH:FTMxMODL = 0x0000 and 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 for users to comprehend. There is no restriction on this configuration however, and an example is shown in Figure 11-29.

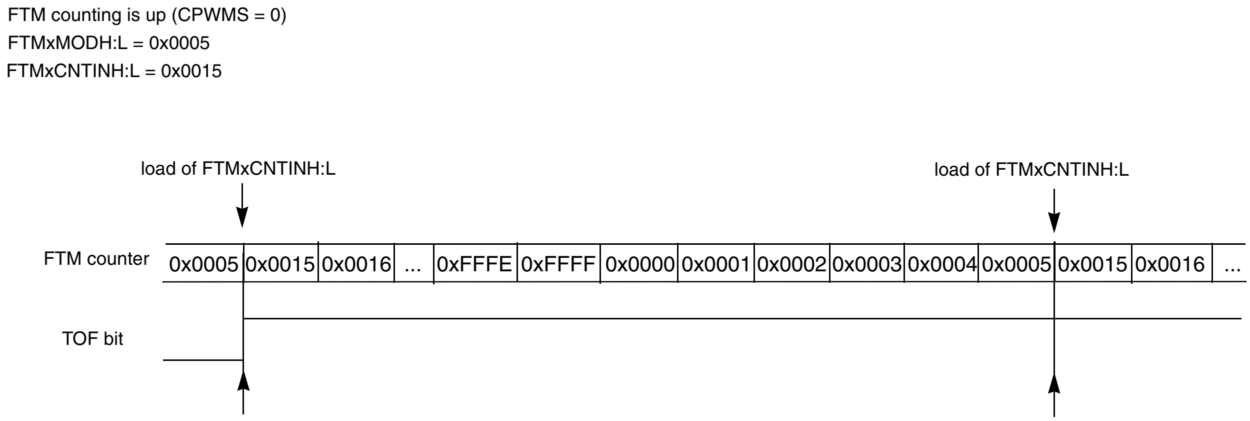


Figure 11-29. Example of up counting when the value of FTMxCNTINH:FTMxCNTINL registers is greater than the value of FTMxMODH:FTMxMODL registers

11.4.3.2 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 the final value of the count (Figure 11-30). The value of FTMxCNTINH:FTMxCNTINL is loaded into the FTM counter, and the counter incremented 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 x (FTMxMODH:FTMxMODL - FTMxCNTINH:FTMxCNTINL) x 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 11-30).

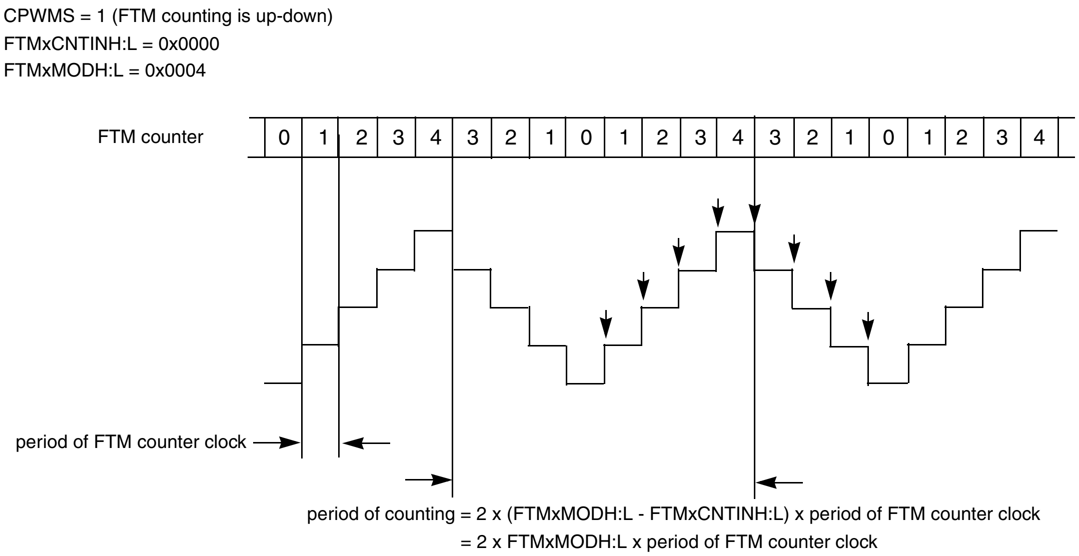

CPWMS = 1 (FTM counting is up-down)
FTMxCNTINH:L = 0x0000
FTMxMODH:L = 0x0004
FTM counter
0 1 2 3 4 3 2 1 0 1 2 3 4 3 2 1 0 1 2 3 4
period of FTM counter clock
period of counting = 2 x (FTMxMODH:L - FTMxCNTINH:L) x period of FTM counter clock
= 2 x FTMxMODH:L x period of FTM counter clock

Figure 11-30. 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 not = 0x0000) when (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.3.3 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 11-31).

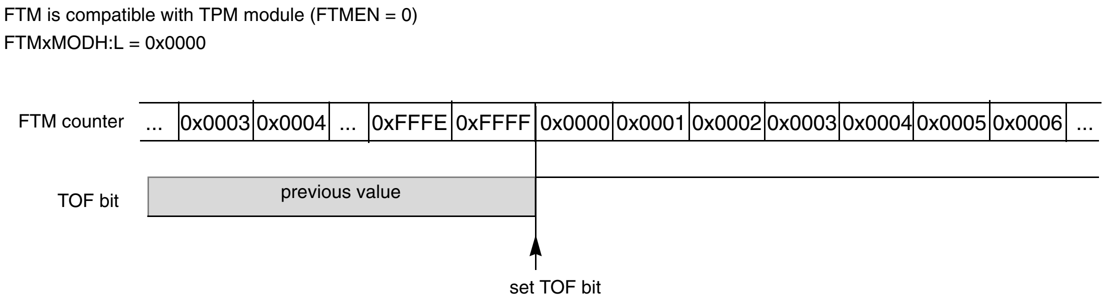

FTM is compatible with TPM module (FTMEN = 0)
FTMxMODH:L = 0x0000
FTM counter
... 0x0003 0x0004 ... 0xFFFE 0xFFFF 0x0000 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 ...
TOF bit
previous value
set TOF bit

Figure 11-31. Example when the FTM counter is a free running

If (FTMEN = 1) and (CPWMS = 0) and (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.

11.4.3.4 Counter reset

Any write to FTMxCNTH or FTMxCNTL register resets the FTM counter to the value in the FTMxCNTINH:FTMxCNTINL registers.

PWM synchronization also can be used to force the value of FTMxCNTINH:FTMxCNTINL into the FTM counter (please see Section 11.4.11, "PWM synchronization).

11.4.4 Input capture mode

The input capture mode is selected when:

- (FTMEN = 0) and (COMBINE = 0) and (CPWMS = 0) and (MSnB:MSnA = 0:0) and (ELSnB:ELSnA not = 0:0)

When a selected edge occurs on the channel input pin, the current value of the FTM counter is captured into the FTMxCnVH:FTMxCnVL registers, the CHnF bit is set and the channel interrupt is generated if (CHnIE = 1)(Figure 11-32). Rising edges, falling edges, any edge, or no edge (disable channel) may be selected as the active edge which triggers the input capture.

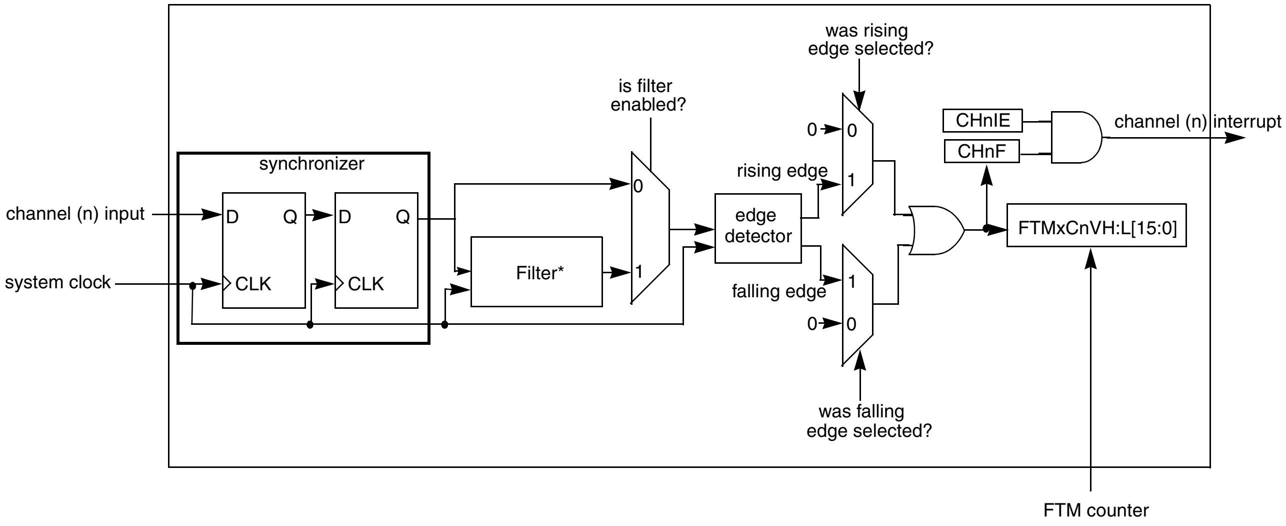
If a channel is configured for input capture, an internal pullup device may be enabled for that channel. The details of how the FTM interacts with pin controls depends upon the micro-controller implementation because the I/O pins and associated general purpose I/O controls are not part of the FlexTimer module. Refer to the I/O port control chaper of the micro-controller specification.

When a channel is configured for input capture, the FTMxCHn pin is forced to act as an edge-sensitive input to the FTM. ELSnB:ELSnA control bits determine which polarity edge or edges will trigger input-capture events. A synchronizer based on the system clock is used to synchronize input edges to the system clock. This implies the minimum pulse width that can be reliably detected on an input capture pin is four system clock periods (with ideal clock pulses as near as two sytem clocks can be detected). FTM uses this pin as a capture input to override the port data and data direction controls for the same pin.

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 by the user. 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 11-32. 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 pin .

NOTE

Input capture mode is only available when (FTMEN = 0) and (FTMxCNTINH:FTMxCNTINL = 0x0000). Input capture mode with (FTMEN = 1) or (FTMxCNTINH:FTMxCNTINL not = 0x0000) is not recommended and its results are not guaranteed.

11.4.4.1 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 11-32). Following synchronization, the input signal enters the filter block (Figure 11-33). 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 will start counting again. Any pulse that is shorter than the minimum value selected by CHnFVAL[3:0] bits (x 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 11-34.

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] not = 0000) then the input signal is delayed by the minimum pulse width (CHnFVAL[3:0] x 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 x CHnFVAL[3:0]) system clock periods after a valid edge occurs on the channel input pin.

The clock for the 5-bit counter in the channel input filter is the system clock divided by 4.

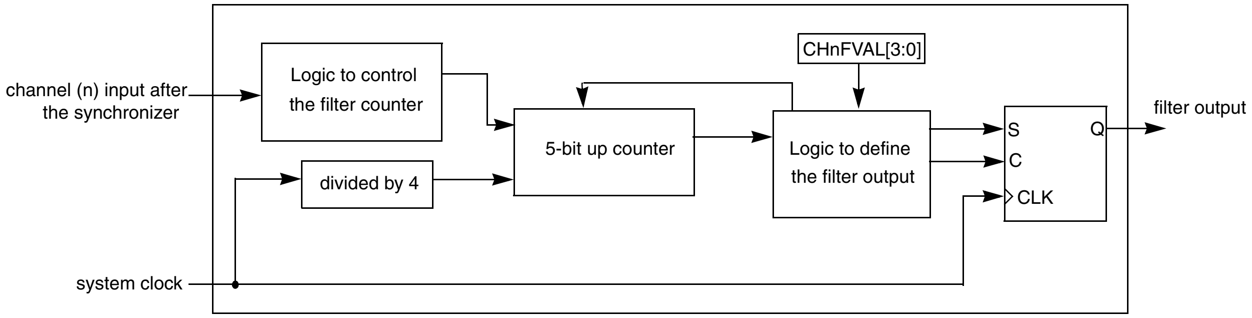

channel (n) input after the synchronizer
Logic to control the filter counter
divided by 4
5-bit up counter
CHnFVAL[3:0]
Logic to define the filter output
S
C
CLK
Q
filter output
system clock

Figure 11-33. Channel Input Filter

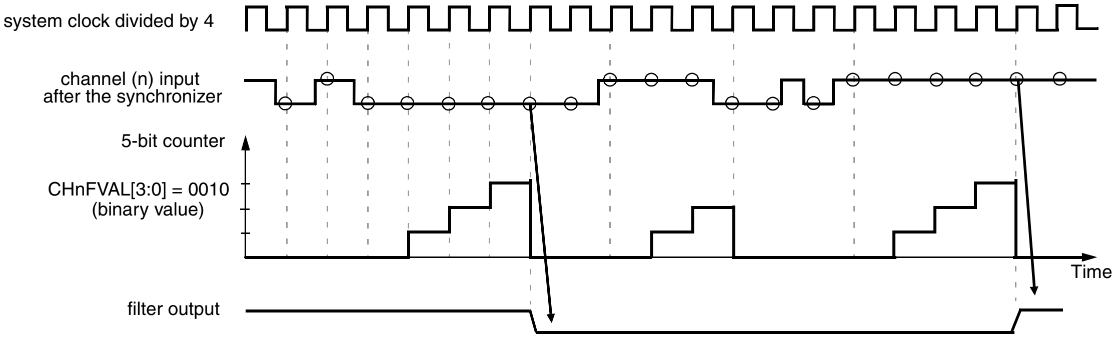

system clock divided by 4
channel (n) input after the synchronizer
5-bit counter
CHnFVAL[3:0] = 0010 (binary value)
Time
filter output

Figure 11-34. Channel input filter example

11.4.5 Output compare mode

The output compare mode is selected when:

- (FTMEN = 0) and (COMBINE = 0) and (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 TOF bit is set and the timer overflow interrupt is generated (if TOIE = 1) at the end of the PWM period (when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxCNTINH:FTMxCNTINL).

The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the channel (n) match (when the FTM counter = FTMxCnVH:FTMxCnVL).

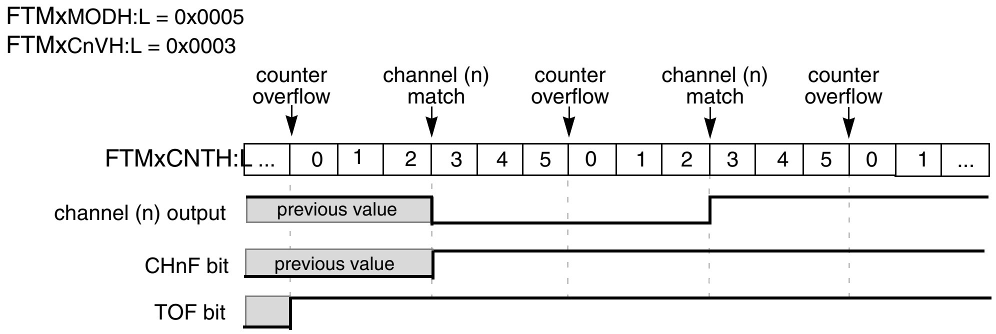


Figure 11-35. Example of the output compare mode when the match toggles the channel output

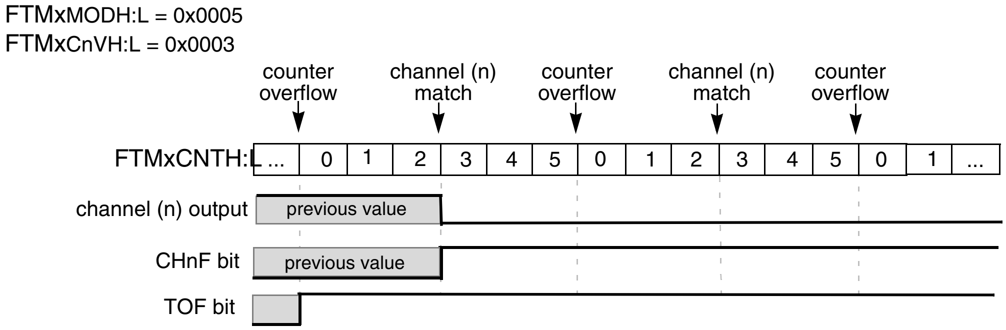


Figure 11-36. Example of the output compare mode when the match clears the channel output

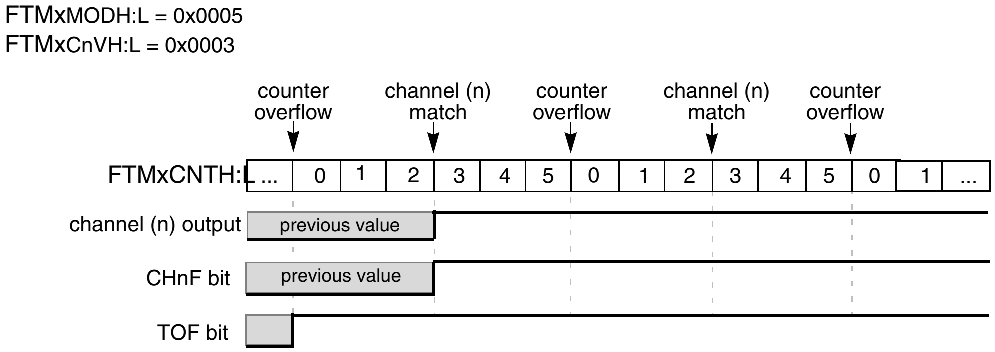


Figure 11-37. Example of the output compare mode when the match sets the channel output

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 when (FTMEN = 0) and (FTMxCNTINH:FTMxCNTINL = 0x0000). Output compare mode with (FTMEN = 1) or (FTMxCNTINH:FTMxCNTINL not = 0x0000) is not recommended and its results are not guaranteed.

11.4.6 Edge-aligned PWM (EPWM) mode

The edge-aligned mode is selected when:

- (FTMEN = 0) and (COMBINE = 0) and (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 TOF bit is set and the timer overflow interrupt is generated (if TOIE = 1) at the end of the EPWM period (when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxCNTINH:FTMxCNTINL). The CHnF bit is set and the channel (n) interrupt is generated (if CHnIE = 1) at the end of the pulse width, that is, at the channel (n) match (when the FTM counter = FTMxCnVH:FTMxCnVL).

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.

EPWM mode can be used when other channels in the same FTM are configured for input capture, output compare or combine modes.

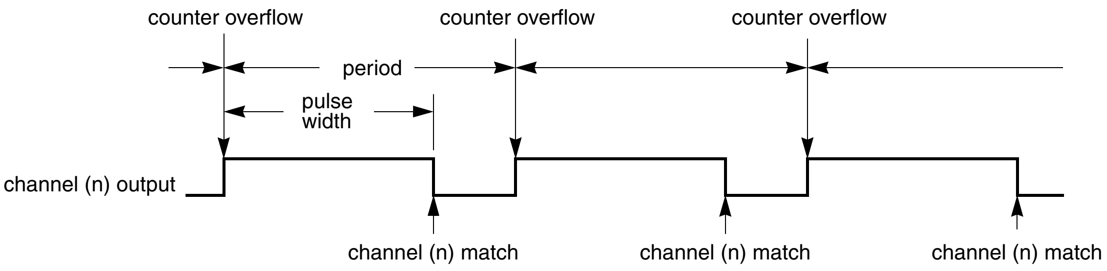


Figure 11-38. EPWM period and pulse width with ELSnB:ELSnA = 1:0

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 11-39).

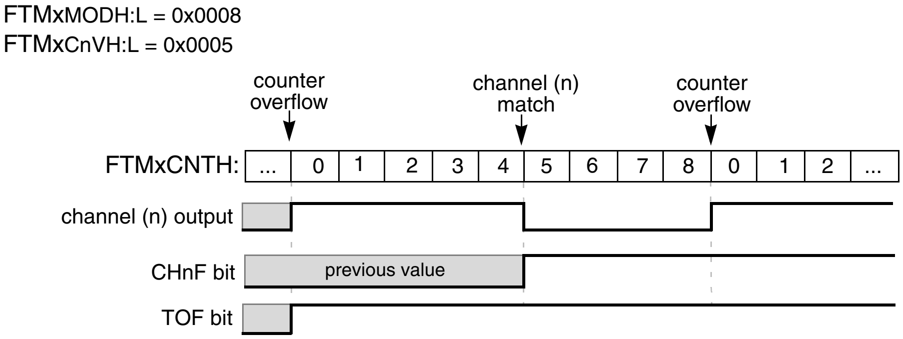


Figure 11-39. EPWM signal with ELSnB:ELSnA = 1:0

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 11-40).

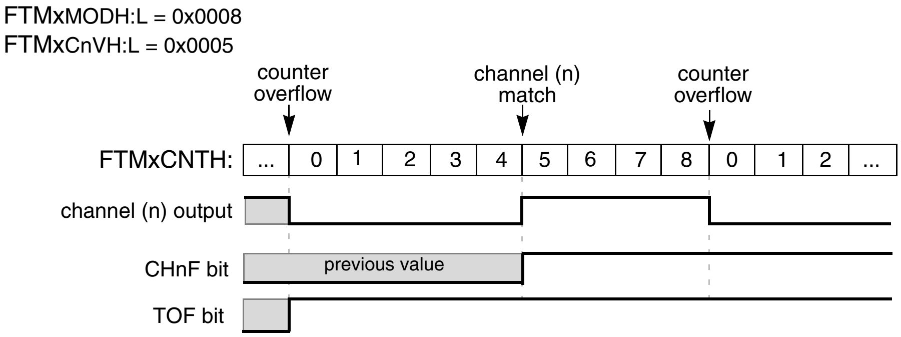


Figure 11-40. EPWM signal with ELSnB:ELSnA = X:1

If (FTMxCnVH:FTMxCnVL = 0x0000) then the channel (n) output is a 0% duty cycle EPWM signal. If (FTMxCnVH:FTMxCnVL > FTMxMODH:FTMxMODL) then the channel (n) output is a 100% duty cycle EPWM signal. This implies that the modulus setting must be less than 0xFFFF in order to get a 100% duty cycle EPWM signal.

NOTE

EPWM mode is only available when (FTMEN = 0) and (FTMxCNTINH:FTMxCNTINL = 0x0000). EPWM mode with (FTMEN = 1) or (FTMxCNTINH:FTMxCNTINL not = 0x0000) is not recommended and its results are not guaranteed.

11.4.7 Center-aligned PWM (CPWM) mode

The center-aligned mode is selected when:

- (FTMEN = 0) and (COMBINE = 0) and (CPWMS = 1)

The CPWM pulse width (duty cycle) is determined by 2 x (FTMxCnVH:FTMxCnVL - FTMxCNTINH:FTMxCNTINL) and the period is determined by 2 x (FTMxMODH:FTMxMODL -

FTMxCNTINH:FTMxCNTINL)(Figure 11-41). FTMxMODH:FTMxMODL should 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 TOF bit is set and the timer overflow interrupt is generated (if TOIE = 1) at the end of the CPWM period (when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxMODH:FTMxMODL - 0x0001). The CHnF bit is set and channel (n) interrupt is generated (if CHnIE = 1) at the begin of the pulse width (when there is a channel (n) match while the FTM counting is down) and at the end of the pulse width (when there is a channel (n) match while the FTM counting is up).

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.

Input capture, output compare, EPWM and combine modes are not compatible with a counter operating in up/down counting mode (CPWMS = 1). Therefore all active channels within an FTM must be used in CPWM mode.

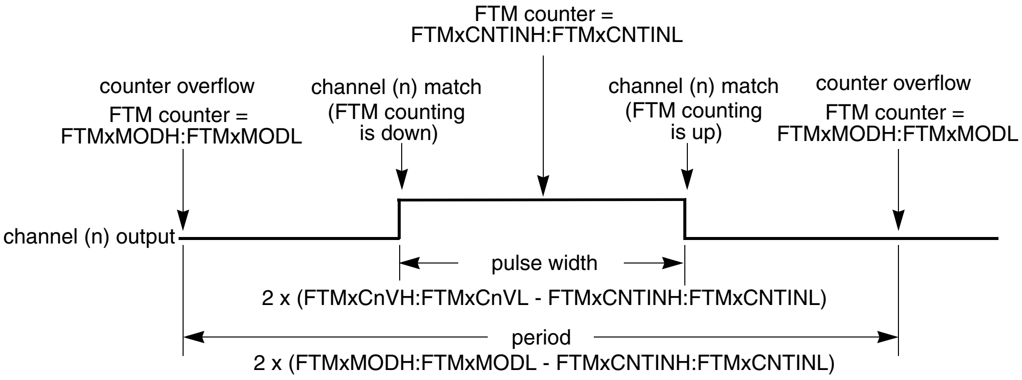


Figure 11-41. CPWM period and pulse width with ELSnB:ELSnA = 1:0

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 11-42).

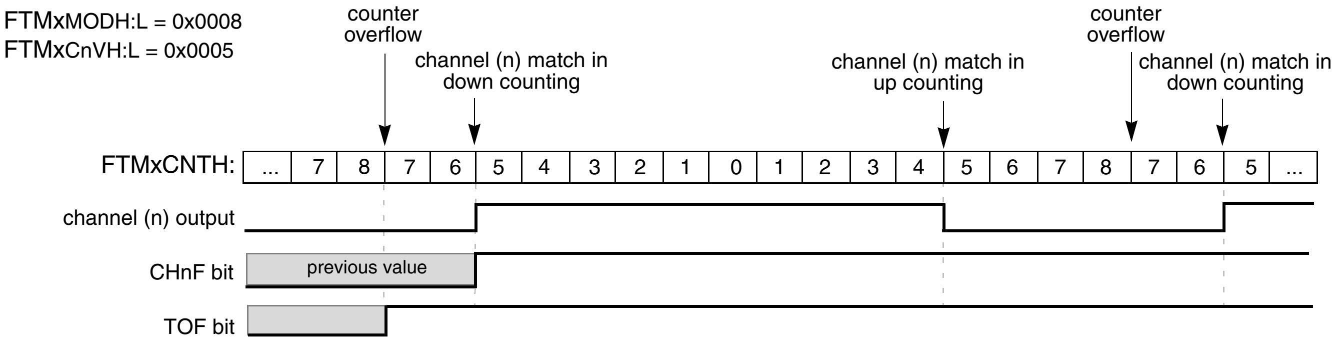


Figure 11-42. CPWM signal with ELSnB:ELSnA = 1:0

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 11-43).

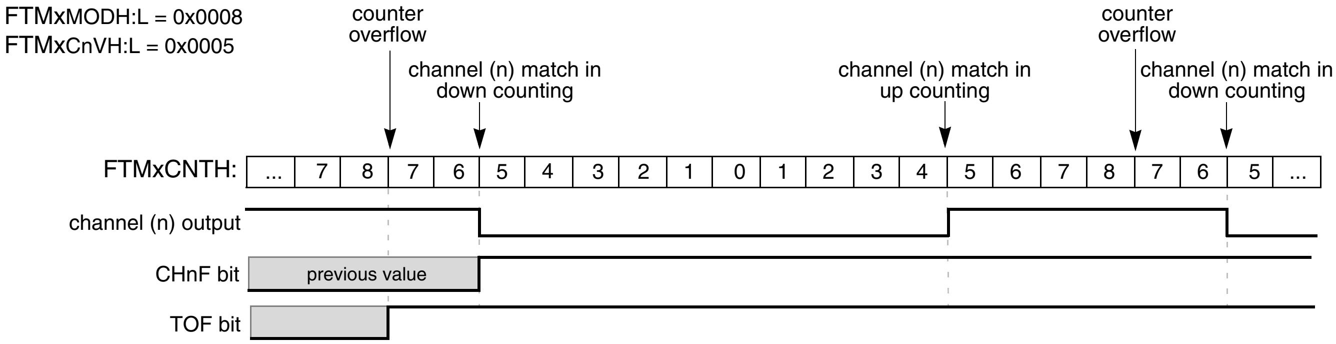


Figure 11-43. CPWM signal with ELSnB:ELSnA = X:1

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.

If (FTMxCnVH:FTMxCnVL is a positive value, that is, FTMxCnVH[7] = 0) and (FTMxCnVH:FTMxCnVL >= FTMxMODH:FTMxMODL) and (FTMxMODH:FTMxMODL not = 0x0000) then the channel (n) output is a 100% duty cycle CPWM signal. 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 cyle CPWM signal). This is not a significant limitation because the resulting period is much longer than required for normal applications.

The CPWM mode should not be used when the FTM counter is a free running counter.

NOTE

CPWM mode is only available when (FTMEN = 0) and (FTMxCNTINH:FTMxCNTINL = 0x0000). CPWM mode with (FTMEN = 1) or (FTMxCNTINH:FTMxCNTINL not = 0x0000) is not recommended and its results are not guaranteed.

11.4.8 Combine mode

The combine mode is selected when:

- (FTMEN = 1) and (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 TOF bit is set and the timer overflow interrupt is generated (if TOIE = 1) at the end of the PWM period (when the FTM counter changes from FTMxMODH:FTMxMODL to FTMxCNTINH:FTMxCNTINL). 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 11-44).

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 11-44).

In combine mode the ELS(n+1)B and ELS(n+1)A bits are not used in the generation of the channel (n) and (n+1) output.

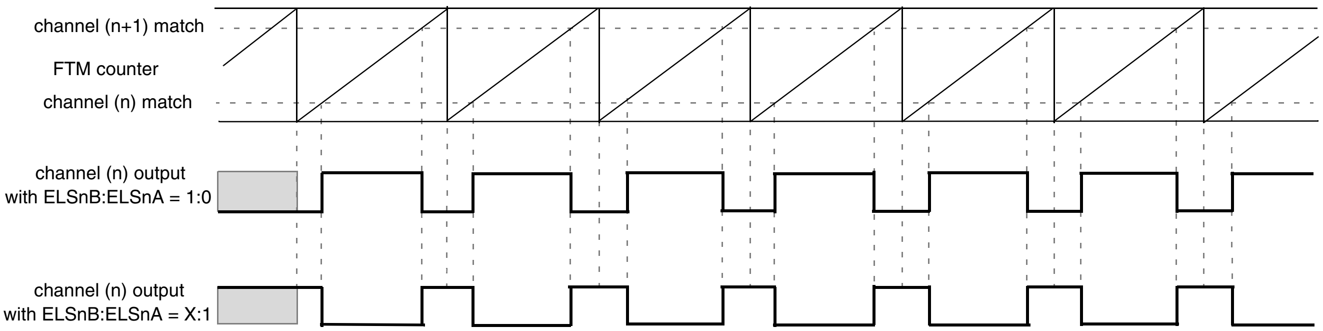


Figure 11-44. Combine mode

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.

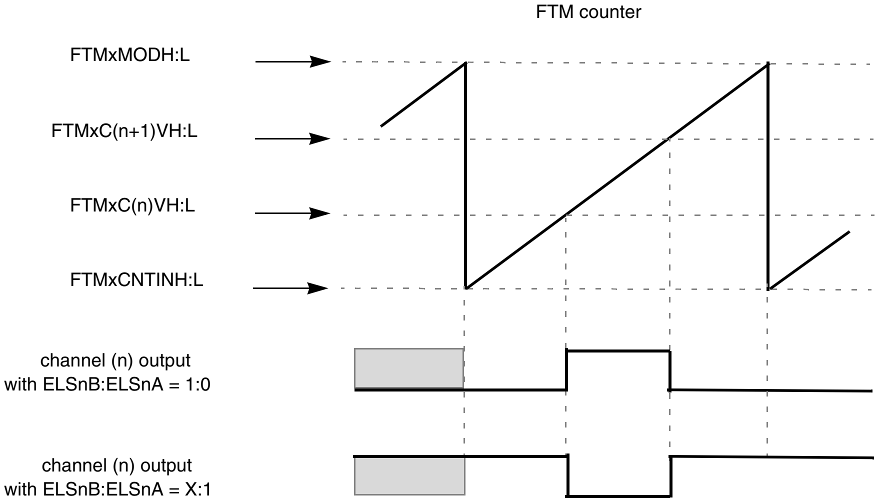


Figure 11-45. Channel (n) output if (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L) and (FTMxC(n)VH:L < FTMxC(n+1)VH:L)

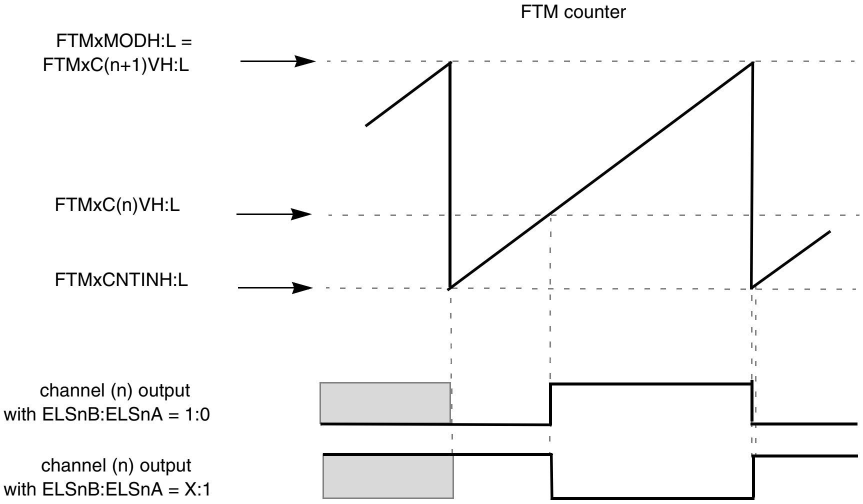


Figure 11-46. Channel (n) output if (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L) and (FTMxC(n+1)VH:L = FTMxMODH:L)

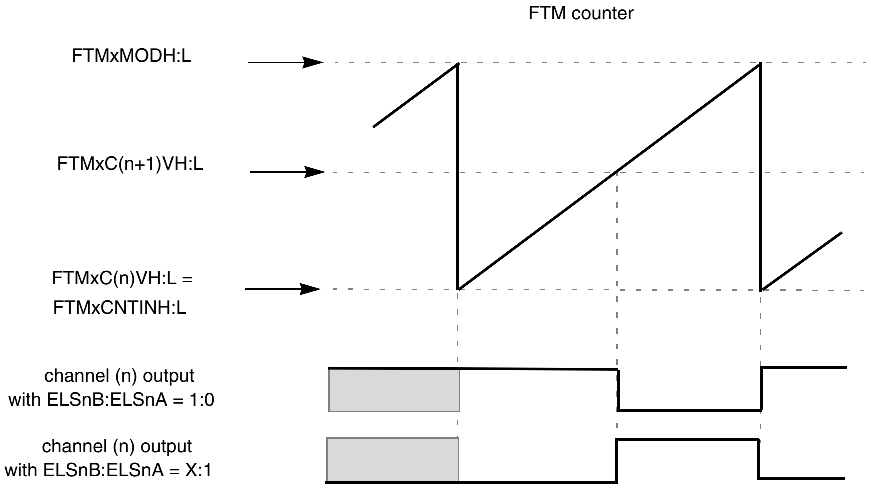


Figure 11-47. Channel (n) output if (FTMxC(n)VH:L = FTMxCNTINH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)

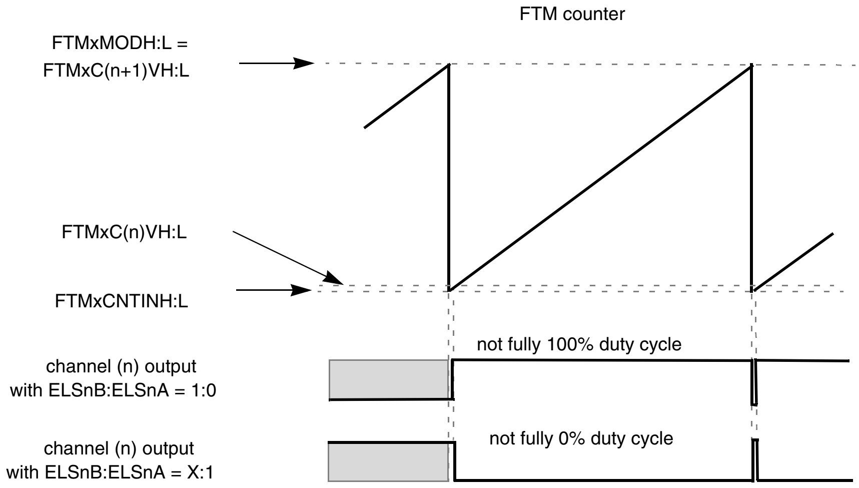


Figure 11-48. Channel (n) output if (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L) and (FTMxC(n)VH:L is almost equal to FTMxCNTINH:L) and (FTMxC(n+1)VH:L = FTMxMODH:L)

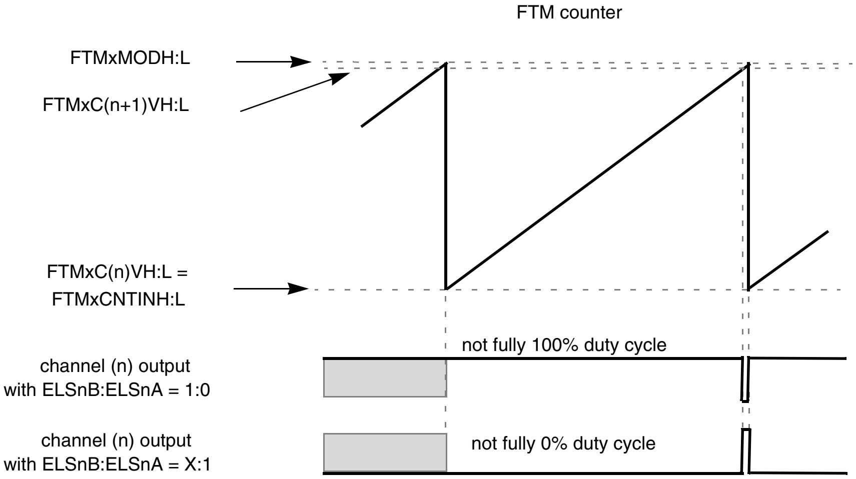


Figure 11-49. Channel (n) output if (FTMxC(n)VH:L = FTMxCNTINH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L) and (FTMxC(n+1)VH:L is almost equal to FTMxMODH:L)

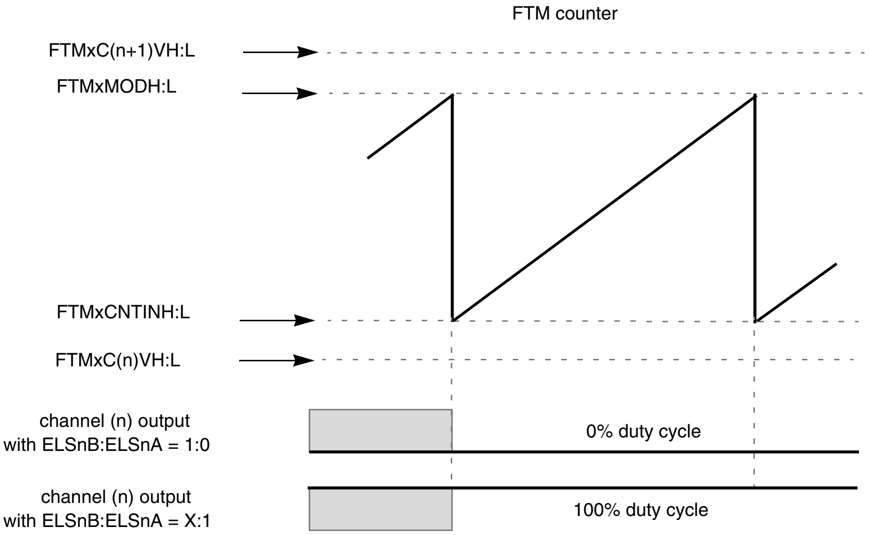


Figure 11-50. Channel (n) output if FTMxC(n)VH:L and FTMxC(n+1)VH:L are not between FTMxCNTINH:L and FTMxMODH:L

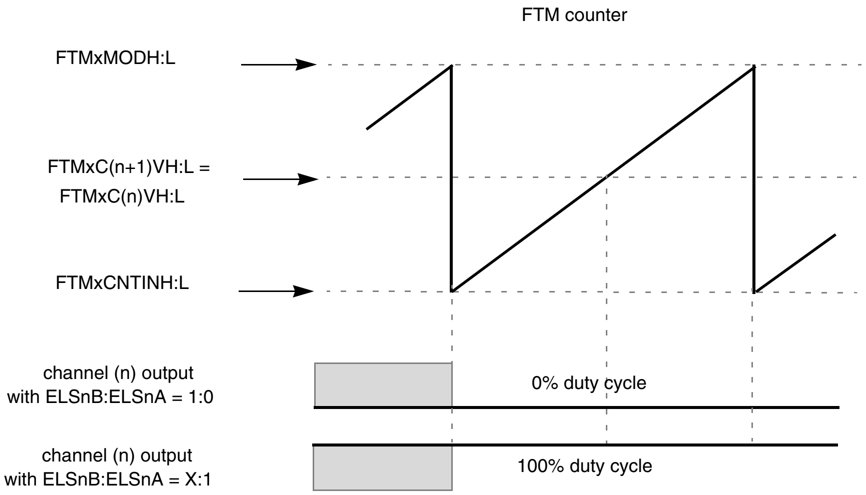


Figure 11-51. Channel (n) output if (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L) and (FTMxCnVH:L = FTMxC(n+1)VH:L)

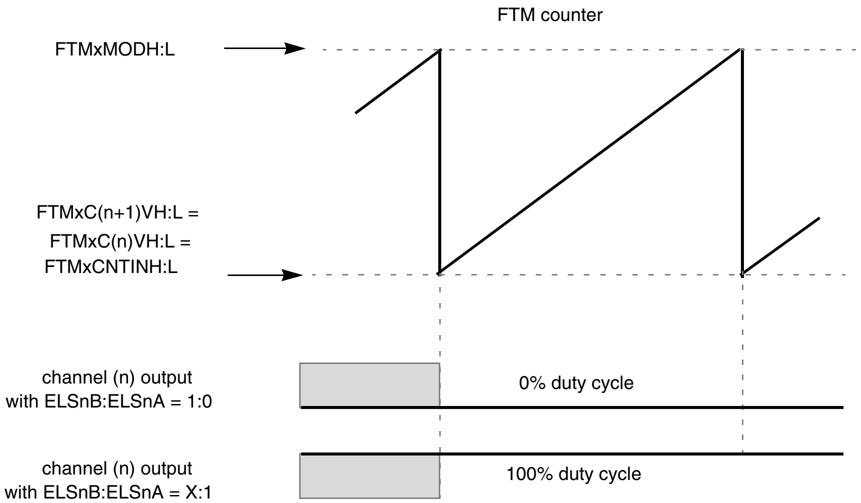


Figure 11-52. Channel (n) output if (FTMxC(n)VH:L = FTMxC(n+1)VH:L = FTMxCNTINH:L)

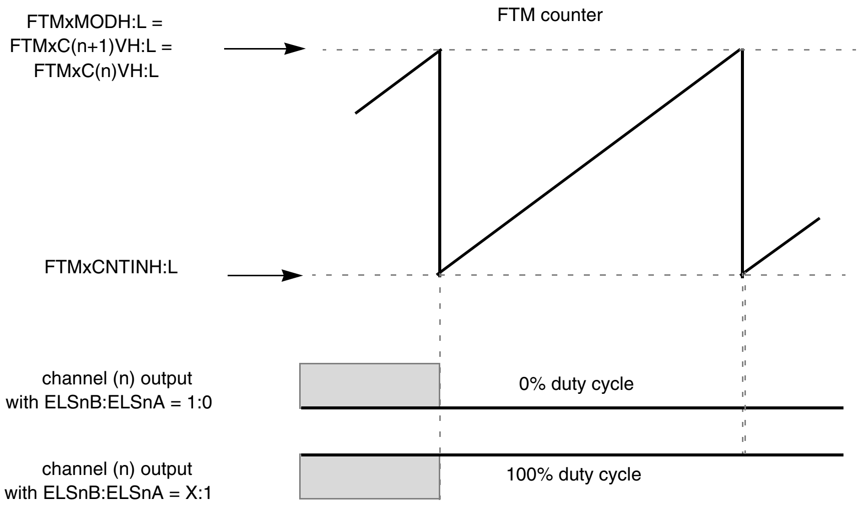


Figure 11-53. Channel (n) output if (FTMxC(n)VH:L = FTMxC(n+1)VH:L = FTMxMODH:L)

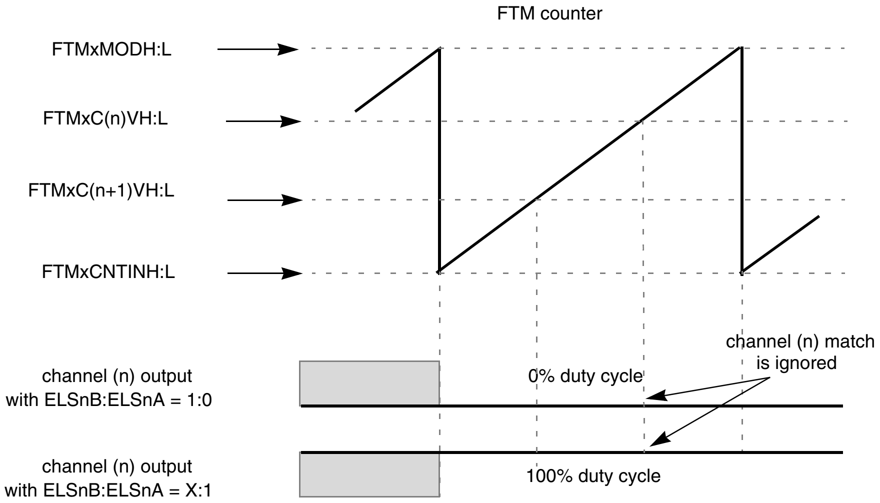


Figure 11-54. Channel (n) output if (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L) and (FTMxC(n)VH:L > FTMxC(n+1)VH:L)

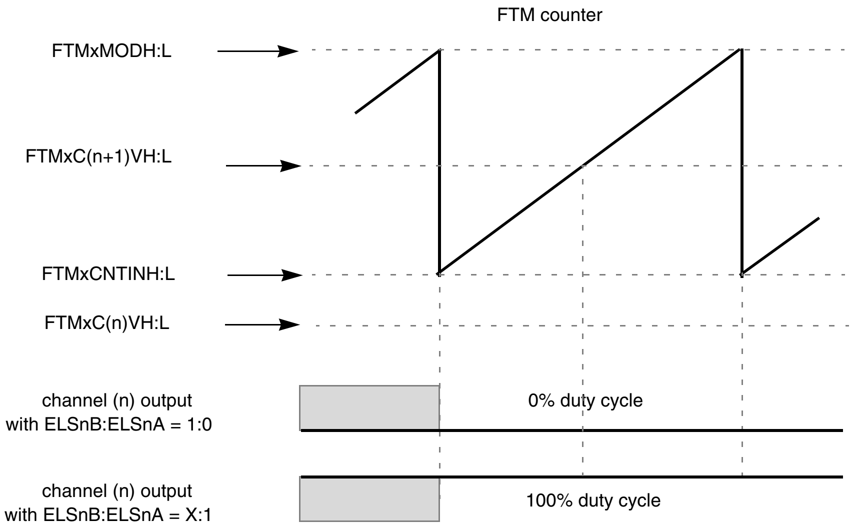


Figure 11-55. Channel (n) output if (FTMxC(n)VH:L < FTMxCNTINH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)

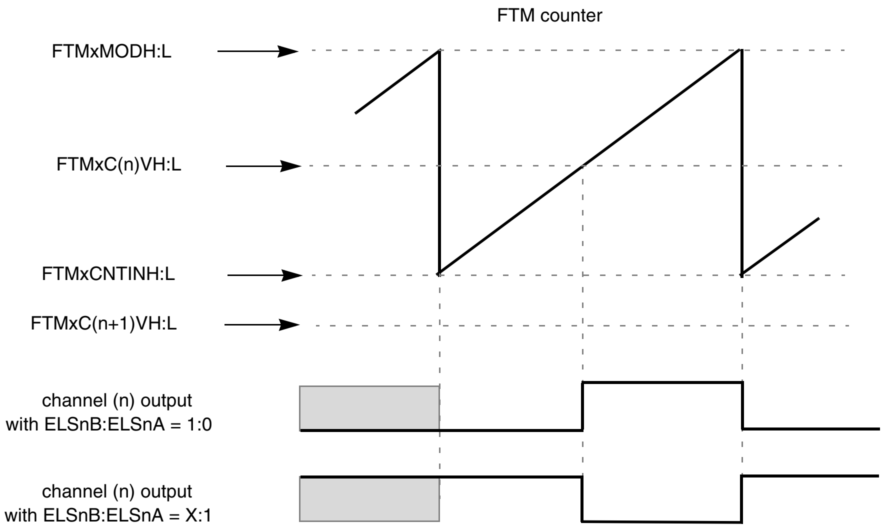


Figure 11-56. Channel (n) output if (FTMxC(n+1)VH:L < FTMxCNTINH:L) and (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)

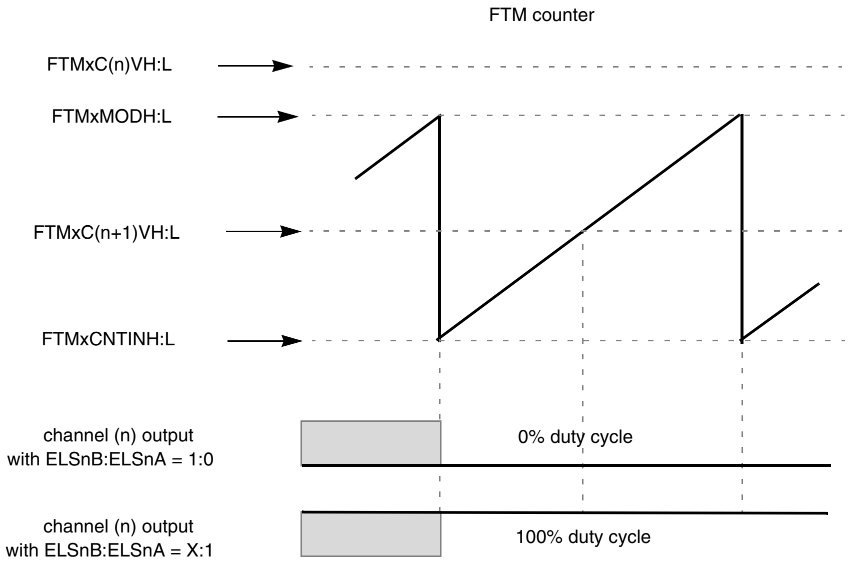


Figure 11-57. Channel (n) output if (FTMxC(n)VH:L > FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n+1)VH:L < FTMxMODH:L)

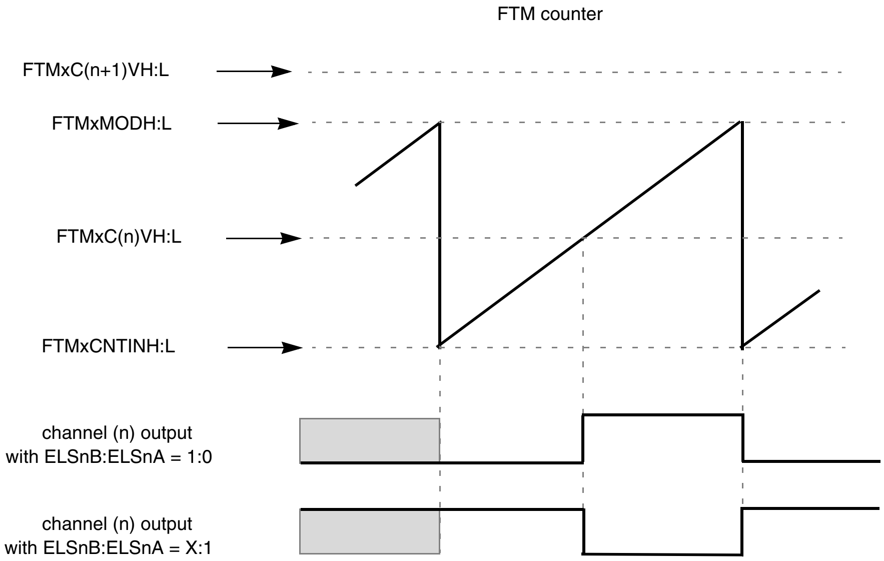


Figure 11-58. Channel (n) output if (FTMxC(n+1)VH:L > FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n)VH:L < FTMxMODH:L)

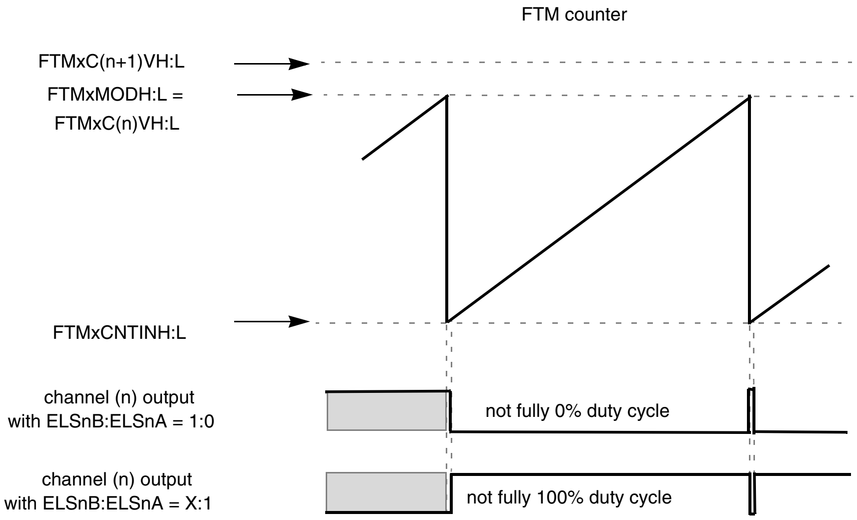


Figure 11-59. Channel (n) output if (FTMxC(n+1)VH:L > FTMxMODH:L) and (FTMxCNTINH:L < FTMxC(n)VH:L = FTMxMODH:L)

11.4.8.1 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.

11.4.9 Complementary mode

The complementary mode is selected when:

- (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0) and (COMP = 1)

In complementary mode the channel (n+1) output is the inverse of the channel (n) output.

If (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0) and (COMP = 0), then the channel (n+1) output is the same as the channel (n) output.

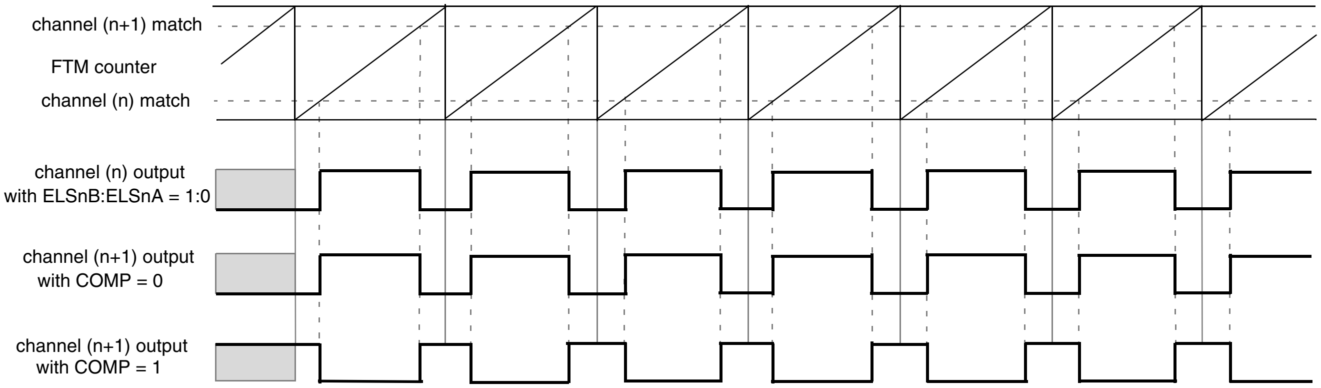


Figure 11-60. Channel (n+1) output in complementary mode with (ELSnB:ELSnA = 1:0)

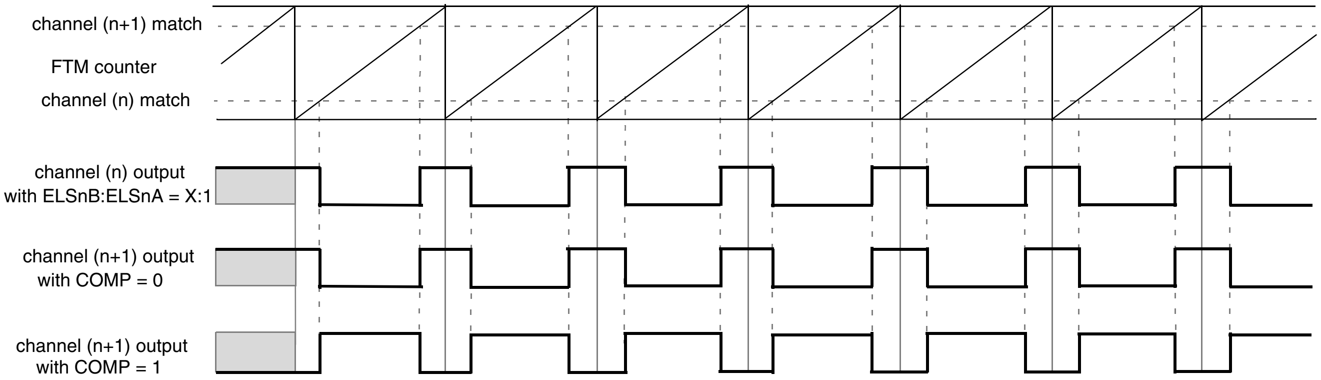


Figure 11-61. Channel (n+1) output in complementary mode with (ELSnB:ELSnA = X:1)

NOTE

Complementary mode is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Complementary mode with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.10 Load of the registers with write buffers

FTMxCNTINH:L registers are always updated with their write buffer after both bytes have been written.

If (CLKS[1:0] = 00) then FTMxMODH:L registers are updated when their second byte is written (independent of FTMEN bit). If (CLKS[1:0] not = 00) then FTMxMODH:L registers are updated with their write buffer according to the FTMEN bit. If (CLKS[1:0] not = 00 and FTMEN = 1) then FTMxMODH:L registers are updated by PWM synchronization (Section 11.4.11, “PWM synchronization). If (CLKS[1:0] not = 00 and FTMEN = 0) then FTMxMODH:L registers are updated according to the CPWMS bit, that is:

- If (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 (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] = 00) then FTMxCnVH:L registers are updated when their second byte is written (independent of FTMEN bit). If (CLKS[1:0] not = 00) then FTMxCnVH:L registers are updated with their write buffer according to the FTMEN bit. If (CLKS[1:0] not = 00 and FTMEN = 1) then FTMxCnVH:L registers are updated by PWM synchronization (Section 11.4.11, “PWM synchronization). If (CLKS[1:0] not = 00 and FTMEN = 0) then FTMxCnVH:L registers are updated according to the selected mode, that is:

- If (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 (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 (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).

11.4.11 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 synchronizaton with (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

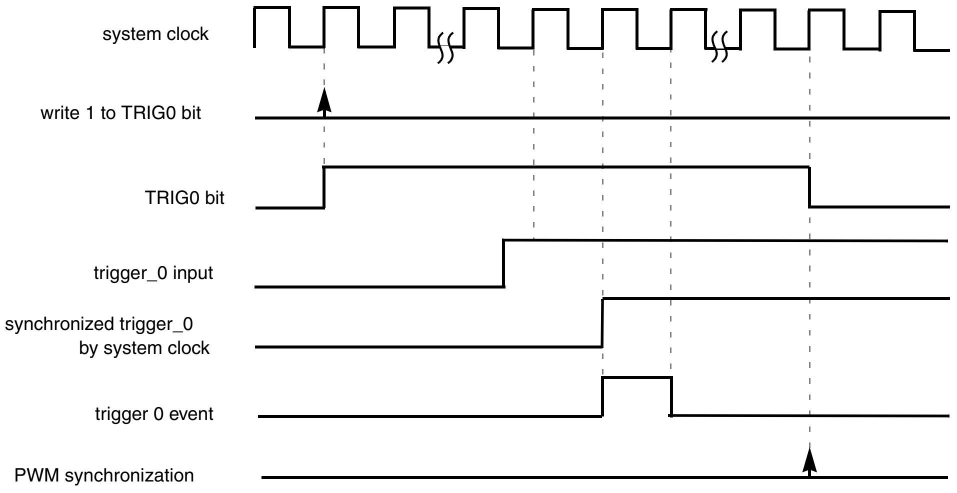
11.4.11.1 Hardware Trigger

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 PWM synchronization (initiated by a selected hardware trigger event) is completed.

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 11-62. Hardware trigger event

11.4.11.2 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 complete.

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 REINIT = 0 and there is a software trigger event, then the load of FTMxMODH:L and FTMxCnVH:L reigsters is only made at the boundary cycle (CNTMIN and CNTMAX). In this case, the SWSYNC bit is cleared only at the boundary cycle, so the user does not know when this bit will be cleared. Thus, 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.

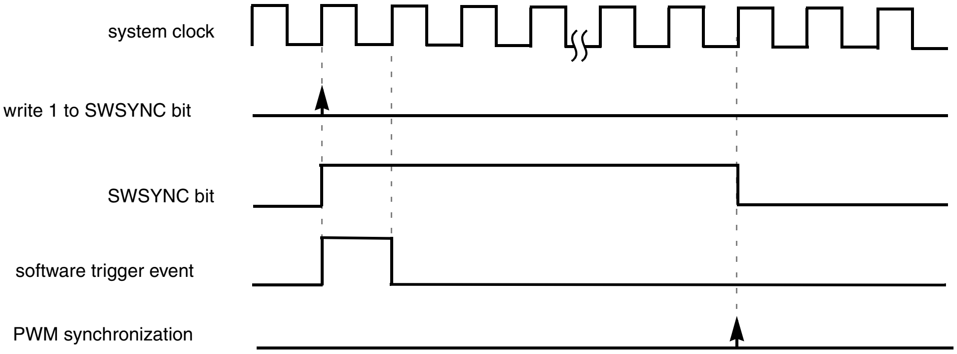


Figure 11-63. Software trigger event

11.4.11.3 Boundary cycle

The CNTMAX and CNTMIN bits select the boundary cycle when the FTMxMODH:L and FTMxCnVH:L registers will be updated with the value of their write buffer by PWM synchronization, except if (REINIT = 1).

If (CNTMIN = 1) then the boundary cycle is the FTMxCNTINH:L value. Registers FTMxMODH:L and FTMxCnVH:L 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. Registers FTMxMODH:L and FTMxCnVH:L 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 would not be made, except if (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 would be made in the first boundary cycle that occurs with valid conditions for FTMxMODH:L or FTMxCnVH:L synchronization, except if (REINIT = 1).

The CNTMAX and CNTMIN bits are cleared only by software.

NOTE

PWM synchronization boundary cycle is only available when (CNTMAX = 1). PWM synchronizaton with (CNTMIN = 1) is not recommended and its results are not guaranteed.

11.4.11.4 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 and either a hardware or software trigger event in one of the following situations.

If 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 11-64). If the trigger event was a hardware trigger, then the trigger enable bit (TRIGn) is cleared on the next selected boundary cycle (Figure 11-65).

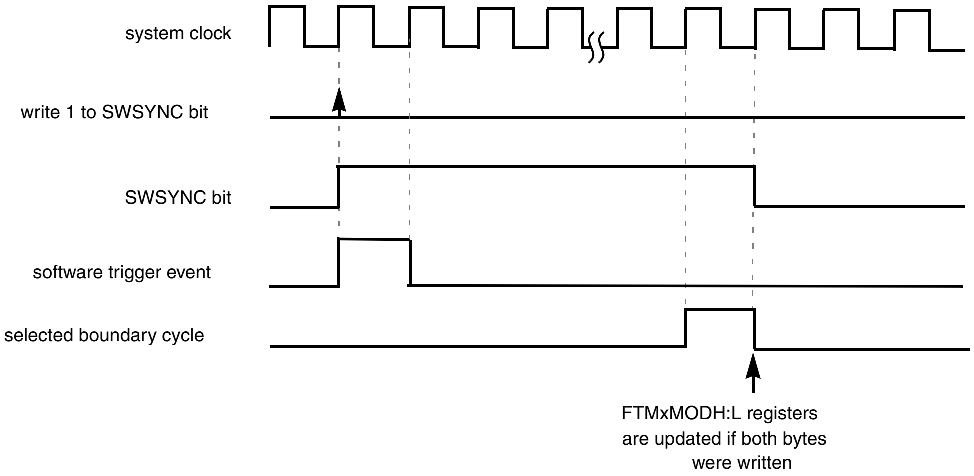


Figure 11-64. FTMxMODH:L synchronization when (REINIT = 0) and (software trigger was used)

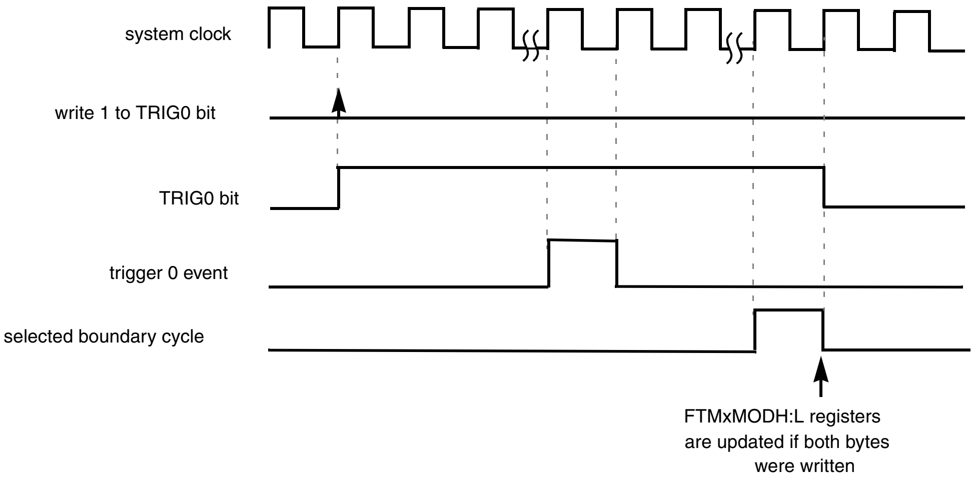


Figure 11-65. FTMxMODH:L synchronization when (REINIT = 0) and (a hardware trigger was used)

If 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 11-66). If the trigger event was a hardware trigger, then the TRIGn bit is cleared (Figure 11-67).

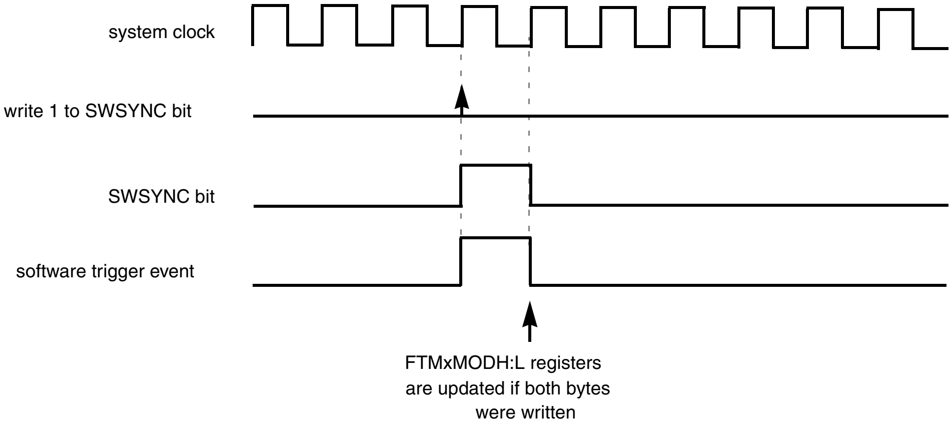


Figure 11-66. FTMxMODH:L synchronization when (REINIT = 1) and (software trigger was used)

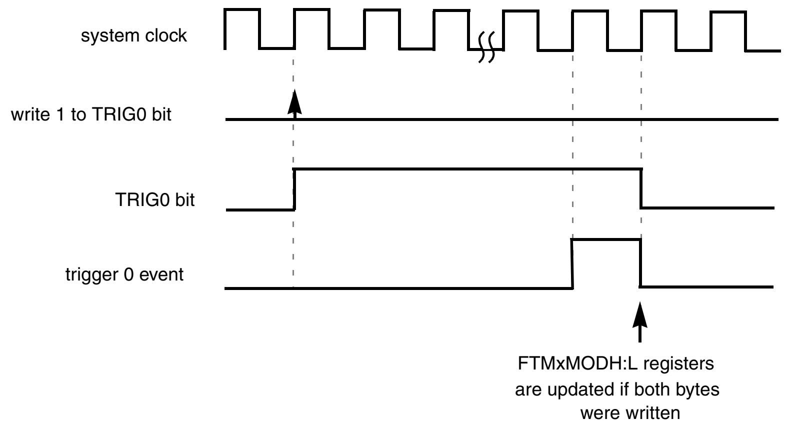


Figure 11-67. FTMxMODH:L synchronization when (REINIT = 1) and (a hardware trigger was used)

11.4.11.5 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 11.4.11.4, "FTMxMODH:FTMxMODL synchronization").

11.4.11.6 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 the SYNCHOM bit.

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.

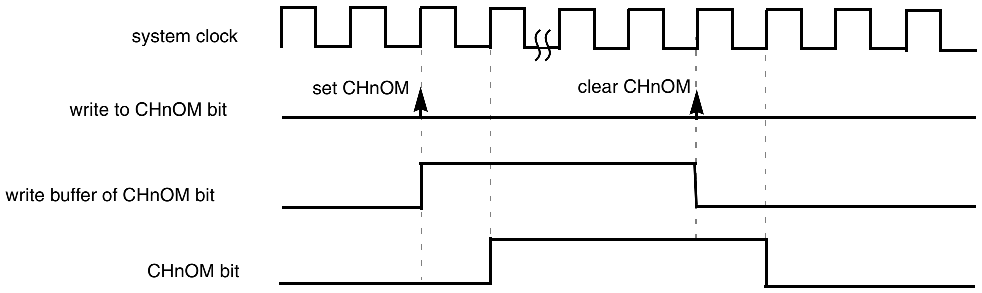


Figure 11-68. CHnOM synchronization when (SYNCHOM = 0)

If SYNCHOM = 1, 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 11-69). If the trigger event was a hardware trigger, then the trigger enable bit (TRIGn) is cleared on the next selected boundary cycle (Figure 11-70).

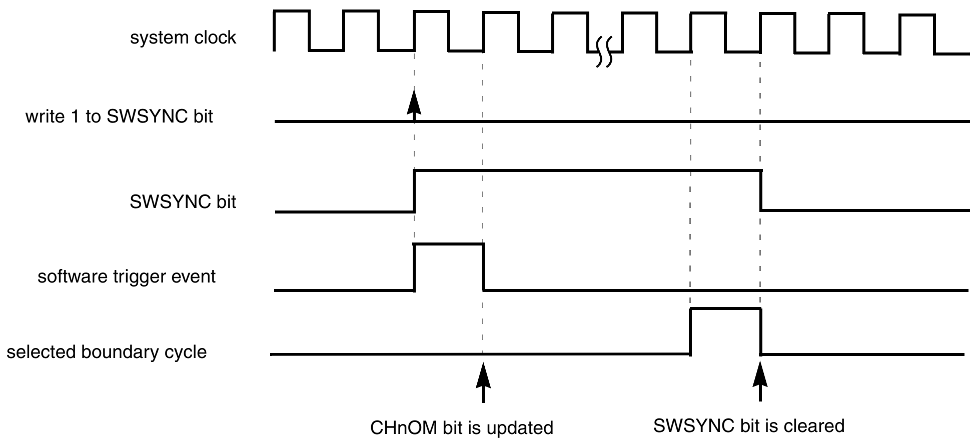


Figure 11-69. CHnOM synchronization when (SYNCHOM = 1) and (software trigger was used)

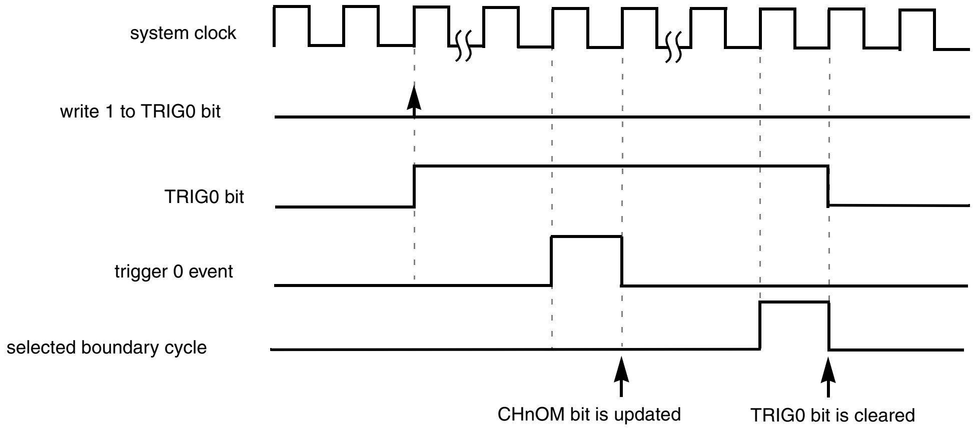


Figure 11-70. CHnOM synchronization when (SYNCHOM = 1) and (a hardware trigger was used)

11.4.11.7 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, 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 11-71). If the trigger event was a hardware trigger, then the TRIGn bit is cleared (Figure 11-72).

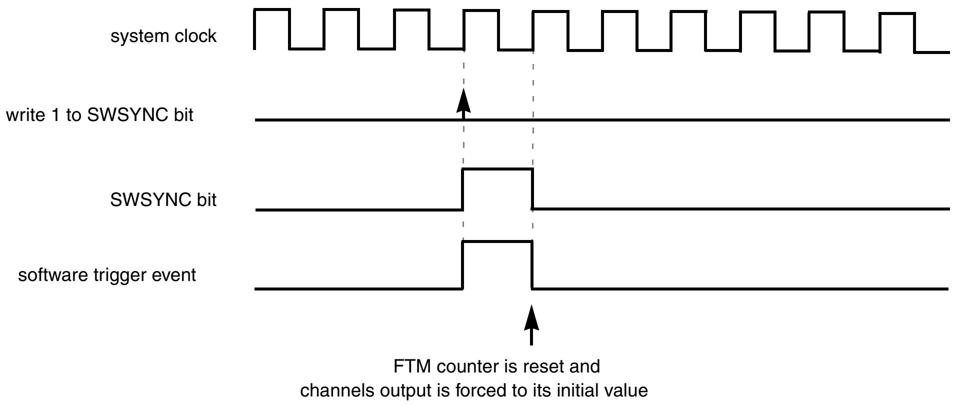


Figure 11-71. FTM counter synchronization when (REINIT = 1) and (software trigger was used)

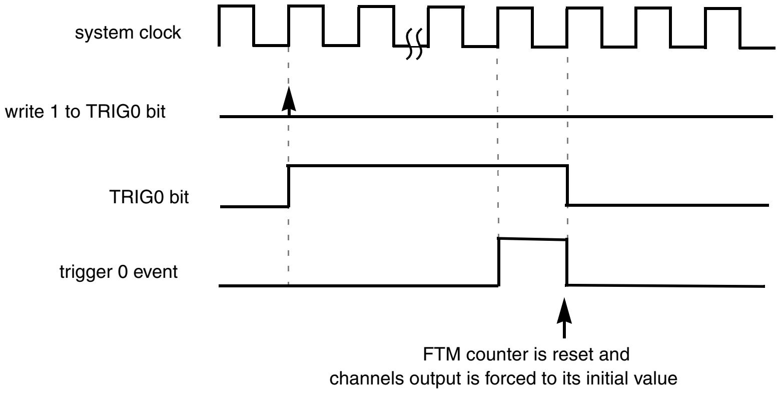


Figure 11-72. FTM counter synchronization when (REINIT = 1) and (a hardware trigger was used)

11.4.11.8 Summary of PWM synchronization

Table 11-23 shows the conditions are required to update the registers FTMxCNTINH:L, FTMxMODH:L, FTMxCnVH:L, FTMxCNTH:L, FTMxOUTMASK and trigger bits (SWSYNC and TRIG[2:0]) with their write buffer by PWM synchronization.

Table 11-23. Update the registers and bits by PWM synchronization

Register or bit	REINIT	SYNCHOM	CNTMAX	CNTMIN	SYNCEN	Description
FTMxCNTINH:L	X	X	X	X	X	Changes take effect after the second byte is written. Effect is seen after the next TOF or selected synchronization trigger occurs if REINIT = 1.
FTMxMODH:L	0	X	1	0	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its maximum value after the selected synchronization trigger has occurred.
	0	X	0	1	X	FTMxMODH:L are updated with their write buffer contents when the counter reaches its minimum value after the selected synchronization trigger has occurred.
	1	X	X	X	X	FTMxMODH:L are updated with their write buffer contents when the selected synchronization trigger occurs.
FTMxCnVH:L	0	X	1	0	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its maximum value after the selected synchronization trigger has occurred.
	0	X	0	1	1	FTMxCnVH:L are updated with their write buffer contents when the counter reaches its minimum value after the selected synchronization trigger has occurred.
	1	X	X	X	1	FTMxCnVH:L are updated with their write buffer contents when the selected synchronization trigger occurs.
FTMxCNTH:L	1	X	X	X	X	FTMxCNTH:L are forced to the FTM counter initial value when the selected synchronization trigger occurs.
FTMxOUTMASK	X	0	X	X	X	Changes to FTMxOUTMASK take effect on the next rising edge of the system clock.
	X	1	X	X	X	FTMxOUTMASK is updated with its write buffer contents when the selected synchronization trigger occurs.
Triger bit (SWSYNC and TRIG[2:0])	0	X	1	0	X	Selected synchronization trigger bit is cleared when the counter reaches its maximum value after the selected synchronization trigger has occurred.
	0	X	0	1	X	Selected synchronization trigger bit is cleared when the counter reaches its minimum value after the selected synchronization trigger has occurred.
	1	X	X	X	X	Selected synchronization trigger bit is cleared when the selected synchronization trigger occurs.

11.4.12 Deadtime insertion

The deadtime insertion is enabled when (DTEN = 1) and (DTVAL[5:0] is non- zero).

FTMxDEADTIME register defines the deadtime delay which can be used for all FTM channels. The DTPS[1:0] bits define the prescaler for the system clock and the DTVAL[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.

If the deadtime delay is greater than or equal to the channel (n) duty cycle (FTMxC(n+1)VH:L - FTMxC(n)VH:L), then the channel (n) output is always 0. Equally, if the deadtime delay is greater than or equal to the channel (n+1) duty cycle (FTMxMODH:L - FTMxCNTINH:L - (FTMxC(n+1)VH:L - FTMxC(n)VH:L)), then the channel (n+1) output is always 0.

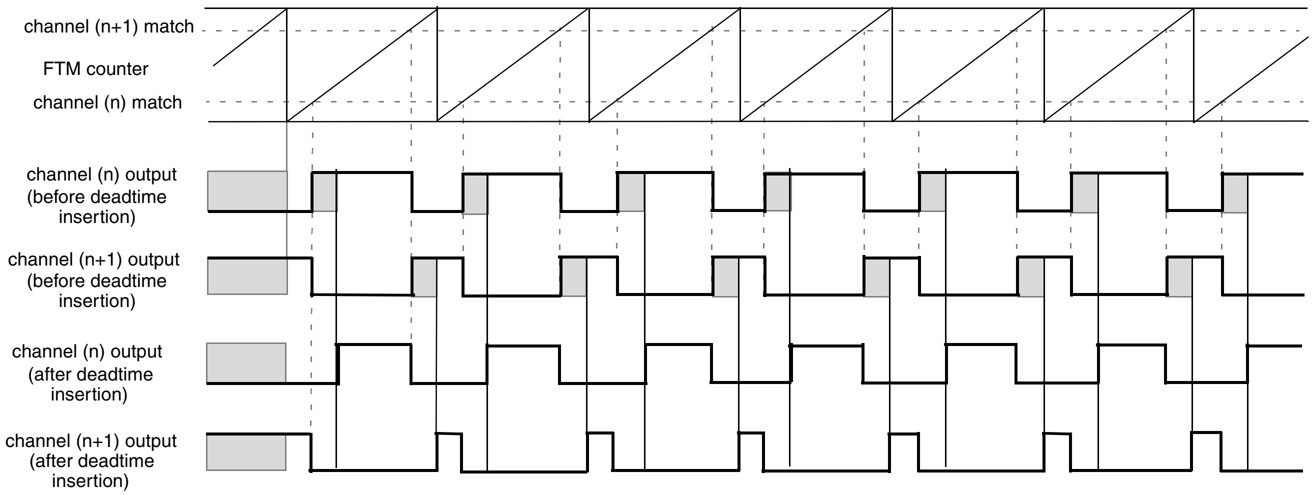

channel (n+1) match
FTM counter
channel (n) match
channel (n) output (before deadtime insertion)
channel (n+1) output (before deadtime insertion)
channel (n) output (after deadtime insertion)
channel (n+1) output (after deadtime insertion)

Figure 11-73. Deadtime insertion with ELSnB:ELSnA = 1:0

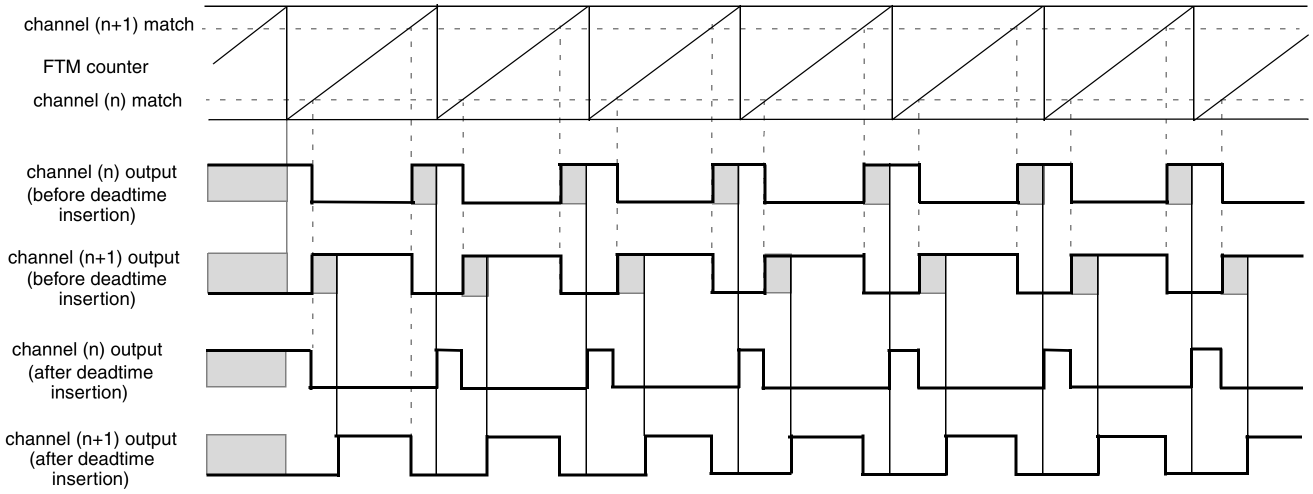

channel (n+1) match
FTM counter
channel (n) match
channel (n) output (before deadtime insertion)
channel (n+1) output (before deadtime insertion)
channel (n) output (after deadtime insertion)
channel (n+1) output (after deadtime insertion)

Figure 11-74. Deadtime insertion with ELSnB:ELSnA = X:1

NOTE

Deadtime insertion is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0) and (COMP = 1). Deadtime insertion with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) or (COMP = 0) is not recommended and its results are not guaranteed.

11.4.13 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 11.4.11.6, “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 not enabled until the beginning of the next PWM cycle (Figure 11-75).

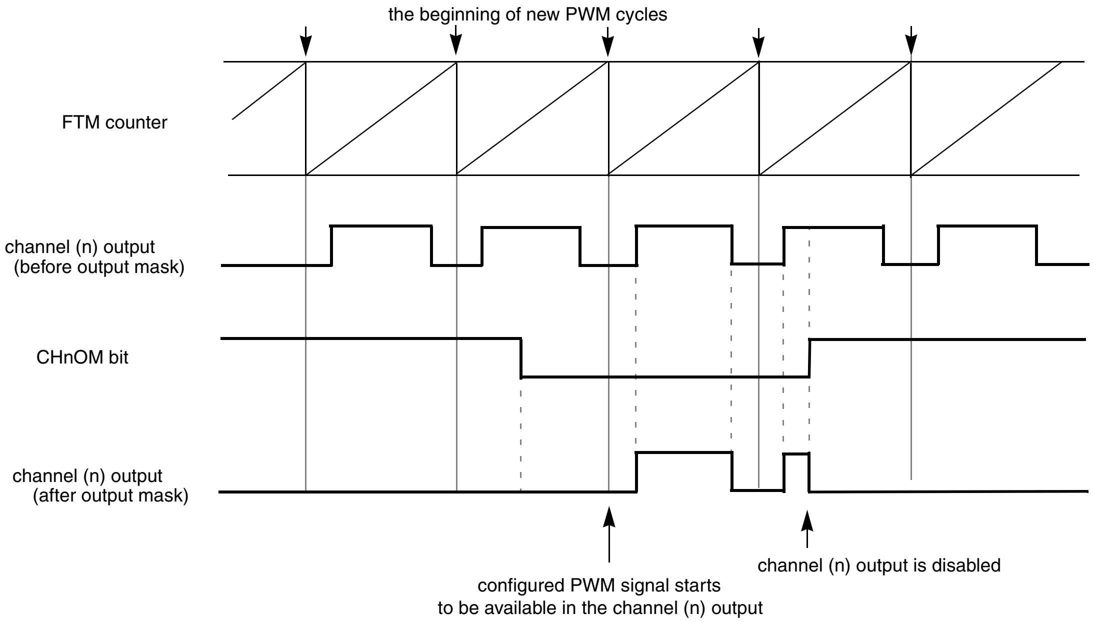

the beginning of new PWM cycles
FTM counter
channel (n) output
(before output mask)
CHnOM bit
channel (n) output
(after output mask)
configured PWM signal starts
to be available in the channel (n) output
channel (n) output is disabled

Figure 11-75. Output mask

The Table 11-24 shows the output mask result before the polarity control.

Table 11-24. Output mask result for channel (n) (before the polarity control)

CHnOM bit	Output mask input	Output mask result
0	inactive state	inactive state
	active state	active state
1	inactive state	inactive state
	active state	

NOTE

Output mask is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Output mask with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.14 Fault control mode

The fault control mode is enabled if (FTMEN = 1) and (FAULTM[1:0] not = 00).

First the fault input signal is synchronized by the system clock (synchronizer block in Figure 11-76). Following synchronization, the fault input signal enters the filter block. When there is a state change in the fault input signal, the 5-bit counter is reset and starts counting up. As long as the new state is stable on the fault input, 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 value is validated. It is then transmitted as a pulse edge to the edge detector.

If the opposite edge appears on the fault input signal before validation (counter overflow), the counter is reset. At the next input transition, the counter will start 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 filter function is disabled when the FFVAL[3:0] bits are zero. In this case the fault input signal is delayed 4 rising edges of the system clock and the CHnF bit is set on 5th rising edge of the system clock after a rising edge occurs on the fault input pin.

If (FFVAL[3:0] not = 0000) then the input signal is delayed (4 + FFVAL[3:0]) rising edges of the system clock, that is, the CHnF bit is set (4 + FFVAL[3:0]) rising edges of the system clock after a rising edge occurs on the fault input pin.

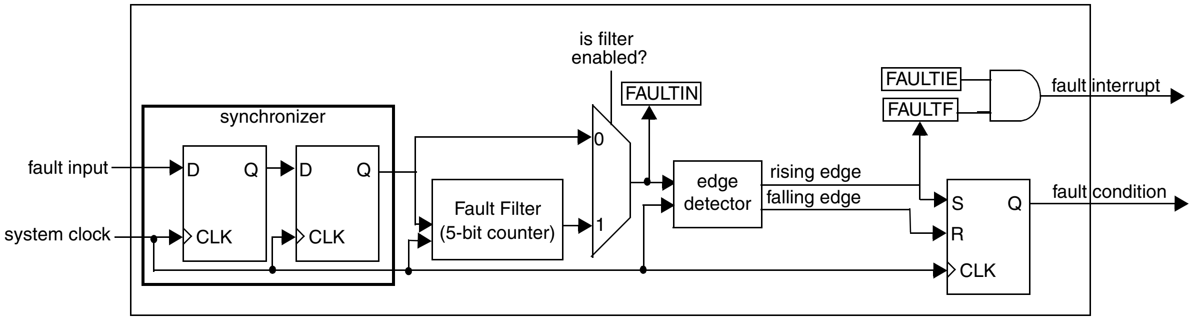


Figure 11-76. Fault Control Block Diagram

If the fault control is enabled and a rising edge at the fault input signal (fault condition) is detected, then the FAULTF bit is set.

If the fault control is enabled (FAULTM[1:0] non zero), a fault condition has occurred 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). 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 mode is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Fault control with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.14.1 Automatic Fault Clearing

If the automatic fault clearing is selected (FAULTM[1:0] = 11), then the disabled channels outputs are enabled when the fault input signal (FAULTIN) returns to zero (at the filter output if the filter is enabled, or at the synchronizer output if the filter is disabled) and a new PWM cycle begins (Figure 11-77).

When automatic fault clearing is selected, clearing the FAULTF bit will not affect disabled channels.

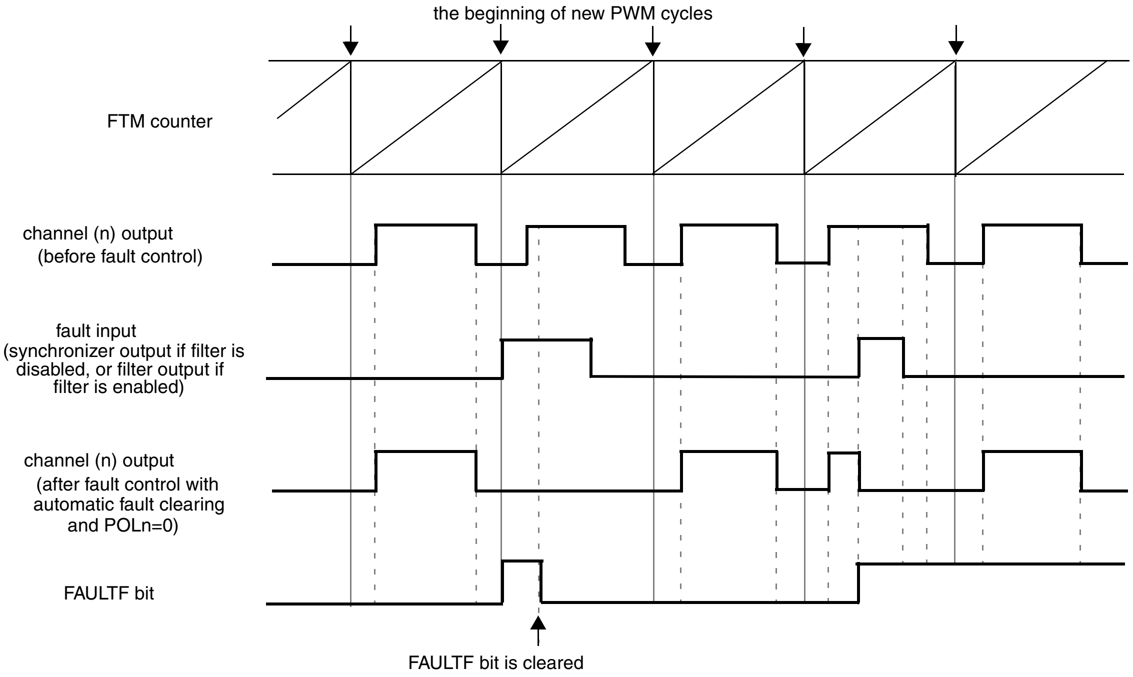

the beginning of new PWM cycles
FTM counter
channel (n) output
(before fault control)
fault input
(synchronizer output if filter is disabled, or filter output if filter is enabled)
channel (n) output
(after fault control with automatic fault clearing and POLn=0)
FAULTF bit
FAULTF bit is cleared

Figure 11-77. Fault control with automatic fault clearing

11.4.14.2 Manual Fault Clearing

If the manual fault clearing is selected (FAULTM[1:0] = 01 or 10), then disabled channels outputs are enabled when the FAULTF bit is cleared and a new PWM cycle begins (Figure 11-78).

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.

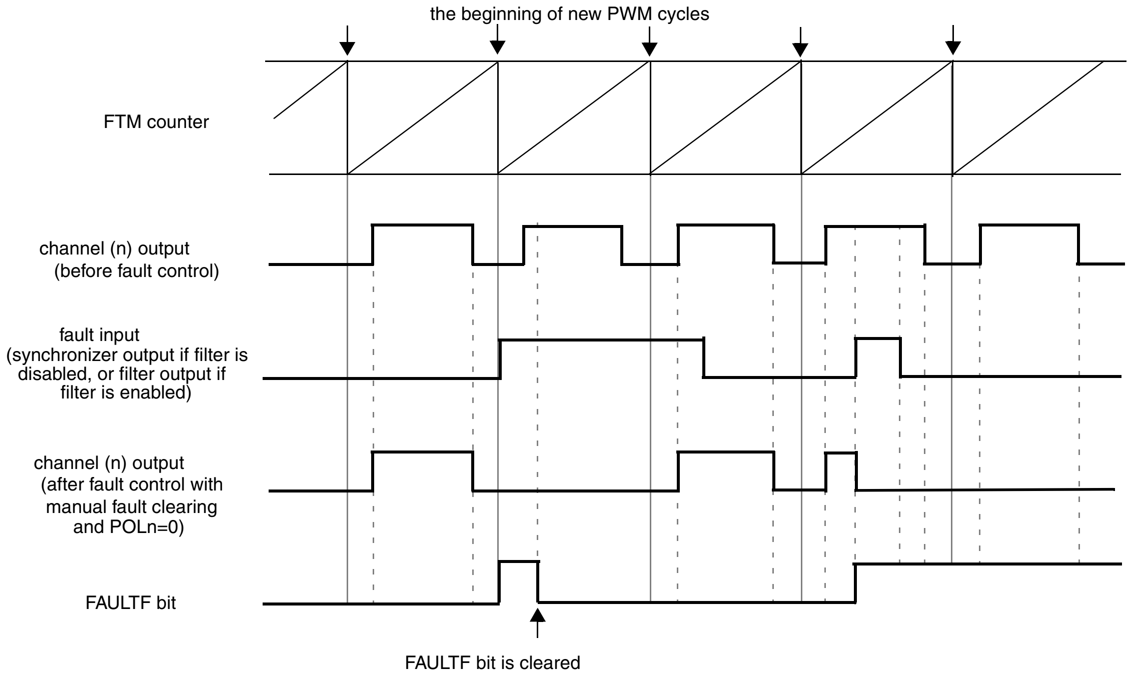


Figure 11-78. Fault control with manual fault clearing

11.4.15 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) and (COMBINE = 1) and (CPWMS = 0). Polarity control with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.16 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 11-79 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 intialization only if the FTM counter is disabled (that is, CLKS[1:0] = 00), otherwise unpredictable behavior is likely.

NOTE

Initialization is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Initialization with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.17 Features Priority

Figure 11-79 shows the priority of the features that can be combined to generate channel (n) and (n+1) outputs.

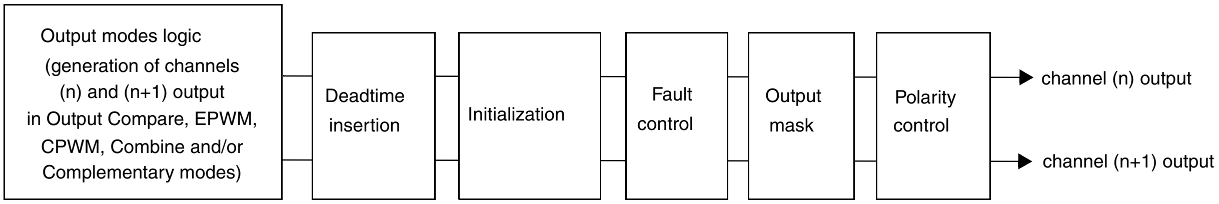


Figure 11-79. FTM features priority

11.4.18 Match Trigger

The match triggers are generated if (FTMEN = 1) and (one or more channels were selected by the CHjTRIG bit where j = 2, 3, 4, 5, 6 or 7). The CHjTRIG bit defines if the channel (j) match (that is, FTM counter = FTMxC(j)VH:L) generates the match trigger.

The match trigger is an output signal to provide a trigger for other modules.

One or more FTM channels can be selected to generate multiple match triggers in one PWM period (as shown in Figure 11-80).

The width of a match trigger is a system clock period.

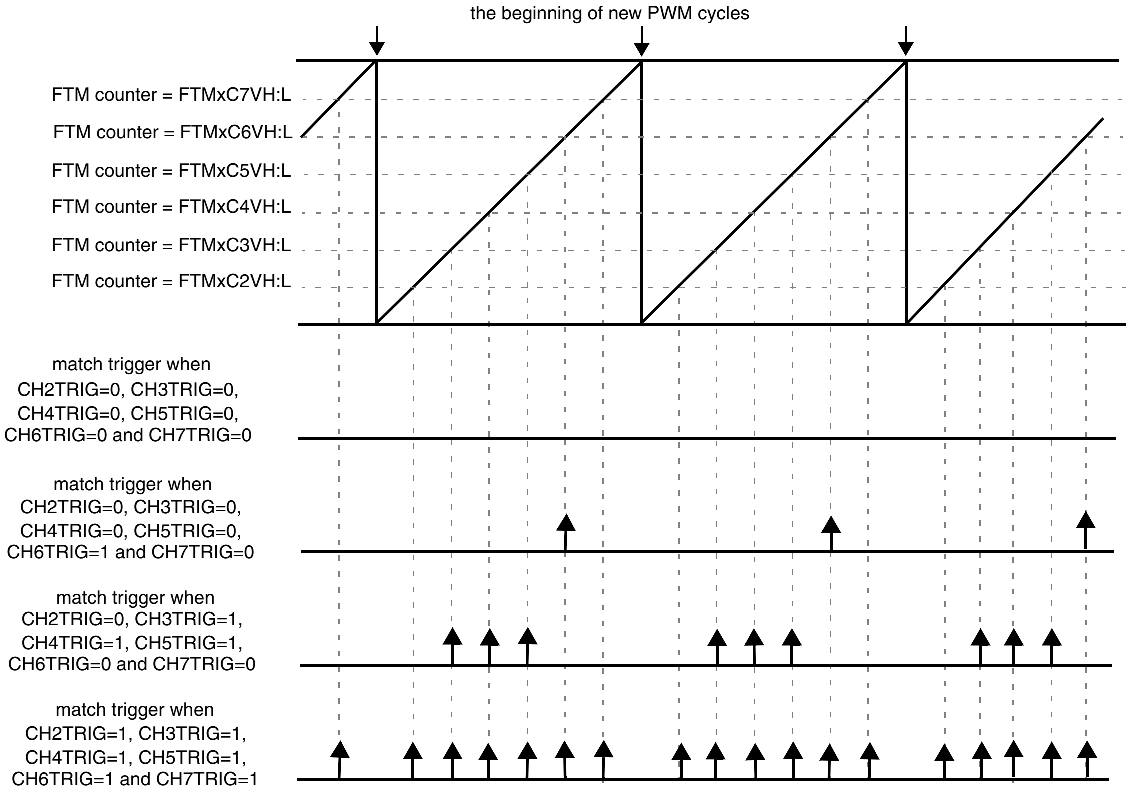

the beginning of new PWM cycles
FTM counter = FTMxC7VH:L
FTM counter = FTMxC6VH:L
FTM counter = FTMxC5VH:L
FTM counter = FTMxC4VH:L
FTM counter = FTMxC3VH:L
FTM counter = FTMxC2VH:L
match trigger when CH2TRIG=0, CH3TRIG=0, CH4TRIG=0, CH5TRIG=0, CH6TRIG=0 and CH7TRIG=0
match trigger when CH2TRIG=0, CH3TRIG=0, CH4TRIG=0, CH5TRIG=0, CH6TRIG=1 and CH7TRIG=0
match trigger when CH2TRIG=0, CH3TRIG=1, CH4TRIG=1, CH5TRIG=1, CH6TRIG=0 and CH7TRIG=0
match trigger when CH2TRIG=1, CH3TRIG=1, CH4TRIG=1, CH5TRIG=1, CH6TRIG=1 and CH7TRIG=1

Figure 11-80. Match triggers

NOTE

Match trigger is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Match trigger with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.19 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 11-81)
- when there is a write to FTMxCNTH or FTMxCNTL register (Figure 11-82)
- when there is the synchronization trigger event and (REINIT = 1) (Figure 11-83)
- if (FTMxCNTH:L = FTMxCNTINH:L) and (CLKS[1:0] = 00) and a value different from zero is written to CLKS[1:0] bits (Figure 11-84)

The initialization trigger is an output signal to provide a trigger for other modules.

The width of an initialization trigger is a system clock period.

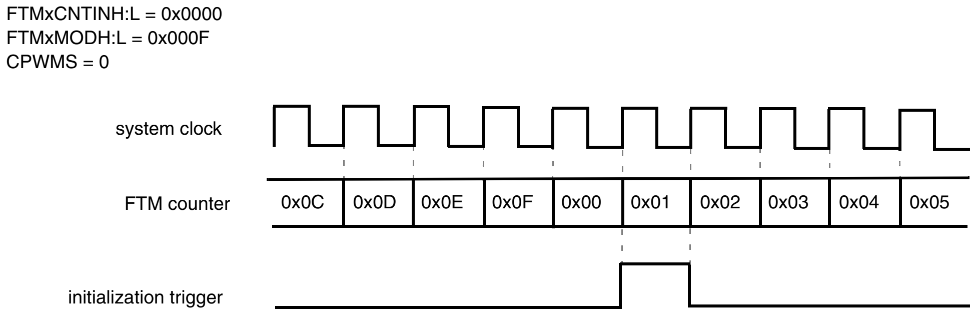


Figure 11-81. Initialization trigger generated when the FTM counting achieved the value of FTMxCNTINH:L

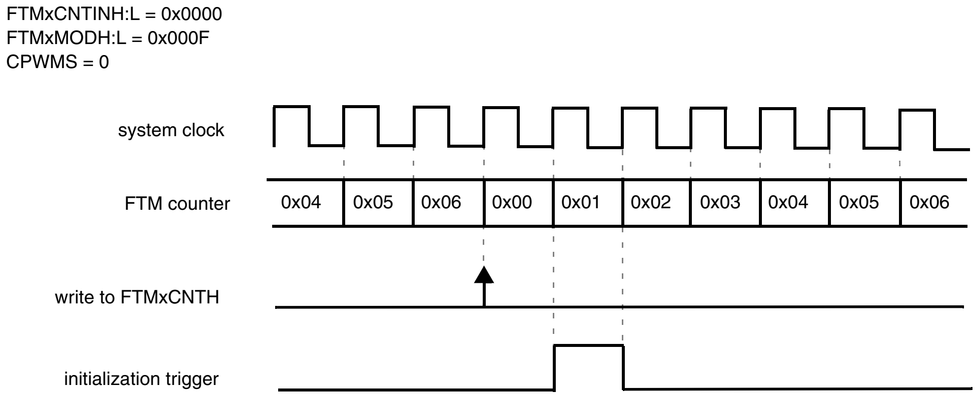


Figure 11-82. Initialization trigger generated when there is a write to FTMxCNTH (or FTMxCNTL)

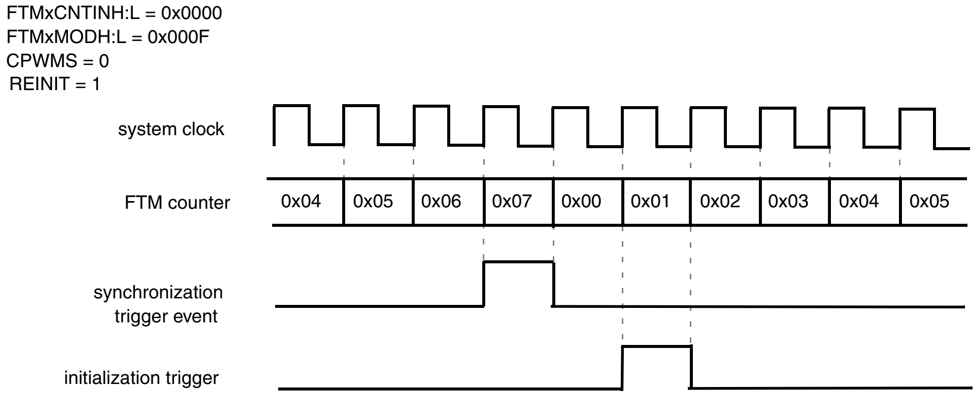


Figure 11-83. Initialization trigger generated when there is the synchronization trigger event and (REINIT = 1)

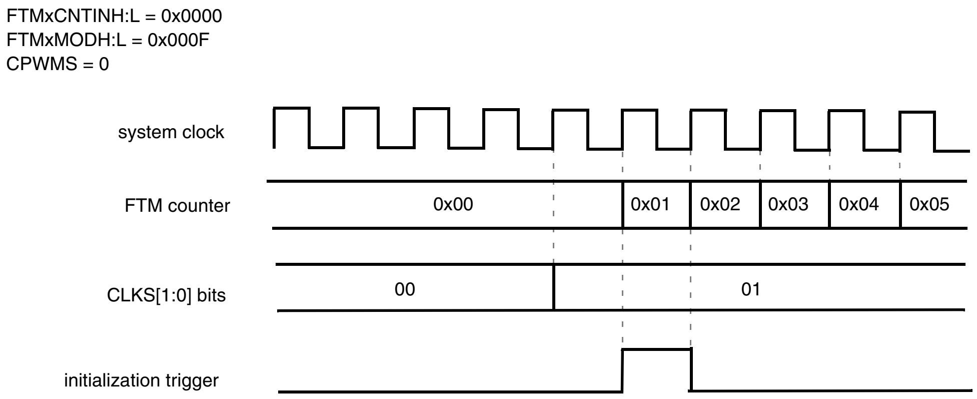


Figure 11-84. Initialization trigger generated if (FTMxCNTH:L = FTMxCNTINH:L) and (CLKS[1:0] = 00) and a value different from zero is written to CLKS[1:0] bits

NOTE

Initialization trigger is only available when (FTMEN = 1) and (COMBINE = 1) and (CPWMS = 0). Initialization trigger with (FTMEN = 0) or (COMBINE = 0) or (CPWMS = 1) is not recommended and its results are not guaranteed.

11.4.20 TPM Emulation

This section describe the FTM features that are selected according to the FTMEN bit.

11.4.20.1 FTM counter clock

If (CLKS[1:0] = 01), then FTMEN bit selects if the system clock is divided by 1 or divided by 2 in the FTM (Section 11.3.3, “FTM Status and Control Register (FTMxSC)).

If (CLKS[1:0] = 01) and (FTMEN = 0), then the clock of FTM counter is the system clock divided by 2.

If (CLKS[1:0] = 01) and (FTMEN = 1), then the clock of FTM counter is the system clock.

11.4.20.2 FTMxMODH:L and FTMxCnVH:L synchronization

If (FTMEN = 0), then the FTMxMODH:L and FTMxCnVH:L registers are updated according to the Section 11.4.10, “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 11.4.11, “PWM synchronization).

11.4.20.3 Free-running counter

If (FTMEN = 0), then the FTM counter is a free-running counter when (FTMxMODH:L = 0x0000) or (FTMxMODH:L = 0xFFFF) (Section 11.4.3.3, “Free running counter).

If (FTMEN = 1), then the FTM counter is a free-running counter when (CPWMS = 0) and (FTMxCNTINH:L = 0x0000) and (FTMxMODH:L = 0xFFFF).

11.4.20.4 Write to FTMxSC

If (FTMEN = 0), then a write to the FTMxSC register resets the write coherency mechanism of FTMxMODH:L registers (Section 11.3.5, “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.

11.4.20.5 Write to FTMxCnSC

If (FTMEN = 0), then a write to the FTMxCnSC register resets the write coherency mechanism of FTMxCnVH:L registers (Section 11.3.7, “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.

11.5 Reset Overview

The FTM is reset whenever any MCU reset occurs

11.6 FTM Interrupts

11.6.1 Timer Overflow Interrupt

The timer overflow interrupt is generated when (TOIE = 1) and (TOF = 1).

11.6.2 Channel (n) Interrupt

The channel (n) interrupt is generated when (CHnIE = 1) and (CHnF = 1).

11.6.3 Fault Interrupt

The fault interrupt is generated when (FAULTIE = 1) and (FAULTF = 1).

Chapter 12
Inter-Integrated Circuit (IICV2)

12.1 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 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}.

12.1.1 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
- Acknowledge bit generation/detection
- Bus busy detection
- General call recognition
- 10-bit address extension

12.1.2 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 while the MCU is in wait mode and can provide a wake-up interrupt.
- Stop mode — The IIC is inactive in stop3 mode for reduced power consumption. The stop instruction does not affect IIC register states. Stop2 resets the register contents.

12.1.3 Block Diagram

Figure 12-1 is a block diagram of the IIC.

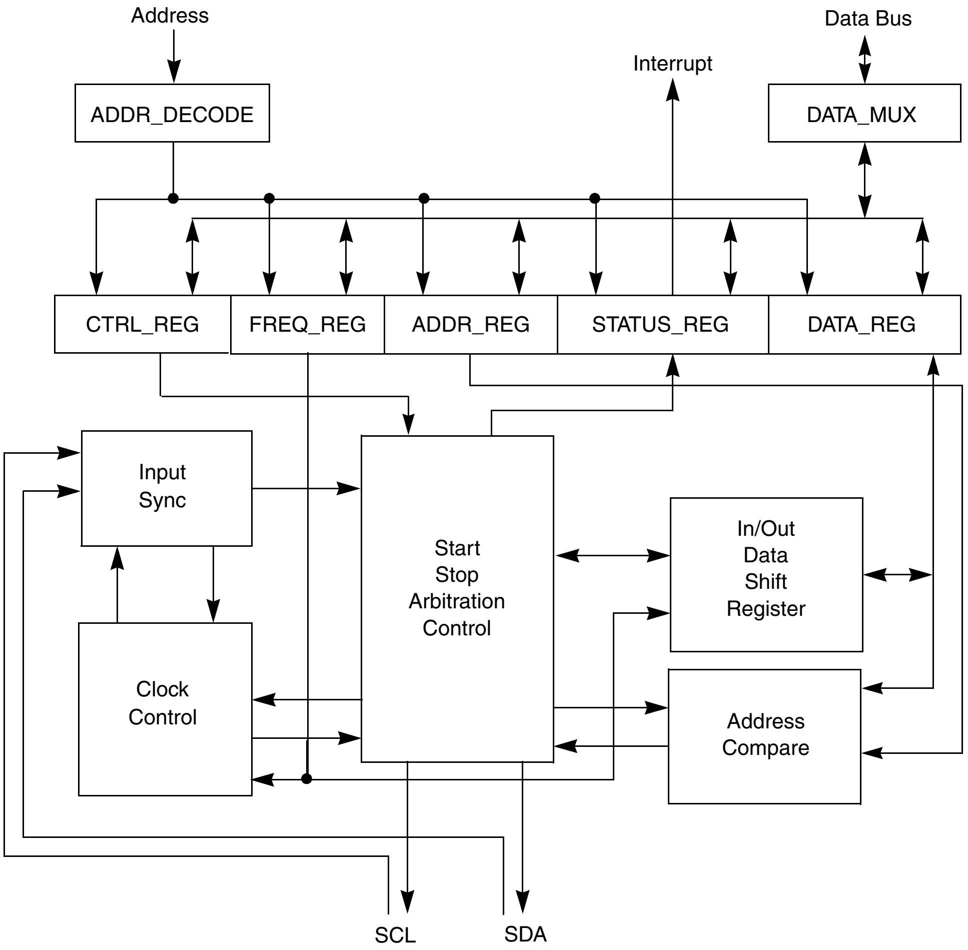


Figure 12-1. IIC Functional Block Diagram

12.2 External Signal Description

This section describes each user-accessible pin signal.

12.2.1 SCL — Serial Clock Line

The bidirectional SCL is the serial clock line of the IIC system.

12.2.2 SDA — Serial Data Line

The bidirectional SDA is the serial data line of the IIC system.

12.3 Register Definition

This section consists of the IIC register descriptions in address order.

Refer to the direct-page register summary in the memory chapter of this document 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.

12.3.1 IIC Address Register (IICA)

	7	6	5	4	3	2	1	0
R	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
W								▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 12-2. IIC Address Register (IICA)

Table 12-1. IICA Field Descriptions

Field	Description
7–1 AD[7:1]	Slave Address. 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.

12.3.2 IIC Frequency Divider Register (IICF)

	7	6	5	4	3	2	1	0
R	MULT		ICR					
W								
Reset	0	0	0	0	0	0	0	0

Figure 12-3. IIC Frequency Divider Register (IICF)

Table 12-2. IICF Field Descriptions

Field	Description
7–6 MULT	IIC Multiplier Factor. The MULT bits define the multiplier factor, mul. This factor, along with the SCL divider, generates 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	IIC Clock Rate. The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits determine the IIC baud rate, the SDA hold time, the SCL Start hold time, and the SCL Stop hold time. Table 12-4 provides the SCL divider and hold values for corresponding values of the ICR. The SCL divider multiplied by multiplier factor mul generates IIC baud rate. $\text{IIC baud rate} = \frac{\text{bus speed (Hz)}}{\text{mul} \times \text{SCLdivider}}$ ***Eqn. 12-1*** SDA hold time is the delay from the falling edge of SCL (IIC clock) to the changing of SDA (IIC data). **SDA hold time = bus period (s) × mul × SDA hold value** ***Eqn. 12-2*** 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). **SCL Start hold time = bus period (s) × mul × SCL Start hold value** ***Eqn. 12-3*** SCL stop hold time is the delay from the rising edge of SCL (IIC clock) to the rising edge of SDA SDA (IIC data) while SCL is high (Stop condition). **SCL Stop hold time = bus period (s) × mul × SCL Stop hold value** ***Eqn. 12-4***

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 100kbps.

Table 12-3. Hold Time Values for 8 MHz Bus Speed

MULT	ICR	Hold Times (µs)		
		SDA	SCL Start	SCL Stop
0x2	0x00	3.500	4.750	5.125
0x1	0x07	2.500	4.250	5.125
0x1	0x0B	2.250	4.000	5.250
0x0	0x14	2.125	4.000	5.250
0x0	0x18	1.125	3.000	5.500

Table 12-4. IIC Divider and Hold Values

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SDA Hold (Stop) Value	ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SCL Hold (Stop) Value
00	20	7	6	11	**20**	160	17	78	81
01	22	7	7	12	**21**	192	17	94	97
02	24	8	8	13	**22**	224	33	110	113
03	26	8	9	14	**23**	256	33	126	129
04	28	9	10	15	**24**	288	49	142	145
05	30	9	11	16	**25**	320	49	158	161
06	34	10	13	18	**26**	384	65	190	193
07	40	10	16	21	**27**	480	65	238	241
08	28	7	10	15	**28**	320	33	158	161
09	32	7	12	17	**29**	384	33	190	193
0A	36	9	14	19	**2A**	448	65	222	225
0B	40	9	16	21	**2B**	512	65	254	257
0C	44	11	18	23	**2C**	576	97	286	289
0D	48	11	20	25	**2D**	640	97	318	321
0E	56	13	24	29	**2E**	768	129	382	385
0F	68	13	30	35	**2F**	960	129	478	481
10	48	9	18	25	**30**	640	65	318	321
11	56	9	22	29	**31**	768	65	382	385
12	64	13	26	33	**32**	896	129	446	449
13	72	13	30	37	**33**	1024	129	510	513
14	80	17	34	41	**34**	1152	193	574	577
15	88	17	38	45	**35**	1280	193	638	641
16	104	21	46	53	**36**	1536	257	766	769
17	128	21	58	65	**37**	1920	257	958	961
18	80	9	38	41	**38**	1280	129	638	641
19	96	9	46	49	**39**	1536	129	766	769
1A	112	17	54	57	**3A**	1792	257	894	897
1B	128	17	62	65	**3B**	2048	257	1022	1025
1C	144	25	70	73	**3C**	2304	385	1150	1153
1D	160	25	78	81	**3D**	2560	385	1278	1281
1E	192	33	94	97	**3E**	3072	513	1534	1537
1F	240	33	118	121	**3F**	3840	513	1918	1921

12.3.3 IIC Control Register (IICC1)

	7	6	5	4	3	2	1	0
R	IICEN	IICIE	MST	TX	TXAK	0	0	0
W						RSTA	▨	▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 12-4. IIC Control Register (IICC1)

Table 12-5. IICC1 Field Descriptions

Field	Description
7 IICEN	IIC Enable. The IICEN bit determines whether the IIC module is enabled. 0 IIC is not enabled 1 IIC is enabled
6 IICIE	IIC Interrupt Enable. The IICIE bit determines whether an IIC interrupt is requested. 0 IIC interrupt request not enabled 1 IIC interrupt request enabled
5 MST	Master Mode Select. The MST bit changes from a 0 to a 1 when a start signal is generated on the bus and master mode is selected. When this bit changes from a 1 to a 0 a stop signal is generated and the mode of operation changes from master to slave. 0 Slave mode 1 Master mode
4 TX	Transmit Mode Select. The TX bit selects the direction of master and slave transfers. In master mode, this bit should 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 should be set by software according to the SRW bit in the status register. 0 Receive 1 Transmit
3 TXAK	Transmit Acknowledge Enable. This bit specifies the value driven onto the SDA during data acknowledge cycles for master and slave receivers. 0 An acknowledge signal is sent out to the bus after receiving one data byte 1 No acknowledge signal response is sent
2 RSTA	Repeat start. Writing a 1 to this bit generates a repeated start condition provided it is the current master. This bit is always read as cleared. Attempting a repeat at the wrong time results in loss of arbitration.

12.3.4 IIC Status Register (IICS)

	7	6	5	4	3	2	1	0
R	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
W	▨		▨		▨	▨		▨
Reset	1	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 12-5. IIC Status Register (IICS)

Table 12-6. IICS Field Descriptions

Field	Description
7 TCF	Transfer Complete Flag. This bit is set on the completion of a byte transfer. 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	Addressed as a Slave. The IAAS bit is set when the calling address matches the programmed slave address or when the GCAEN bit is set and a general call is received. Writing the IICC register clears this bit. 0 Not addressed 1 Addressed as a slave
5 BUSY	Bus Busy. 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	Arbitration Lost. This bit is set by hardware when the arbitration procedure is lost. The ARBL bit must be cleared by software by writing a 1 to it. 0 Standard bus operation 1 Loss of arbitration
2 SRW	Slave Read/Write. When addressed as a slave, the SRW bit indicates the value of the R/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
1 IICIF	IIC Interrupt Flag. The IICIF bit is set when an interrupt is pending. This bit must be cleared by software, by writing a 1 to it in the interrupt routine. One of the following events can set the IICIF bit: • One byte transfer completes • Match of slave address to calling address • Arbitration lost 0 No interrupt pending 1 Interrupt pending
0 RXAK	Receive Acknowledge. 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. 0 Acknowledge received 1 No acknowledge received

12.3.5 IIC Data I/O Register (IICD)

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

Figure 12-6. IIC Data I/O Register (IICD)

Table 12-7. IICD Field Descriptions

Field	Description
7–0 DATA	**Data** — 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.

NOTE

When transitioning out of master receive mode, the IIC mode should 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.

The 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, reading the IICD does not initiate the receive.

Reading the IICD returns the last byte received while the IIC is configured in master receive or slave receive modes. The IICD does not reflect every byte 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 is used for the address transfer and should comprise of the calling address (in bit 7 to bit 1) concatenated with the required R/$\overline{\text{W}}$ bit (in position bit 0).

12.3.6 IIC Control Register 2 (IICC2)

	7	6	5	4	3	2	1	0
R	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
W			▨	▨	▨			
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 12-7. IIC Control Register (IICC2)

Table 12-8. IICC2 Field Descriptions

Field	Description
7 GCAEN	General Call Address Enable. The GCAEN bit enables or disables general call address. 0 General call address is disabled 1 General call address is enabled
6 ADEXT	Address Extension. The ADEXT bit controls the number of bits used for the slave address. 0 7-bit address scheme 1 10-bit address scheme
2–0 AD[10:8]	Slave Address. 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.

12.4 Functional Description

This section provides a complete functional description of the IIC module.

12.4.1 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 illustrated in Figure 12-8.

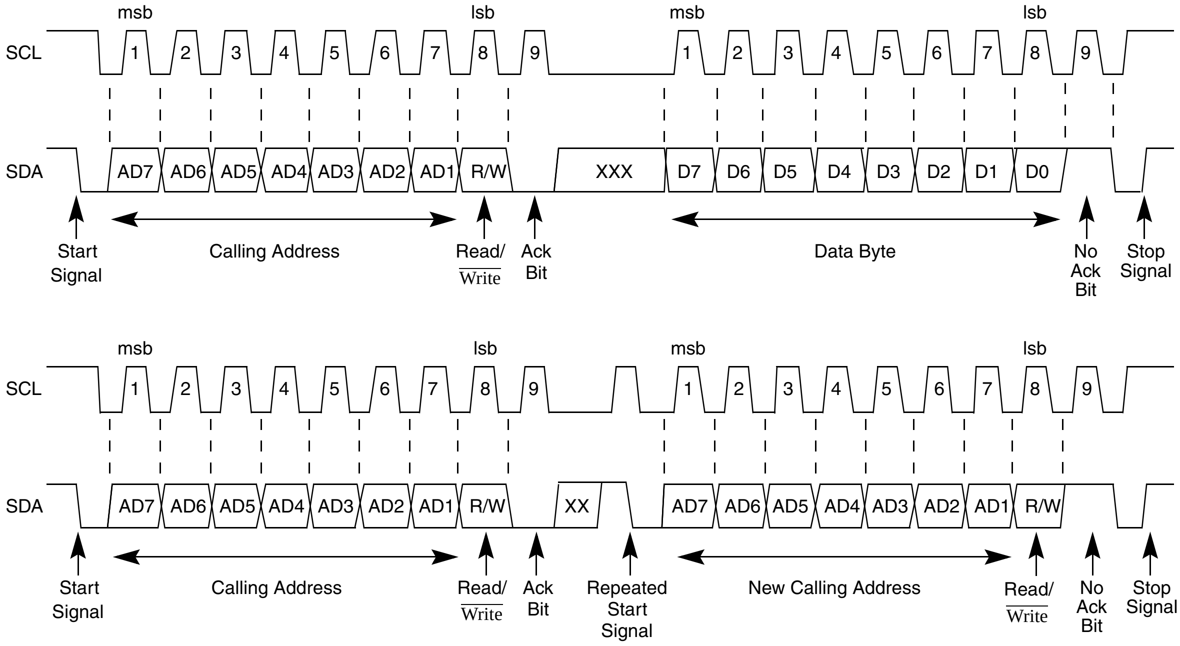


Figure 12-8. IIC Bus Transmission Signals

12.4.1.1 Start Signal

When the bus is free, no master device is engaging the bus (SCL and SDA lines are at logical high), a master may initiate communication by sending a start signal. As shown in Figure 12-8, 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.

12.4.1.2 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 R/$\overline{\text{W}}$ bit. The R/$\overline{\text{W}}$ bit tells the slave the desired direction of data transfer.

1 = Read transfer, the slave transmits data to the master.
0 = Write transfer, the master transmits data to the slave.

Only the slave with a calling address that matches the one transmitted by the master responds by sending back an acknowledge bit. This is done by pulling the SDA low at the ninth clock (see Figure 12-8).

No two slaves in the system may have the same address. If the IIC module is the master, it must not transmit an address 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.

12.4.1.3 Data Transfer

Before successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/$\overline{\text{W}}$ 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 12-8. 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, which 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 ninth 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.

12.4.1.4 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 12-8).

The master can generate a stop even if the slave has generated an acknowledge at which point the slave must release the bus.

12.4.1.5 Repeated Start Signal

As shown in Figure 12-8, 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.

12.4.1.6 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.

12.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock remains within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see Figure 12-9). 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.

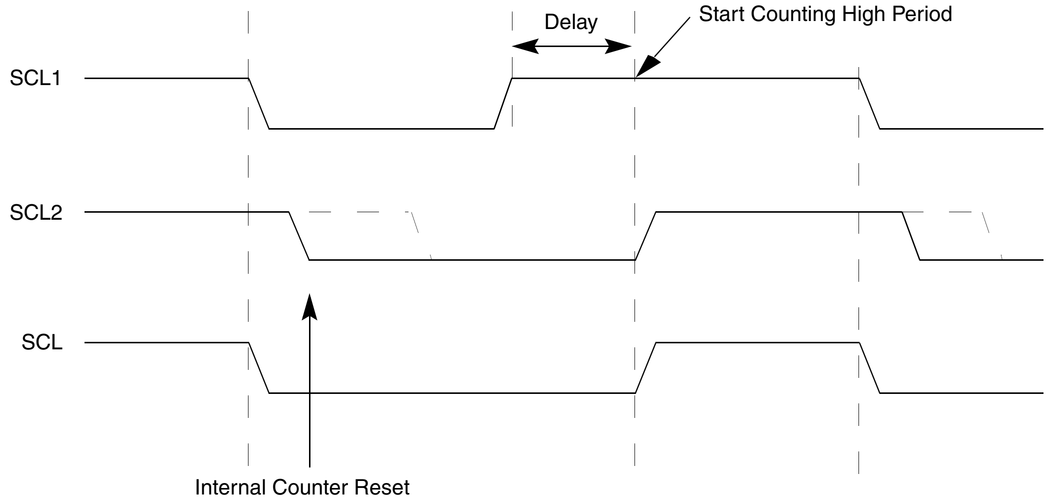


Figure 12-9. IIC Clock Synchronization

12.4.1.8 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 a case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

12.4.1.9 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.

12.4.2 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.

12.4.2.1 Master-Transmitter Addresses a Slave-Receiver

The transfer direction is not changed (see Table 12-9). 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. More than one device can find a match and generate an acknowledge (A1). Then, each slave that finds a match compares the eight bits of the second byte of the slave address with its own address. Only one slave finds a match and generates 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 12-9. 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. Software must ensure the contents of IICD are ignored and not treated as valid data for this interrupt.

12.4.2.2 Master-Receiver Addresses a Slave-Transmitter

The transfer direction is changed after the second R/$\overline{W}$ bit (see Table 12-10). 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 12-10. 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. Software must ensure the contents of IICD are ignored and not treated as valid data for this interrupt.

12.4.3 General Call Address

General calls can be requested in 7-bit address or 10-bit address. If the GCAEN bit is set, the IIC matches the general call address as well as its own slave address. When the IIC responds to a general call, it acts as a slave-receiver and the IAAS bit is set after the address cycle. Software must read the IICD register after the first byte transfer to determine whether the address matches is its own slave address or a general call. If the value is 00, the match is a general call. If the GCAEN bit is clear, the IIC ignores any data supplied from a general call address by not issuing an acknowledgement.

12.5 Resets

The IIC is disabled after reset. The IIC cannot cause an MCU reset.

12.6 Interrupts

The IIC generates a single interrupt.

An interrupt from the IIC is generated when any of the events in Table 12-11 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 a 1 to it in the interrupt routine. You can determine the interrupt type by reading the status register.

Table 12-11. 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

12.6.1 Byte Transfer Interrupt

The TCF (transfer complete flag) bit is set at the falling edge of the ninth clock to indicate the completion of byte transfer.

12.6.2 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.

12.6.3 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 writing a 1 to it.

12.7 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: IICA
   - — 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 12-11

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 12-11
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 12-11 can manage master and slave IIC operations. For slave operation, an incoming IIC message that contains the proper address begins IIC communication. For master operation, communication must be initiated by writing to the IICD register.

Register Model

IICA	AD[7:1]	0

When addressed as a slave (in slave mode), the module responds to this address

IICF	MULT	ICR

Baud rate = BUSCLK / (2 x MULT x (SCL DIVIDER))

IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0

Module configuration

IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK

Module status flags

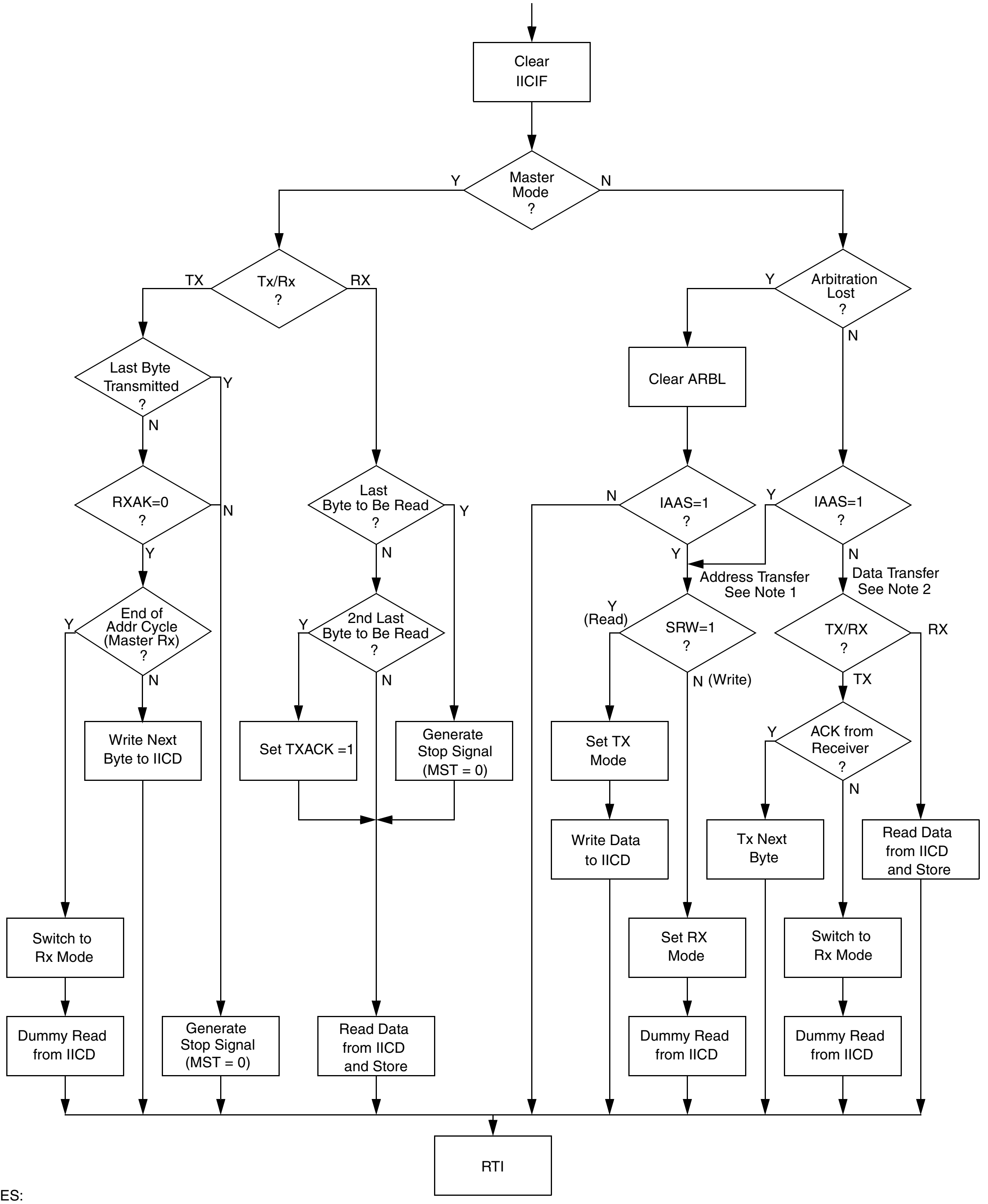
IICD	DATA

Data register; Write to transmit IIC data read to read IIC data

IICC2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8

Address configuration

Figure 12-10. IIC Module Quick Start



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 12-11. Typical IIC Interrupt Routine

Chapter 13
Interrupt Controller (CF1_INTC)

13.1 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 13-1 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 13-1. Exception Processing Comparison

Attribute	HCS08	V1 ColdFire
Exception Vector Table	32 two-byte entries, fixed location at upper end of memory	113 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 + 40 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 13-1. Exception Processing Comparison (continued)

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

13.1.1 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 ... > 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 × 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.

Table 13-2 shows the exception priorities for the MCF51AC256 series of microcontrollers.

Table 13-2. MCF51AC256 Series Exception/Interrupt Priority Table

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment	Vector Number(s)
0	0x000	—	—	Initial supervisor stack pointer	0
1	0x004	—	—	Initial program counter	1
2–63	0x008–0x0FC	—	—	Reserved for internal CPU exceptions (see Table 7-6)	2–63
64	0x100	7,mid	Next	IRQ_pin	64
65	0x104	7,3	Next	Low_voltage_detect	65
66	0x108	7,2	Next	MCG_lock	66
67	0x10C	5,6	Next	FTM1_ch0	67
68	0x110	5,5	Next	FTM1_ch1	68
69	0x114	5,4	Next	FTM1_ch2	69
70	0x118	5,mid	Next	FTM1_ch3	70
71	0x11C	5,3	Next	FTM1_ch4	71
72	0x120	5,2	Next	FTM1_ch5	72

Table 13-2. MCF51AC256 Series Exception/Interrupt Priority Table (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment	Vector Number(s)
73	0x124	5,1	Next	FTM1_ovfl	73
74	0x128	4,6	Next	FTM2_ch0	74
75	0x12C	4,5	Next	FTM2_ch1	75
76	0x130	4,4	Next	FTM2_ch2	76
77	0x134	4,mid	Next	FTM2_ch3	77
78	0x138	4,3	Next	FTM2_ch4	78
79	0x13C	4,2	Next	FTM2_ch5	79
80	0x140	4,1	Next	FTM2_ovfl	80
81	0x144	3,7	Next	SPI1	81
82	0x148	3,3	Next	SCI1_err	82
83	0x14C	3,2	Next	SCI1_rx	83
84	0x150	3,1	Next	SCI1_tx	84
85	0x154	2,7	Next	SCI2_err	85
86	0x158	2,6	Next	SCI2_rx	86
87	0x15C	2,5	Next	SCI2_tx	87
88	0x160	2,4	Next	KBI1	88
89	0x164	2,3	Next	ADC1	89
90	0x168	2,2	Next	IIC1	90
91	0x16C	2,1	Next	RTI	91
92	0x170	1,6	Next	TPM3_ch0	92
93	0x174	1,5	Next	TPM3_ch1	93
94–102	0x178–0x198		—	Reserved; unused for V1	94–102
103	0x19C	7,0	Next	Level 7 Software Interrupt	103
104	0x1A0	6,0	Next	Level 6 Software Interrupt	104
105	0x1A4	5,0	Next	Level 5 Software Interrupt	105
106	0x1A8	4,0	Next	Level 4 Software Interrupt	106

Table 13-2. MCF51AC256 Series Exception/Interrupt Priority Table (continued)

Vector Number(s)	Vector Address Offset (Hex)	Interrupt Level, Priority	Stacked Program Counter	Assignment	Vector Number(s)
107	0x1AC	3,0	Next	Level 3 Software Interrupt	107
108	0x1B0	2,0	Next	Level 2 Software Interrupt	108
109	0x1B4	1,0	Next	Level 1 Software Interrupt	109
110	0x1B8	1,4	Next	TPM3_ovfl	110
111	0x1BC	1,3	Next	SPI2	111
112	0x1C0	5,7	Next	FTM1_fault	112
113	0x1C4	4,7	Next	FTM2_fault	113
114	0x1C8	3,6	Next	MSCAN_wakeup	114
115	0x1CC	3,5	Next	MSCAN_errors	115
116	0x1D0	3,4	Next	MSCAN_Rx	116
117	0x1D4	3,mid	Next	MSCAN_Tx	117
118	0x1D8	1,2	Next	ACMP1	118
119	0x1DC	1,1	Next	ACMP2	119
120–255	0x1E0–0x3FC		—	Reserved; unused for V1	120–255

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:

```
ColdFire Vector Number = 62 + 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 13-1.

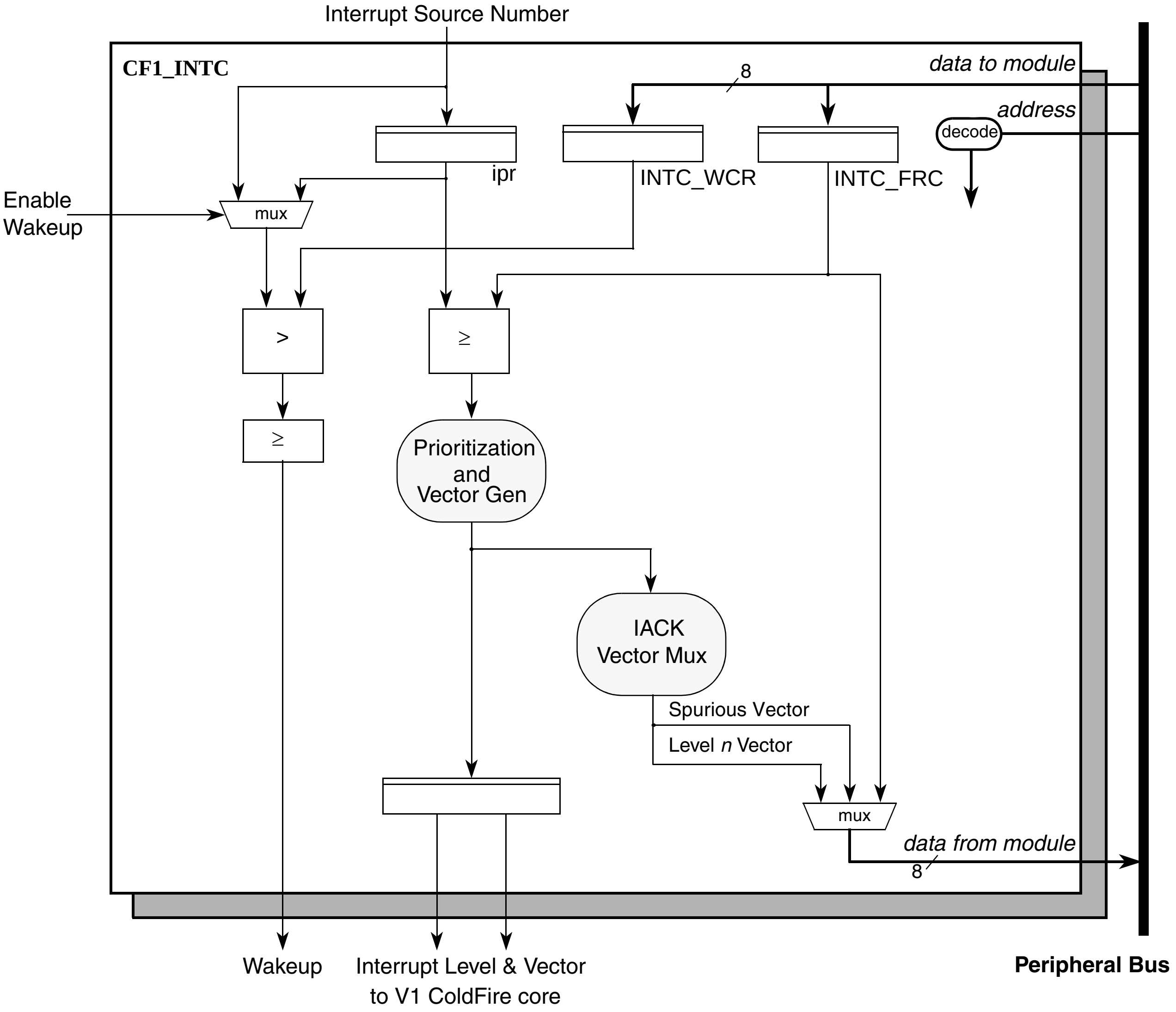


Figure 13-1. CF1_INTC Block Diagram

13.1.2 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 40 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests

- Fixed association between interrupt request source and level plus priority
  - 40 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 and stop modes

13.1.3 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 or stop 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.

13.2 External Signal Description

The CF1_INTC module does not include any external interfaces.

13.3 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 13-3.

Table 13-3. CF1_INTC Memory Map

Offset Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/ Page
0x10	INTC_FRC	CF1_INTC Force Interrupt Register	8	R/W	0x00	13.3.1/13-8
0x18	INTC_PL6P7	CF1_INTC Programmable Level 6, Priority 7	8	R/W	0x00	13.3.2/13-9
0x19	INTC_PL6P6	CF1_INTC Programmable Level 6, Priority 6	8	R/W	0x00	13.3.2/13-9
0x1B	INTC_WCR	CF1_INTC Wakeup Control Register	8	R/W	0x80	13.3.3/13-10
0x1E	INTC_SFRC	CF1_INTC Set Interrupt Force Register	8	Write	—	13.3.4/13-11
0x1F	INTC_CFRC	CF1_INTC Clear Interrupt Force Register	8	Write	—	13.3.5/13-12
0x20	INTC_SWIACK	CF1_INTC Software Interrupt Acknowledge	8	Read	0x00	13.3.6/13-13
0x24	INTC_LVL1IACK	CF1_INTC Level 1 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x28	INTC_LVL2IACK	CF1_INTC Level 2 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x2C	INTC_LVL3IACK	CF1_INTC Level 3 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x30	INTC_LVL4IACK	CF1_INTC Level 4 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x34	INTC_LVL5IACK	CF1_INTC Level 5 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x38	INTC_LVL6IACK	CF1_INTC Level 6 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13
0x3C	INTC_LVL7IACK	CF1_INTC Level 7 Interrupt Acknowledge	8	Read	0x18	13.3.6/13-13

13.3.1 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, 39–32. This is for compatibility with other ColdFire interrupt controllers.

Offset: CF1_INTC_BASE + 0x10 (INTC_FRC) Access: Read/Write

	39	38	37	36	35	34	33	32
R	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7
W								
Reset	0	0	0	0	0	0	0	0

Figure 13-2. Force Interrupt Register (INTC_FRC)

Table 13-4. INTC_FRC Field Descriptions

Field	Description
39	Reserved, must be cleared.
38 LVL1	Force Level 1 interrupt. 0 Negates the forced level 1 interrupt request. 1 Forces a level 1 interrupt request.
37 LVL2	Force Level 2 interrupt. 0 Negates the forced level 2 interrupt request. 1 Forces a level 2 interrupt request.
36 LVL3	Force Level 3 interrupt. 0 Negates the forced level 3 interrupt request. 1 Forces a level 3 interrupt request.
35 LVL4	Force Level 4 interrupt. 0 Negates the forced level 4 interrupt request. 1 Forces a level 4 interrupt request.
34 LVL5	Force Level 5 interrupt. 0 Negates the forced level 5 interrupt request. 1 Forces a level 5 interrupt request.
33 LVL6	Force Level 6 interrupt. 0 Negates the forced level 6 interrupt request. 1 Forces a level 6 interrupt request.
32 LVL7	Force Level 7 interrupt. 0 Negates the forced level 7 interrupt request. 1 Forces a level 7 interrupt request.

13.3.2 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 13.6.2, “Using INTC_PL6P{7,6} Registers.”

Offset: CF1_INTC_BASE + 0x18 (INTC_PL6P7)
CF1_INTC_BASE + 0x19 (INTC_PL6P6)

Access: Read/Write

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

Figure 13-3. Programmable Level 6, Priority {7,6} Registers (INTC_PL6P{7,6})

Table 13-5. INTC_PL6P{7,6} Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4–0 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. **Note:** The value must be a valid interrupt number. Unused or reserved interrupt numbers are ignored.

13.3.3 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 or stop modes. The INTC_WCR register defines wakeup condition for interrupt recognition during wait and stop modes. 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 or stop 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 to place the processor into wait or stop 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 wakeup signal is defined as:

```
wakeup = INTC_WCR[ENB] & (level of any asserted_int_request > INTC_WCR[MASK])
```

Offset: CF1_INTC_BASE + 0x1B (INTC_WCR) Access: Read/Write

	7	6	5	4	3	2	1	0
R	ENB	0	0	0	0	MASK		
W								
Reset	1	0	0	0	0	0	0	0

Figure 13-4. Wakeup Control Register (INTC_WCR)

Table 13-6. INTC_WCR Field Descriptions

Field	Description
7 ENB	Enable wakeup signal. 0 Wakeup signal disabled 1 Enables the assertion of the combinational 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 or stop 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 wakeup signal to the clock generation logic.

13.3.4 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

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

Figure 13-5. INTC_SFRC Register

Table 13-7. INTC_SFRC Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SET	For data values within the 32–38 range, the corresponding bit in the INTC_FRC register is set, as defined below. 0x20 Bit 32, INTC_FRC[LVL7] is set 0x21 Bit 33, INTC_FRC[LVL6] is set 0x22 Bit 34, INTC_FRC[LVL5] is set 0x23 Bit 35, INTC_FRC[LVL4] is set 0x24 Bit 36, INTC_FRC[LVL3] is set 0x25 Bit 37, INTC_FRC[LVL2] is set 0x26 Bit 38, INTC_FRC[LVL1] is set **Note:** Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x20–0x26 (32–38) range to ensure compatibility with future devices.

13.3.5 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

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

Figure 13-6. INTC_CFRC Register

Table 13-8. INTC_CFRC Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 CLR	For data values within the 32–38 range, the corresponding bit in the INTC_FRC register is cleared, as defined below. 0x20 Bit 32, INTC_FRC[LVL7] is cleared 0x21 Bit 33, INTC_FRC[LVL6] is cleared 0x22 Bit 34, INTC_FRC[LVL5] is cleared 0x23 Bit 35, INTC_FRC[LVL4] is cleared 0x24 Bit 36, INTC_FRC[LVL3] is cleared 0x25 Bit 37, INTC_FRC[LVL2] is cleared 0x26 Bit 38, INTC_FRC[LVL1] is cleared **Note:** Data values outside this range do not affect the INTC_FRC register. It is recommended the data values be restricted to the 0x20–0x26 (32–38) range to ensure compatibility with future devices.

13.3.6 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 13.6.3, “More on Software IACKs.”

Offset: CF1_INTC_BASE + 0x20 (INTC_SWIACK)
CF1_INTC_BASE + 0x20 + (4×*n*) (INTC_LVL*n*IACK)

Access: Read-only

	7	6	5	4	3	2	1	0
R	0	VECN						
W								
SWIACK Reset	0	0	0	0	0	0	0	0
LVL*n*IACK Reset	0	0	0	1	1	0	0	0

Table 13-9. Software and Level-n IACK Registers (INTC_SWIACK, INTC_LVL*n*IACK)

Table 13-10. INTC_SWIACK, INTC_LVL*n*IACK Field Descriptions

Field	Description
7	Reserved, must be cleared.
6–0 VECN	Vector number. Indicates the appropriate vector number. 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. For the LVL*n*IACK register, it is the highest priority request within the specified level-*n*. If there are no pending requests within the level, VECN is 0x18 (24) to signal a spurious interrupt.

13.3.7 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 13-12) and a tabular representation based on request priority (Table 13-13).

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 13-11. Legend for Table 13-12

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 13-12. [Level][Priority within Level] Matrix Interrupt Assignments

	Priority within Level																	
	7		6		5		4		midpoint		3		2		1		0	
7	—		—		—		—		IRQ		LVD		MCG_lock		—		force_lvl7	
									0	64	1	65	2	66			FRC[lvl7]	103
6	remppped		remppped		—		—		—		—		—		—		force_lvl6	
	PL6P7	*	PL6P6	*													FRC[lvl6]	104
5	FTM1_fault		FTM1_ch0		FTM1_ch1		FTM1_ch2		FTM1_ch3		FTM1_ch4		FTM1_ch5		FTM1_ovfl		force_lvl5	
	48	112	3	67	4	68	5	69	6	70	7	71	8	72	9	73	FRC[lvl5]	105

Table 13-12. [Level][Priority within Level] Matrix Interrupt Assignments (continued)

	Priority within Level																	
	7		6		5		4		midpoint		3		2		1		0	
4	FTM2_fault		FTM2_ch0		FTM2_ch1		FTM2_ch2		FTM2_ch3		FTM2_ch4		FTM2_ch5		FTM2_ovfl		force_lvl4	
	49	113	10	74	11	75	12	76	13	77	14	78	15	79	16	80	FRC[lvl4]	106
3	SPI1		MSCAN _wakeup		MSCAN _errors		MSCAN _RX		MSCAN _Tx		SCI1_err		SCI1_rx		SCI1_tx		force_lvl3	
	17	81	50	114	51	115	52	116	53	117	18	82	19	83	20	84	FRC[lvl3]	107
2	SCI2_err		SCI2_rx		SCI2_tx		KBI1		—		ADC1		IIC1		RTI		force_lvl2	
	21	85	22	86	23	87	24	88			25	89	26	90	27	91	FRC[lvl2]	108
1	—		TPM3_ch0		TPM3_ch1		TPM3_ovfl		—		SPI2		ACMP1		ACMP2		force_lvl1	
			28	92	29	93	46	110			47	111	54	118	55	119	FRC[lvl1]	109

Table 13-13 presents the same information on interrupt request assignments, but from the highest priority request to the lowest.

Table 13-13. Interrupt Assignments

IRQ Source	Level	Priority with Level	Interrupt Source Number	Vector
IRQ_pin	7	mid	0	64
Low_voltage_detect	7	3	1	65
MCG_lock	7	2	2	66
force_lvl7	7	0	INTC_FRC[lvl7]	103
remapped_l6p7	6	7	INTC_PL6L7	*
remapped_l6p6	6	6	INTC_PL6L7	*
force_lvl6	6	0	INTC_FRC[lvl6]	104
FTM1_fault	5	7	48	112
FTM1_ch0	5	6	3	67
FTM1_ch1	5	5	4	68
FTM1_ch2	5	4	5	69
FTM1_ch3	5	mid	6	70
FTM1_ch4	5	3	7	71
FTM1_ch5	5	2	8	72
FTM1_ovfl	5	1	9	73
force_lvl5	5	0	INTC_FRC[lvl5]	105
FTM2_fault	4	7	49	113
FTM2_ch0	4	6	10	74
FTM2_ch1	4	5	11	75
FTM2_ch2	4	4	12	76
FTM2_ch3	4	mid	13	77
FTM2_ch4	4	3	14	78

Table 13-13. Interrupt Assignments

IRQ Source	Level	Priority with Level	Interrupt Source Number	Vector
FTM2_ch5	4	2	15	79
FTM2_ovfl	4	1	16	80
force_lvl4	4	0	INTC_FRC[lvl4]	106
SPI1	3	7	17	81
MSCAN_wakeup	3	6	50	114
MSCAN_errors	3	5	51	115
MSCAN_Rx	3	4	52	116
MSCAN_Tx	3	mid	53	117
SCI1_err	3	3	18	82
SCI1_rx	3	2	19	83
SCI1_tx	3	1	20	84
force_lvl3	3	0	INTC_FRC[lvl3]	107
SCI2_err	2	7	21	85
SCI2_rx	2	6	22	86
SCI2_tx	2	5	23	87
KBI1	2	4	24	88
ADC1	2	3	25	89
IIC1	2	2	26	90
RTI	2	1	27	91
force_lvl2	2	0	INTC_FRC[lvl2]	108
TPM3_ch0	1	6	28	92
TPM3_ch1	1	5	29	93
TPM3_ovfl	1	4	46	110
SPI2	1	3	47	111
ACMP1	1	2	54	118
ACMP2	1	1	55	119
force_lvl1	1	0	INTC_FRC[lvl1]	109

13.4 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.

13.4.1 Handling of Non-Maskable Level 7 Interrupt Requests

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.

13.5 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). On the first revision of silicon for this device, the wakeup control register (INTC_WCR) is disabled, so it must be written before the processor executes any stop instructions to properly exit from any wait or stop mode.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.

13.6 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.

13.6.1 Emulation of the HCS08's 1-Level IRQ Handling

As noted in Table 13-1, 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.

13.6.2 Using INTC_PL6P{7,6} Registers

Section 13.3.2, "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 (SCI1) as the highest two maskable interrupts. The default assignments for the SCI1 transmit and receive interrupts are:

- sci1_rx = interrupt source 19 = vector 83 = level 3, priority 3
- sci1_tx = interrupt source 20 = vector 84 = level 3, priority 2

To remap these two requests, the INTC_PL6P{7,6} registers are programmed with the desired interrupt source number:

- Setting INTC_PL6P7 to 19(0x13), remaps sci1_rx as level 6, priority 7.
- Setting INTC_PL6P6 to 18(0x12), remaps sci1_tx as level 6, priority 6.

The reset state of the INTC_PL6P{7,6} registers disables any request remapping.

Another example is to remap the interrupt requests of two ACMPx as the highest two maskable interrupts. The default assignments for the ACMP1 and ACMP2 e interrupts are:

- ACMP1 = interrupt source 54 = vector 118 = level 2, priority 2
- ACMP2 = interrupt source 55 = vector 119 = level 2, priority 1

To remap these two requests, the INTC_PL6P{7,6} registers are programmed with the desired interrupt source number:

- Setting INTC_PL6P7 to 38(0x26), remaps ACMP1 as level 6, priority 7.
- Setting INTC_PL6P6 to 39(0x27), remaps ACMP2 as level 6, priority 6.

NOTE

The INTC_PL6P{7,6} registers are programmed with the desired interrupt source number if its vector number lies in range of 64 to 93. If its vector number is larger than 109, the INTC_PL6P{7,6} registers are programmed with the desired interrupt source number minus 16.

13.6.3 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 13-7.

```
                 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_alternate_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 13-7. 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 14
Keyboard Interrupt (KBIV1)

14.1 Introduction

The keyboard interrupt (KBI) module provides up to eight independently enabled external interrupt sources.

14.1.1 KBI Block Diagram

Figure 14-1 shows the block diagram for a KBI module.

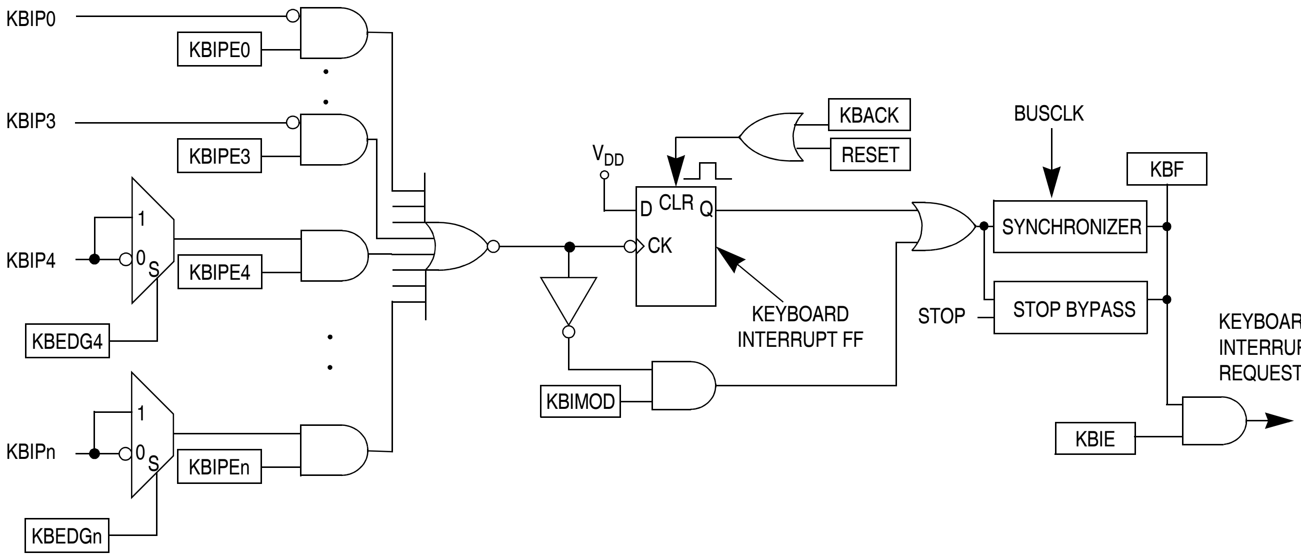


Figure 14-1. KBI Block Diagram

14.2 Register Definition

This section provides information about all registers and control bits associated with the KBI module.

Refer to the direct-page register summary in the Memory chapter of this data sheet for the absolute address assignments for all KBI 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.

14.2.1 KBI Status and Control Register (KBISC)

	7	6	5	4	3	2	1	0
R	KBEDG7	KBEDG6	KBEDG5	KBEDG4	KBF	0	KBIE	KBIMOD
W					▨	KBACK		
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 14-2. KBI Status and Control Register (KBISC)

Table 14-1. KBISC Register Field Descriptions

Field	Description
7:4 KBEDG[7:4]	**Keyboard Edge Select for KBI Port Bits** — Each of these read/write bits selects the polarity of the edges and/or levels that are recognized as trigger events on the corresponding KBI port pin when it is configured as a keyboard interrupt input (KBIPEn = 1). Also see the KBIMOD control bit, which determines whether the pin is sensitive to edges-only or edges and levels. 0 Falling edges/low levels 1 Rising edges/high levels
3 KBF	**Keyboard Interrupt Flag** — This read-only status flag is set whenever the selected edge event has been detected on any of the enabled KBI port pins. This flag is cleared by writing a 1 to the KBACK control bit. The flag will remain set if KBIMOD = 1 to select edge-and-level operation and any enabled KBI port pin remains at the asserted level. KBF can be used as a software pollable flag (KBIE = 0) or it can generate a hardware interrupt request to the CPU (KBIE = 1). 0 No KBI interrupt pending 1 KBI interrupt pending
2 KBACK	**Keyboard Interrupt Acknowledge** — This write-only bit (reads always return 0) is used to clear the KBF status flag by writing a 1 to KBACK. When KBIMOD = 1 to select edge-and-level operation and any enabled KBI port pin remains at the asserted level, KBF is being continuously set so writing 1 to KBACK does not clear the KBF flag.
1 KBIE	**Keyboard Interrupt Enable** — This read/write control bit determines whether hardware interrupts are generated when the KBF status flag equals 1. When KBIE = 0, no hardware interrupts are generated, but KBF can still be used for software polling. 0 KBF does not generate hardware interrupts (use polling) 1 KBI hardware interrupt requested when KBF = 1
KBIMOD	**Keyboard Detection Mode** — This read/write control bit selects either edge-only detection or edge-and-level detection. KBI port bits 3 through 0 can detect falling edges-only or falling edges and low levels. KBI port bits 7 through 4 can be configured to detect either: • Rising edges-only or rising edges and high levels (KBEDGn = 1) • Falling edges-only or falling edges and low levels (KBEDGn = 0) 0 Edge-only detection 1 Edge-and-level detection

14.2.2 KBI Pin Enable Register (KBIPE)

	7	6	5	4	3	2	1	0
R W	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 14-3. KBI Pin Enable Register (KBIPE)

Table 14-2. KBIPE Register Field Descriptions

Field	Description
7:0 KBIPE[7:0]	**Keyboard Pin Enable for KBI Port Bits** — Each of these read/write bits selects whether the associated KBI port pin is enabled as a keyboard interrupt input or functions as a general-purpose I/O pin. 0 Bit n of KBI port is a general-purpose I/O pin not associated with the KBI 1 Bit n of KBI port enabled as a keyboard interrupt input

14.3 Functional Description

14.3.1 Pin Enables

The KBIPEn control bits in the KBIPE register allow a user to enable (KBIPEn = 1) any combination of KBI-related port pins to be connected to the KBI module. Pins corresponding to 0s in KBIPE are general-purpose I/O pins that are not associated with the KBI module.

14.3.2 Edge and Level Sensitivity

Synchronous logic is used to detect edges. Prior to detecting an edge, enabled keyboard inputs in a KBI module must be at the deasserted logic level.

A falling edge is detected when an enabled keyboard 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 signal is seen as a logic 0 during one bus cycle and then a logic 1 during the next cycle.

The KBIMOD control bit can be set to reconfigure the detection logic so that it detects edges and levels. In KBIMOD = 1 mode, the KBF status flag becomes set when an edge is detected (when one or more enabled pins change from the deasserted to the asserted level while all other enabled pins remain at their deasserted levels), but the flag is continuously set (and cannot be cleared) as long as any enabled keyboard input pin remains at the asserted level. When the MCU enters stop mode, the synchronous edge-detection logic is bypassed (because clocks are stopped). In stop mode, KBI inputs act as asynchronous level-sensitive inputs so they can wake the MCU from stop mode.

14.3.3 KBI Interrupt Controls

The KBF status flag becomes set (1) when an edge event has been detected on any KBI input pin. If KBIE = 1 in the KBISC register, a hardware interrupt will be requested whenever KBF = 1. The KBF flag is cleared by writing a 1 to the keyboard acknowledge (KBACK) bit.

When KBIMOD = 0 (selecting edge-only operation), KBF is always cleared by writing 1 to KBACK. When KBIMOD = 1 (selecting edge-and-level operation), KBF cannot be cleared as long as any keyboard input is at its asserted level.

Chapter 15
Freescale's Controller Area Network (MSCANV1)

15.1 Introduction

Freescale's controller area network (MSCAN) definition is based on the MSCAN12 definition, which is the specific implementation of the MSCAN concept targeted for the M68HC12 microcontroller Family.

The module is a communication controller implementing the CAN 2.0A/B protocol as defined in the Bosch specification dated September 1991. For users to fully understand the MSCAN specification, it is recommended that the Bosch specification be read first to familiarize the reader with the terms and concepts contained within this document.

Though not exclusively intended for automotive applications, CAN protocol is designed to meet the specific requirements of a vehicle serial data bus: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth.

MSCAN uses an advanced buffer arrangement resulting in predictable real-time behavior and simplified application software.

15.1.1 Features

The basic features of the MSCAN are as follows:

- Implementation of the CAN protocol — Version 2.0 A/B
  - Standard and extended data frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbps[1]
  - Support for remote frames
- Five receive buffers with FIFO storage scheme
- Three transmit buffers with internal prioritization using a "local priority" concept
- Flexible maskable identifier filter supports two full-size (32-bit) extended identifier filters, or four 16-bit filters, or eight 8-bit filters
- Programmable wakeup functionality with integrated low-pass filter
- Programmable loopback mode supports self-test operation
- Programmable listen-only mode for monitoring of CAN bus
- Programmable bus-off recovery functionality
- Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states (warning, error passive, bus-off)

[1] Depending on the actual bit timing and the clock jitter of the PLL.

- Programmable MSCAN clock source either bus clock or oscillator clock
- Internal timer for time-stamping of received and transmitted messages
- Three low-power modes: sleep, power down, and MSCAN enable
- Global initialization of configuration registers

15.1.2 Modes of Operation

The following modes of operation are specific to the MSCAN. See Section 15.5, "Functional Description," for details.

- Listen-Only Mode
- MSCAN Sleep Mode
- MSCAN Initialization Mode
- MSCAN Power Down Mode
- Loopback Self Test Mode

15.1.3 Block Diagram

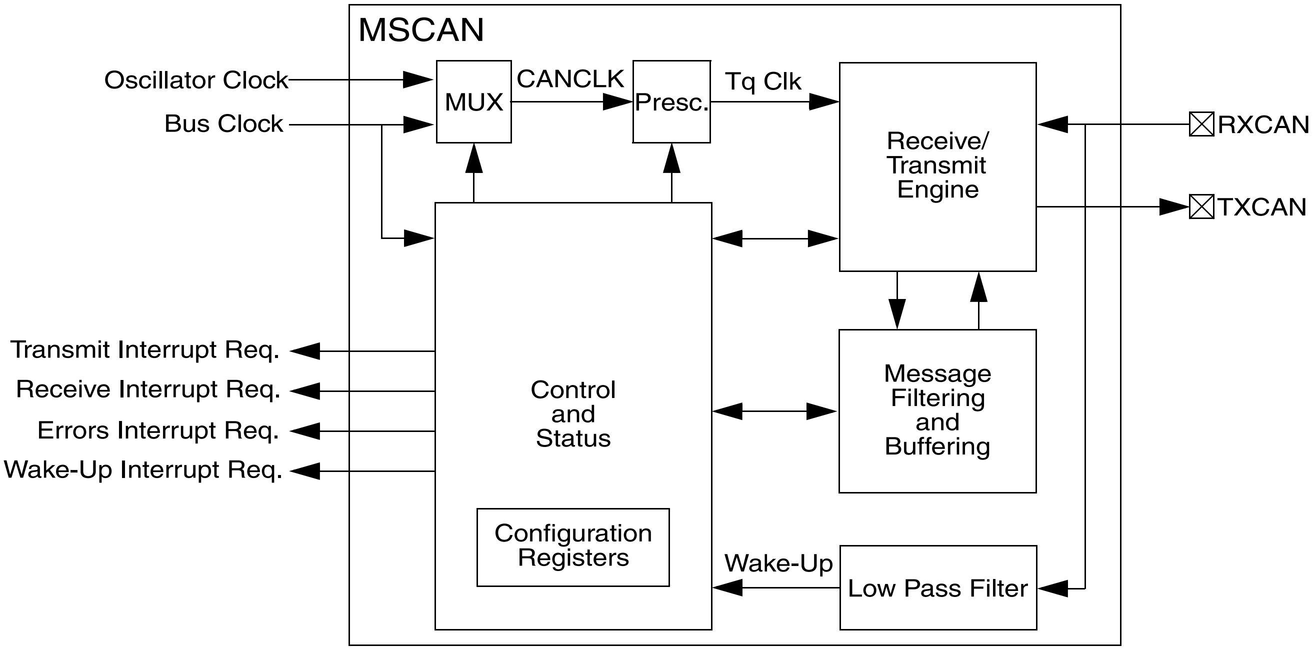


Figure 15-1. MSCAN Block Diagram

15.2 External Signal Description

The MSCAN uses two external pins:

15.2.1 RXCAN — CAN Receiver Input Pin

RXCAN is the MSCAN receiver input pin.

15.2.2 TXCAN — CAN Transmitter Output Pin

TXCAN is the MSCAN transmitter output pin. The TXCAN output pin represents the logic level on the CAN bus:

0 = Dominant state
1 = Recessive state

15.2.3 CAN System

A typical CAN system with MSCAN is shown in Figure 15-2. Each CAN node is connected physically to the CAN bus lines through a transceiver device. The transceiver is capable of driving the large current needed for the CAN bus and has current protection against defective CAN or defective nodes.

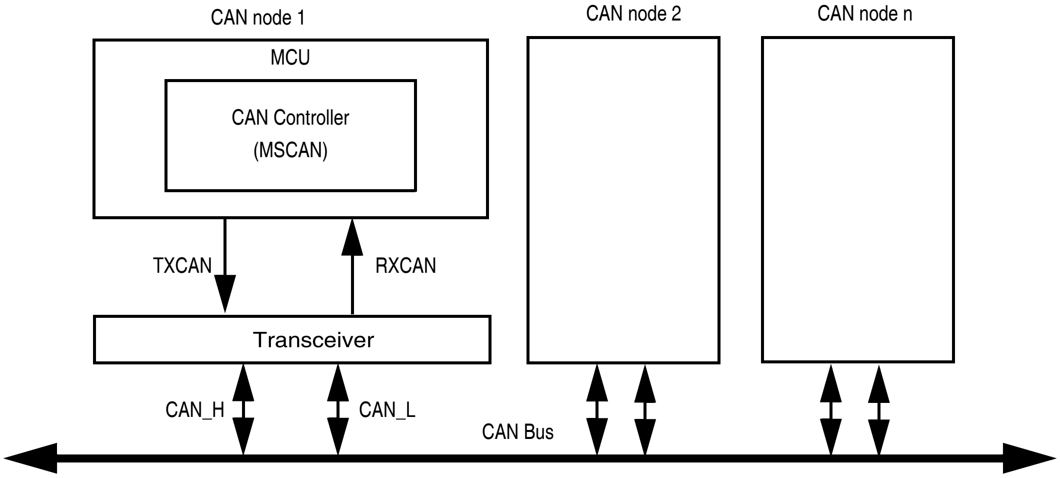


Figure 15-2. CAN System

15.3 Register Definition

This section describes in detail all the registers and register bits in the MSCAN module. 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. All bits of all registers in this module are completely synchronous to internal clocks during a register read.

15.3.1 MSCAN Control Register 0 (CANCTL0)

The CANCTL0 register provides various control bits of the MSCAN module as described below.

	7	6	5	4	3	2	1	0
R	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
W	▨	▨		▨				
Reset:	0	0	0	0	0	0	0	1

▨ = Unimplemented

Figure 15-3. MSCAN Control Register 0 (CANCTL0)

NOTE

The CANCTL0 register, except WUPE, INITRQ, and SLPRQ, is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

Read: Anytime

Write: Anytime when out of initialization mode; exceptions are read-only RXACT and SYNCH, RXFRM (which is set by the module only), and INITRQ (which is also writable in initialization mode).

Table 15-1. CANCTL0 Register Field Descriptions

Field	Description
7 RXFRM[1]	**Received Frame Flag** — This bit is read and clear only. It is set when a receiver has received a valid message correctly, independently of the filter configuration. After it is set, it remains set until cleared by software or reset. Clearing is done by writing a 1. Writing a 0 is ignored. This bit is not valid in loopback mode. 0 No valid message was received since last clearing this flag 1 A valid message was received since last clearing of this flag
6 RXACT	**Receiver Active Status** — This read-only flag indicates the MSCAN is receiving a message. The flag is controlled by the receiver front end. This bit is not valid in loopback mode. 0 MSCAN is transmitting or idle[2] 1 MSCAN is receiving a message (including when arbitration is lost)[2]
5 CSWAI[3]	**CAN Stops in Wait Mode** — Enabling this bit allows for lower power consumption in wait mode by disabling all the clocks at the CPU bus interface to the MSCAN module. 0 The module is not affected during wait mode 1 The module ceases to be clocked during wait mode
4 SYNCH	**Synchronized Status** — This read-only flag indicates whether the MSCAN is synchronized to the CAN bus and able to participate in the communication process. It is set and cleared by the MSCAN. 0 MSCAN is not synchronized to the CAN bus 1 MSCAN is synchronized to the CAN bus
3 TIME	**Timer Enable** — This bit activates an internal 16-bit wide free running timer which is clocked by the bit clock rate. If the timer is enabled, a 16-bit time stamp will be assigned to each transmitted/received message within the active TX/RX buffer. As soon as a message is acknowledged on the CAN bus, the time stamp will be written to the highest bytes (0x000E, 0x000F) in the appropriate buffer (see Section 15.4, "Programmer's Model of Message Storage"). The internal timer is reset (all bits set to 0) when disabled. This bit is held low in initialization mode. 0 Disable internal MSCAN timer 1 Enable internal MSCAN timer

Table 15-1. CANCTL0 Register Field Descriptions (continued)

Field	Description
2 WUPE[4]	**Wake-Up Enable** — This configuration bit allows the MSCAN to restart from sleep mode when traffic on CAN is detected (see Section 15.5.5.4, "MSCAN Sleep Mode"). This bit must be configured before sleep mode entry for the selected function to take effect. 0 Wake-up disabled — The MSCAN ignores traffic on CAN 1 Wake-up enabled — The MSCAN is able to restart
1 SLPRQ[5]	**Sleep Mode Request** — This bit requests the MSCAN to enter sleep mode, which is an internal power saving mode (see Section 15.5.5.4, "MSCAN Sleep Mode"). The sleep mode request is serviced when the CAN bus is idle, i.e., the module is not receiving a message and all transmit buffers are empty. The module indicates entry to sleep mode by setting SLPAK = 1 (see Section 15.3.2, "MSCAN Control Register 1 (CANCTL1)"). SLPRQ cannot be set while the WUPIF flag is set (see Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG)"). Sleep mode will be active until SLPRQ is cleared by the CPU or, depending on the setting of WUPE, the MSCAN detects activity on the CAN bus and clears SLPRQ itself. 0 Running — The MSCAN functions normally 1 Sleep mode request — The MSCAN enters sleep mode when CAN bus idle
0 INITRQ[6,7]	**Initialization Mode Request** — When this bit is set by the CPU, the MSCAN skips to initialization mode (see Section 15.5.5.5, "MSCAN Initialization Mode"). Any ongoing transmission or reception is aborted and synchronization to the CAN bus is lost. The module indicates entry to initialization mode by setting INITAK = 1 (Section 15.3.2, "MSCAN Control Register 1 (CANCTL1)"). The following registers enter their hard reset state and restore their default values: CANCTL0[8], CANRFLG[9], CANRIER[10], CANTFLG, CANTIER, CANTARQ, CANTAAK, and CANTBSEL. The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0-7, and CANIDMR0-7 can only be written by the CPU when the MSCAN is in initialization mode (INITRQ = 1 and INITAK = 1). The values of the error counters are not affected by initialization mode. When this bit is cleared by the CPU, the MSCAN restarts and then tries to synchronize to the CAN bus. If the MSCAN is not in bus-off state, it synchronizes after 11 consecutive recessive bits on the CAN bus; if the MSCAN is in bus-off state, it continues to wait for 128 occurrences of 11 consecutive recessive bits. Writing to other bits in CANCTL0, CANRFLG, CANRIER, CANTFLG, or CANTIER must be done only after initialization mode is exited, which is INITRQ = 0 and INITAK = 0. 0 Normal operation 1 MSCAN in initialization mode

[1] The MSCAN must be in normal mode for this bit to become set.

[2] See the Bosch CAN 2.0A/B specification for a detailed definition of transmitter and receiver states.

[3] In order to protect from accidentally violating the CAN protocol, the TXCAN pin is immediately forced to a recessive state when the CPU enters wait (CSWAI = 1) or stop mode (see Section 15.5.5.2, "Operation in Wait Mode" and Section 15.5.5.3, "Operation in Stop Mode").

[4] The CPU has to make sure that the WUPE bit and the WUPIE wake-up interrupt enable bit (see Section 15.3.5, "MSCAN Receiver Interrupt Enable Register (CANRIER)) is enabled, if the recovery mechanism from stop or wait is required.

[5] The CPU cannot clear SLPRQ before the MSCAN has entered sleep mode (SLPRQ = 1 and SLPAK = 1).

[6] The CPU cannot clear INITRQ before the MSCAN has entered initialization mode (INITRQ = 1 and INITAK = 1).

[7] In order to protect from accidentally violating the CAN protocol, the TXCAN pin is immediately forced to a recessive state when the initialization mode is requested by the CPU. Thus, the recommended procedure is to bring the MSCAN into sleep mode (SLPRQ = 1 and SLPAK = 1) before requesting initialization mode.

[8] Not including WUPE, INITRQ, and SLPRQ.

[9] TSTAT1 and TSTAT0 are not affected by initialization mode.

[10] RSTAT1 and RSTAT0 are not affected by initialization mode.

15.3.2 MSCAN Control Register 1 (CANCTL1)

The CANCTL1 register provides various control bits and handshake status information of the MSCAN module as described below.

	7	6	5	4	3	2	1	0
R	CANE	CLKSRC	LOOPB	LISTEN	BORM	WUPM	SLPAK	INITAK
W							▨	▨
Reset:	0	0	0	1	0	0	0	1

▨ = Unimplemented

Figure 15-4. MSCAN Control Register 1(CANCTL1)

Read: Anytime
Write: Anytime when INITRQ = 1 and INITAK = 1, except CANE which is write once in normal and anytime in special system operation modes when the MSCAN is in initialization mode (INITRQ = 1 and INITAK = 1).

Table 15-2. CANCTL1 Register Field Descriptions

Field	Description
7 CANE	**MSCAN Enable** 0 MSCAN module is disabled 1 MSCAN module is enabled
6 CLKSRC	**MSCAN Clock Source** — This bit defines the clock source for the MSCAN module (only for systems with a clock generation module; Section 15.5.3.3, "Clock System," and Section Figure 15-41., "MSCAN Clocking Scheme,"). 0 MSCAN clock source is the oscillator clock 1 MSCAN clock source is the bus clock
5 LOOPB	**Loopback Self Test Mode** — When this bit is set, the MSCAN performs an internal loopback which can be used for self test operation. The bit stream output of the transmitter is fed back to the receiver internally.Section 15.5.4.6, "Loopback Self Test Mode. 0 Loopback self test disabled 1 Loopback self test enabled
4 LISTEN	**Listen Only Mode** — This bit configures the MSCAN as a CAN bus monitor. When LISTEN is set, all valid CAN messages with matching ID are received, but no acknowledgement or error frames are sent out (see Section 15.5.4.4, "Listen-Only Mode"). In addition, the error counters are frozen. Listen only mode supports applications which require "hot plugging" or throughput analysis. The MSCAN is unable to transmit any messages when listen only mode is active. 0 Normal operation 1 Listen only mode activated
3 BORM	**Bus-Off Recovery Mode** — This bits configures the bus-off state recovery mode of the MSCAN. Refer to Section 15.6.2, "Bus-Off Recovery," for details. 0 Automatic bus-off recovery (see Bosch CAN 2.0A/B protocol specification) 1 Bus-off recovery upon user request
2 WUPM	**Wake-Up Mode** — If WUPE in CANCTL0 is enabled, this bit defines whether the integrated low-pass filter is applied to protect the MSCAN from spurious wake-up (see Section 15.5.5.4, "MSCAN Sleep Mode"). 0 MSCAN wakes up on any dominant level on the CAN bus 1 MSCAN wakes up only in case of a dominant pulse on the CAN bus that has a length of T_{wup}

Table 15-2. CANCTL1 Register Field Descriptions (continued)

Field	Description
1 SLPAK	**Sleep Mode Acknowledge** — This flag indicates whether the MSCAN module has entered sleep mode (see Section 15.5.5.4, “MSCAN Sleep Mode”). It is used as a handshake flag for the SLPRQ sleep mode request. Sleep mode is active when SLPRQ = 1 and SLPAK = 1. Depending on the setting of WUPE, the MSCAN will clear the flag if it detects activity on the CAN bus while in sleep mode.CPU clearing the SLPRQ bit will also reset the SLPAK bit. 0 Running — The MSCAN operates normally 1 Sleep mode active — The MSCAN has entered sleep mode
0 INITAK	**Initialization Mode Acknowledge** — This flag indicates whether the MSCAN module is in initialization mode (see Section 15.5.5.5, “MSCAN Initialization Mode”). It is used as a handshake flag for the INITRQ initialization mode request. Initialization mode is active when INITRQ = 1 and INITAK = 1. The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0–CANIDAR7, and CANIDMR0–CANIDMR7 can be written only by the CPU when the MSCAN is in initialization mode. 0 Running — The MSCAN operates normally 1 Initialization mode active — The MSCAN is in initialization mode

15.3.3 MSCAN Bus Timing Register 0 (CANBTR0)

The CANBTR0 register configures various CAN bus timing parameters of the MSCAN module.

	7	6	5	4	3	2	1	0
R W	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Reset:	0	0	0	0	0	0	0	0

Figure 15-5. MSCAN Bus Timing Register 0 (CANBTR0)

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-3. CANBTR0 Register Field Descriptions

Field	Description
7:6 SJW[1:0]	**Synchronization Jump Width** — The synchronization jump width defines the maximum number of time quanta (Tq) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the CAN bus (see Table 15-4).
5:0 BRP[5:0]	**Baud Rate Prescaler** — These bits determine the time quanta (Tq) clock which is used to build up the bit timing (see Table 15-5).

Table 15-4. Synchronization Jump Width

SJW1	SJW0	Synchronization Jump Width
0	0	1 Tq clock cycle
0	1	2 Tq clock cycles
1	0	3 Tq clock cycles
1	1	4 Tq clock cycles

Table 15-5. Baud Rate Prescaler

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Prescaler value (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
:	:	:	:	:	:	:
1	1	1	1	1	1	64

15.3.4 MSCAN Bus Timing Register 1 (CANBTR1)

The CANBTR1 register configures various CAN bus timing parameters of the MSCAN module.

	7	6	5	4	3	2	1	0
R W	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
Reset:	0	0	0	0	0	0	0	0

Figure 15-6. MSCAN Bus Timing Register 1 (CANBTR1)

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-6. CANBTR1 Register Field Descriptions

Field	Description
7 SAMP	**Sampling** — This bit determines the number of CAN bus samples taken per bit time. 0 One sample per bit. 1 Three samples per bit[1]. If SAMP = 0, the resulting bit value is equal to the value of the single bit positioned at the sample point. If SAMP = 1, the resulting bit value is determined by using majority rule on the three total samples. For higher bit rates, it is recommended that only one sample is taken per bit time (SAMP = 0).
6:4 TSEG2[2:0]	**Time Segment 2** — Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point (see Figure 15-42). Time segment 2 (TSEG2) values are programmable as shown in Table 15-7.
3:0 TSEG1[3:0]	**Time Segment 1** — Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point (see Figure 15-42). Time segment 1 (TSEG1) values are programmable as shown in Table 15-8.

[1] In this case, PHASE_SEG1 must be at least 2 time quanta (Tq).

Table 15-7. Time Segment 2 Values

TSEG22	TSEG21	TSEG20	Time Segment 2
0	0	0	1 Tq clock cycle[1]
0	0	1	2 Tq clock cycles
:	:	:	:
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

[1] This setting is not valid. Please refer to Table 15-35 for valid settings.

Table 15-8. Time Segment 1 Values

TSEG13	TSEG12	TSEG11	TSEG10	Time segment 1
0	0	0	0	1 Tq clock cycle[1]
0	0	0	1	2 Tq clock cycles[1]
0	0	1	0	3 Tq clock cycles[1]
0	0	1	1	4 Tq clock cycles
:	:	:	:	:
1	1	1	0	15 Tq clock cycles
1	1	1	1	16 Tq clock cycles

[1] This setting is not valid. Please refer to Table 15-35 for valid settings.

The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (Tq) clock cycles per bit (as shown in Table 15-7 and Table 15-8).

$$\text{Bit Time} = \frac{(\text{Prescaler value})}{f_{\text{CANCLK}}} \bullet (1 + \text{TimeSegment1} + \text{TimeSegment2}) \qquad \textit{Eqn. 15-1}$$

15.3.4.1 MSCAN Receiver Flag Register (CANRFLG)

A flag can be cleared only by software (writing a 1 to the corresponding bit position) when the condition which caused the setting is no longer valid. Every flag has an associated interrupt enable bit in the CANRIER register.

	7	6	5	4	3	2	1	0
R	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
W			▨	▨	▨	▨		
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-7. MSCAN Receiver Flag Register (CANRFLG)

NOTE

The CANRFLG register is held in the reset state[1] when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

Read: Anytime
Write: Anytime when out of initialization mode, except RSTAT[1:0] and TSTAT[1:0] flags which are read-only; write of 1 clears flag; write of 0 is ignored.

Table 15-9. CANRFLG Register Field Descriptions

Field	Description
7 WUPIF	**Wake-Up Interrupt Flag** — If the MSCAN detects CAN bus activity while in sleep mode (see Section 15.5.5.4, "MSCAN Sleep Mode,") and WUPE = 1 in CANTCTL0 (see Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)"), the module will set WUPIF. If not masked, a wake-up interrupt is pending while this flag is set. 0 No wake-up activity observed while in sleep mode 1 MSCAN detected activity on the CAN bus and requested wake-up
6 CSCIF	**CAN Status Change Interrupt Flag** — This flag is set when the MSCAN changes its current CAN bus status due to the actual value of the transmit error counter (TEC) and the receive error counter (REC). An additional 4-bit (RSTAT[1:0], TSTAT[1:0]) status register, which is split into separate sections for TEC/REC, informs the system on the actual CAN bus status (see Section 15.3.5, "MSCAN Receiver Interrupt Enable Register (CANRIER)"). If not masked, an error interrupt is pending while this flag is set. CSCIF provides a blocking interrupt. That guarantees that the receiver/transmitter status bits (RSTAT/TSTAT) are only updated when no CAN status change interrupt is pending. If the TECs/RECs change their current value after the CSCIF is asserted, which would cause an additional state change in the RSTAT/TSTAT bits, these bits keep their status until the current CSCIF interrupt is cleared again. 0 No change in CAN bus status occurred since last interrupt 1 MSCAN changed current CAN bus status
5:4 RSTAT[1:0]	**Receiver Status Bits** — The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate receiver related CAN bus status of the MSCAN. The coding for the bits RSTAT1, RSTAT0 is: 00 RxOK: 0 ≤ receive error counter ≤ 96 01 RxWRN: 96 < receive error counter ≤ 127 10 RxERR: 127 < receive error counter 11 Bus-off[1]: transmit error counter > 255
3:2 TSTAT[1:0]	**Transmitter Status Bits** — The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate transmitter related CAN bus status of the MSCAN. The coding for the bits TSTAT1, TSTAT0 is: 00 TxOK: 0 ≤ transmit error counter ≤ 96 01 TxWRN: 96 < transmit error counter ≤ 127 10 TxERR: 127 < transmit error counter ≤ 255 11 Bus-Off: transmit error counter > 255

[1] The RSTAT[1:0], TSTAT[1:0] bits are not affected by initialization mode.

Table 15-9. CANRFLG Register Field Descriptions (continued)

Field	Description
1 OVRIF	**Overrun Interrupt Flag** — This flag is set when a data overrun condition occurs. If not masked, an error interrupt is pending while this flag is set. 0 No data overrun condition 1 A data overrun detected
0 RXF[2]	**Receive Buffer Full Flag** — RXF is set by the MSCAN when a new message is shifted in the receiver FIFO. This flag indicates whether the shifted buffer is loaded with a correctly received message (matching identifier, matching cyclic redundancy code (CRC) and no other errors detected). After the CPU has read that message from the RxFG buffer in the receiver FIFO, the RXF flag must be cleared to release the buffer. A set RXF flag prohibits the shifting of the next FIFO entry into the foreground buffer (RxFG). If not masked, a receive interrupt is pending while this flag is set. 0 No new message available within the RxFG 1 The receiver FIFO is not empty. A new message is available in the RxFG

[1] Redundant Information for the most critical CAN bus status which is “bus-off”. This only occurs if the Tx error counter exceeds a number of 255 errors. Bus-off affects the receiver state. As soon as the transmitter leaves its bus-off state the receiver state skips to RxOK too. Refer also to TSTAT[1:0] coding in this register.

[2] To ensure data integrity, do not read the receive buffer registers while the RXF flag is cleared. For MCUs with dual CPUs, reading the receive buffer registers while the RXF flag is cleared may result in a CPU fault condition.

15.3.5 MSCAN Receiver Interrupt Enable Register (CANRIER)

This register contains the interrupt enable bits for the interrupt flags described in the CANRFLG register.

	7	6	5	4	3	2	1	0
R W	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
Reset:	0	0	0	0	0	0	0	0

Figure 15-8. MSCAN Receiver Interrupt Enable Register (CANRIER)

NOTE

> The CANRIER register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable when not in initialization mode (INITRQ=0 and INITAK=0).
>
> The RSTATE[1:0], TSTATE[1:0] bits are not affected by initialization mode.

Read: Anytime
Write: Anytime when not in initialization mode

Table 15-10. CANRIER Register Field Descriptions

Field	Description
7 WUPIE[1]	**Wake-Up Interrupt Enable** 0 No interrupt request is generated from this event. 1 A wake-up event causes a Wake-Up interrupt request.
6 CSCIE	**CAN Status Change Interrupt Enable** 0 No interrupt request is generated from this event. 1 A CAN Status Change event causes an error interrupt request.
5:4 RSTATE[1:0]	**Receiver Status Change Enable** — These RSTAT enable bits control the sensitivity level in which receiver state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level the RSTAT flags continue to indicate the actual receiver state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by receiver state changes. 01 Generate CSCIF interrupt only if the receiver enters or leaves "bus-off" state. Discard other receiver state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the receiver enters or leaves "RxErr" or "bus-off"[2] state. Discard other receiver state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes.
3:2 TSTATE[1:0]	**Transmitter Status Change Enable** — These TSTAT enable bits control the sensitivity level in which transmitter state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level, the TSTAT flags continue to indicate the actual transmitter state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by transmitter state changes. 01 Generate CSCIF interrupt only if the transmitter enters or leaves "bus-off" state. Discard other transmitter state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the transmitter enters or leaves "TxErr" or "bus-off" state. Discard other transmitter state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes.
1 OVRIE	**Overrun Interrupt Enable** 0 No interrupt request is generated from this event. 1 An overrun event causes an error interrupt request.
0 RXFIE	**Receiver Full Interrupt Enable** 0 No interrupt request is generated from this event. 1 A receive buffer full (successful message reception) event causes a receiver interrupt request.

[1] WUPIE and WUPE (see Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)") must both be enabled if the recovery mechanism from stop or wait is required.

[2] Bus-off state is defined by the CAN standard (see Bosch CAN 2.0A/B protocol specification: for only transmitters. Because the only possible state change for the transmitter from bus-off to TxOK also forces the receiver to skip its current state to RxOK, the coding of the RXSTAT[1:0] flags define an additional bus-off state for the receiver (see Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG)").

15.3.6 MSCAN Transmitter Flag Register (CANTFLG)

The transmit buffer empty flags each have an associated interrupt enable bit in the CANTIER register.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TXE2	TXE1	TXE0
W	▨	▨	▨	▨	▨			
Reset:	0	0	0	0	0	1	1	1

▨ = Unimplemented

Figure 15-9. MSCAN Transmitter Flag Register (CANTFLG)

NOTE

The CANTFLG register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Anytime
Write: Anytime for TXEx flags when not in initialization mode; write of 1 clears flag, write of 0 is ignored

Table 15-11. CANTFLG Register Field Descriptions

Field	Description
2:0 TXE[2:0]	**Transmitter Buffer Empty** — This flag indicates that the associated transmit message buffer is empty, and thus not scheduled for transmission. The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission. The MSCAN sets the flag after the message is sent successfully. The flag is also set by the MSCAN when the transmission request is successfully aborted due to a pending abort request (see Section 15.3.8, "MSCAN Transmitter Message Abort Request Register (CANTARQ)"). If not masked, a transmit interrupt is pending while this flag is set. Clearing a TXEx flag also clears the corresponding ABTAKx (see Section 15.3.9, "MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)"). When a TXEx flag is set, the corresponding ABTRQx bit is cleared (see Section 15.3.8, "MSCAN Transmitter Message Abort Request Register (CANTARQ)"). When listen-mode is active (see Section 15.3.2, "MSCAN Control Register 1 (CANCTL1)") the TXEx flags cannot be cleared and no transmission is started. Read and write accesses to the transmit buffer are blocked, if the corresponding TXEx bit is cleared (TXEx = 0) and the buffer is scheduled for transmission. 0 The associated message buffer is full (loaded with a message due for transmission) 1 The associated message buffer is empty (not scheduled)

15.3.7 MSCAN Transmitter Interrupt Enable Register (CANTIER)

This register contains the interrupt enable bits for the transmit buffer empty interrupt flags.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
W	▨	▨	▨	▨	▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-10. MSCAN Transmitter Interrupt Enable Register (CANTIER)

NOTE

The CANTIER register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Anytime
Write: Anytime when not in initialization mode

Table 15-12. CANTIER Register Field Descriptions

Field	Description
2:0 TXEIE[2:0]	**Transmitter Empty Interrupt Enable** 0 No interrupt request is generated from this event. 1 A transmitter empty (transmit buffer available for transmission) event causes a transmitter empty interrupt request. See Section 15.5.2.2, "Transmit Structures" for details.

15.3.8 MSCAN Transmitter Message Abort Request Register (CANTARQ)

The CANTARQ register allows abort request of messages queued for transmission.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
W	▨	▨	▨	▨	▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-11. MSCAN Transmitter Message Abort Request Register (CANTARQ)

NOTE

The CANTARQ register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Anytime
Write: Anytime when not in initialization mode

Table 15-13. CANTARQ Register Field Descriptions

Field	Description
2:0 ABTRQ[2:0]	**Abort Request** — The CPU sets the ABTRQx bit to request that a scheduled message buffer (TXEx = 0) be aborted. The MSCAN grants the request if the message has not already started transmission, or if the transmission is not successful (lost arbitration or error). When a message is aborted, the associated TXE (see Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)") and abort acknowledge flags (ABTAK, see Section 15.3.9, "MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)") are set and a transmit interrupt occurs if enabled. The CPU cannot reset ABTRQx. ABTRQx is reset whenever the associated TXE flag is set. 0 No abort request 1 Abort request pending

15.3.9 MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)

The CANTAAK register indicates the successful abort of messages queued for transmission, if requested by the appropriate bits in the CANTARQ register.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-12. MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)

NOTE

> The CANTAAK register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1).

Read: Anytime
Write: Unimplemented for ABTAKx flags

Table 15-14. CANTAAK Register Field Descriptions

Field	Description
2:0 ABTAK[2:0]	**Abort Acknowledge** — This flag acknowledges that a message was aborted due to a pending transmission abort request from the CPU. After a particular message buffer is flagged empty, this flag can be used by the application software to identify whether the message was aborted successfully or was sent anyway. The ABTAKx flag is cleared whenever the corresponding TXE flag is cleared. 0 The message was not aborted. 1 The message was aborted.

15.3.10 MSCAN Transmit Buffer Selection Register (CANTBSEL)

The CANTBSEL selections of the actual transmit message buffer, which is accessible in the CANTXFG register space.

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	TX2	TX1	TX0
W	▨	▨	▨	▨	▨			
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-13. MSCAN Transmit Buffer Selection Register (CANTBSEL)

NOTE

The CANTBSEL register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK=1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Find the lowest ordered bit set to 1, all other bits will be read as 0
Write: Anytime when not in initialization mode

Table 15-15. CANTBSEL Register Field Descriptions

Field	Description
2:0 TX[2:0]	**Transmit Buffer Select** — The lowest numbered bit places the respective transmit buffer in the CANTXFG register space (e.g., TX1 = 1 and TX0 = 1 selects transmit buffer TX0; TX1 = 1 and TX0 = 0 selects transmit buffer TX1). Read and write accesses to the selected transmit buffer will be blocked, if the corresponding TXEx bit is cleared and the buffer is scheduled for transmission (see Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)"). 0 The associated message buffer is deselected 1 The associated message buffer is selected, if lowest numbered bit

The following gives a short programming example of the usage of the CANTBSEL register:

To get the next available transmit buffer, application software must read the CANTFLG register and write this value back into the CANTBSEL register. In this example Tx buffers TX1 and TX2 are available. The value read from CANTFLG is therefore 0b0000_0110. When writing this value back to CANTBSEL, the Tx buffer TX1 is selected in the CANTXFG because the lowest numbered bit set to 1 is at bit position 1. Reading back this value out of CANTBSEL results in 0b0000_0010, because only the lowest numbered bit position set to 1 is presented. This mechanism eases the application software the selection of the next available Tx buffer.

- LDD CANTFLG; value read is 0b0000_0110
- STD CANTBSEL; value written is 0b0000_0110
- LDD CANTBSEL; value read is 0b0000_0010

If all transmit message buffers are deselected, no accesses are allowed to the CANTXFG buffer register.

15.3.11 MSCAN Identifier Acceptance Control Register (CANIDAC)

The CANIDAC register is used for identifier filter acceptance control as described below.

	7	6	5	4	3	2	1	0
R	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
W	▨	▨			▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-14. MSCAN Identifier Acceptance Control Register (CANIDAC)

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1), except bits IDHITx, which are read-only

Table 15-16. CANIDAC Register Field Descriptions

Field	Description
5:4 IDAM[1:0]	**Identifier Acceptance Mode** — The CPU sets these flags to define the identifier acceptance filter organization (see Section 15.5.3, "Identifier Acceptance Filter"). Table 15-17 summarizes the different settings. In filter closed mode, no message is accepted such that the foreground buffer is never reloaded.
2:0 IDHIT[2:0]	Identifier Acceptance Hit Indicator — The MSCAN sets these flags to indicate an identifier acceptance hit (see Section 15.5.3, "Identifier Acceptance Filter"). Table 15-18 summarizes the different settings.

Table 15-17. Identifier Acceptance Mode Settings

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	Two 32-bit acceptance filters
0	1	Four 16-bit acceptance filters
1	0	Eight 8-bit acceptance filters
1	1	Filter closed

Table 15-18. Identifier Acceptance Hit Indication

IDHIT2	IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	0	Filter 0 hit
0	0	1	Filter 1 hit
0	1	0	Filter 2 hit
0	1	1	Filter 3 hit
1	0	0	Filter 4 hit
1	0	1	Filter 5 hit
1	1	0	Filter 6 hit
1	1	1	Filter 7 hit

The IDHITx indicators are always related to the message in the foreground buffer (RxFG). When a message gets shifted into the foreground buffer of the receiver FIFO the indicators are updated as well.

15.3.12 MSCAN Miscellaneous Register (CANMISC)

This register provides additional features.

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

▨ = Unimplemented

Figure 15-15. MSCAN Miscellaneous Register (CANMISC)

Read: Anytime
Write: Anytime; write of '1' clears flag; write of '0' ignored

Table 15-19. CANMISC Register Field Descriptions

Field	Description
0 BOHOLD	**Bus-off State Hold Until User Request** — If BORM is set in Section 15.3.2, "MSCAN Control Register 1 (CANCTL1), this bit indicates whether the module has entered the bus-off state. Clearing this bit requests the recovery from bus-off. Refer to Section 15.6.2, "Bus-Off Recovery," for details. 0 Module is not bus-off or recovery has been requested by user in bus-off state 1 Module is bus-off and holds this state until user request

15.3.13 MSCAN Receive Error Counter (CANRXERR)

This register reflects the status of the MSCAN receive error counter.

	7	6	5	4	3	2	1	0
R	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-16. MSCAN Receive Error Counter (CANRXERR)

Read: Only when in sleep mode (SLPRQ = 1 and SLPAK = 1) or initialization mode (INITRQ = 1 and INITAK = 1)

Write: Unimplemented

NOTE

Reading this register when in any other mode other than sleep or initialization mode may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.

Writing to this register when in special modes can alter the MSCAN functionality.

15.3.14 MSCAN Transmit Error Counter (CANTXERR)

This register reflects the status of the MSCAN transmit error counter.

	7	6	5	4	3	2	1	0
R	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

▨ = Unimplemented

Figure 15-17. MSCAN Transmit Error Counter (CANTXERR)

Read: Only when in sleep mode (SLPRQ = 1 and SLPAK = 1) or initialization mode (INITRQ = 1 and INITAK = 1)

Write: Unimplemented

NOTE

> Reading this register when in any other mode other than sleep or initialization mode, may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.
>
> Writing to this register when in special modes can alter the MSCAN functionality.

15.3.15 MSCAN Identifier Acceptance Registers (CANIDAR0-7)

On reception, each message is written into the background receive buffer. The CPU is only signalled to read the message if it passes the criteria in the identifier acceptance and identifier mask registers (accepted); otherwise, the message is overwritten by the next message (dropped).

The acceptance registers of the MSCAN are applied on the IDR0–IDR3 registers (see Section 15.4.1, "Identifier Registers (IDR0–IDR3)") of incoming messages in a bit by bit manner (see Section 15.5.3, "Identifier Acceptance Filter").

For extended identifiers, all four acceptance and mask registers are applied. For standard identifiers, only the first two (CANIDAR0/1, CANIDMR0/1) are applied.

	7	6	5	4	3	2	1	0
R W	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset	0	0	0	0	0	0	0	0

Figure 15-18. MSCAN Identifier Acceptance Registers (First Bank) — CANIDAR0–CANIDAR3

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-20. CANIDAR0–CANIDAR3 Register Field Descriptions

Field	Description
7:0 AC[7:0]	**Acceptance Code Bits** — AC[7:0] comprise a user-defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is then masked with the corresponding identifier mask register.

	7	6	5	4	3	2	1	0
R W	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Reset	0	0	0	0	0	0	0	0

Figure 15-19. MSCAN Identifier Acceptance Registers (Second Bank) — CANIDAR4–CANIDAR7

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-21. CANIDAR4–CANIDAR7 Register Field Descriptions

Field	Description
7:0 AC[7:0]	**Acceptance Code Bits** — AC[7:0] comprise a user-defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is then masked with the corresponding identifier mask register.

15.3.16 MSCAN Identifier Mask Registers (CANIDMR0–CANIDMR7)

The identifier mask register specifies which of the corresponding bits in the identifier acceptance register are relevant for acceptance filtering. To receive standard identifiers in 32 bit filter mode, it is required to program the last three bits (AM[2:0]) in the mask registers CANIDMR1 and CANIDMR5 to "don't care." To receive standard identifiers in 16 bit filter mode, it is required to program the last three bits (AM[2:0]) in the mask registers CANIDMR1, CANIDMR3, CANIDMR5, and CANIDMR7 to "don't care."

	7	6	5	4	3	2	1	0
R W	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset	0	0	0	0	0	0	0	0

Figure 15-20. MSCAN Identifier Mask Registers (First Bank) — CANIDMR0–CANIDMR3

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-22. CANIDMR0–CANIDMR3 Register Field Descriptions

Field	Description
7:0 AM[7:0]	**Acceptance Mask Bits** — If a particular bit in this register is cleared, this indicates that the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates that the state of the corresponding bit in the identifier acceptance register does not affect whether or not the message is accepted. 0 Match corresponding acceptance code register and identifier bits 1 Ignore corresponding acceptance code register bit (don't care)

	7	6	5	4	3	2	1	0
R W	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Reset	0	0	0	0	0	0	0	0

Figure 15-21. MSCAN Identifier Mask Registers (Second Bank) — CANIDMR4–CANIDMR7

Read: Anytime
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

Table 15-23. CANIDMR4–CANIDMR7 Register Field Descriptions

Field	Description
7:0 AM[7:0]	**Acceptance Mask Bits** — If a particular bit in this register is cleared, this indicates that the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates that the state of the corresponding bit in the identifier acceptance register does not affect whether or not the message is accepted. 0 Match corresponding acceptance code register and identifier bits 1 Ignore corresponding acceptance code register bit (don't care)

15.4 Programmer's Model of Message Storage

The following section details the organization of the receive and transmit message buffers and the associated control registers.

To simplify the programmer interface, the receive and transmit message buffers have the same outline. Each message buffer allocates 16 bytes in the memory map containing a 13 byte data structure.

An additional transmit buffer priority register (TBPR) is defined for the transmit buffers. Within the last two bytes of this memory map, the MSCAN stores a special 16-bit time stamp, which is sampled from an internal timer after successful transmission or reception of a message. This feature is only available for transmit and receiver buffers if the TIME bit is set (see Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)").

The time stamp register is written by the MSCAN. The CPU can only read these registers.

Table 15-24. Message Buffer Organization

Offset Address	Register
0x00X0	Identifier Register 0
0x00X1	Identifier Register 1
0x00X2	Identifier Register 2
0x00X3	Identifier Register 3
0x00X4	Data Segment Register 0
0x00X5	Data Segment Register 1
0x00X6	Data Segment Register 2
0x00X7	Data Segment Register 3
0x00X8	Data Segment Register 4
0x00X9	Data Segment Register 5
0x00XA	Data Segment Register 6
0x00XB	Data Segment Register 7
0x00XC	Data Length Register
0x00XD	Transmit Buffer Priority Register[1]
0x00XE	Time Stamp Register (High Byte)[2]
0x00XF	Time Stamp Register (Low Byte)[3]

[1] Not applicable for receive buffers

[2] Read-only for CPU

[3] Read-only for CPU

Figure 15-22 shows the common 13-byte data structure of receive and transmit buffers for extended identifiers. The mapping of standard identifiers into the IDR registers is shown in Figure 15-23.

All bits of the receive and transmit buffers are 'x' out of reset because of RAM-based implementation[1]. All reserved or unused bits of the receive and transmit buffers always read 'x'.

[1] Exception: The transmit priority registers are 0 out of reset.

Figure 15-22. Receive/Transmit Message Buffer — Extended Identifier Mapping

Register Name		Bit 7	6	5	4	3	2	1	Bit0
IDR0	R	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	W								
IDR1	R	ID20	ID19	ID18	SRR[1]	IDE[1]	ID17	ID16	ID15
	W								
IDR2	R	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	W								
IDR3	R	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR[2]
	W								
DSR0	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR1	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR2	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR3	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR4	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR5	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR6	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DSR7	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	W								
DLR	R	▨	▨	▨	▨	DLC3	DLC2	DLC1	DLC0
	W								

▨ = Unused, always read ‘x’

[1] SRR and IDE are both 1s.

[2] The position of RTR differs between extended and standard identifier mapping.

Read: For transmit buffers, anytime when TXEx flag is set (see Section 15.3.6, “MSCAN Transmitter Flag Register (CANTFLG)”) and the corresponding transmit buffer is selected in CANTBSEL (see Section 15.3.10, “MSCAN Transmit Buffer Selection Register (CANTBSEL)”). For receive buffers, only when RXF flag is set (see Section 15.3.4.1, “MSCAN Receiver Flag Register (CANRFLG)”).

Write: For transmit buffers, anytime when TXEx flag is set (see Section 15.3.6, “MSCAN Transmitter Flag Register (CANTFLG)”) and the corresponding transmit buffer is selected in CANTBSEL (see Section 15.3.10, “MSCAN Transmit Buffer Selection Register (CANTBSEL)”). Unimplemented for receive buffers.

Reset: Undefined (0x00XX) because of RAM-based implementation

Figure 15-23. Receive/Transmit Message Buffer — Standard Identifier Mapping

Register Name		Bit 7	6	5	4	3	2	1	Bit 0
IDR0	R W	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
IDR1	R W	ID2	ID1	ID0	RTR[1]	IDE[2]	▨	▨	▨
IDR2	R W	▨	▨	▨	▨	▨	▨	▨	▨
IDR3	R W	▨	▨	▨	▨	▨	▨	▨	▨

▨ = Unused, always read ‘x’

[1] The position of RTR differs between extended and standard identifier mapping.

[2] IDE is 0.

15.4.1 Identifier Registers (IDR0–IDR3)

The identifier registers for an extended format identifier consist of a total of 32 bits; ID[28:0], SRR, IDE, and RTR bits. The identifier registers for a standard format identifier consist of a total of 13 bits; ID[10:0], RTR, and IDE bits.

15.4.1.1 IDR0–IDR3 for Extended Identifier Mapping

	7	6	5	4	3	2	1	0
R W	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
Reset:	x	x	x	x	x	x	x	x

Figure 15-24. Identifier Register 0 (IDR0) — Extended Identifier Mapping

Table 15-25. IDR0 Register Field Descriptions — Extended

Field	Description
7:0 ID[28:21]	**Extended Format Identifier** — The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

	7	6	5	4	3	2	1	0
R W	ID20	ID19	ID18	SRR[1]	IDE[1]	ID17	ID16	ID15
Reset:	x	x	x	x	x	x	x	x

Figure 15-25. Identifier Register 1 (IDR1) — Extended Identifier Mapping

[1] SRR and IDE are both 1s.

Table 15-26. IDR1 Register Field Descriptions — Extended

Field	Description
7:5 ID[20:18]	**Extended Format Identifier —** The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.
4 SRR	**Substitute Remote Request** — This fixed recessive bit is used only in extended format. It must be set to 1 by the user for transmission buffers and is stored as received on the CAN bus for receive buffers.
3 IDE	**ID Extended** — This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the CPU how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)
2:0 ID[17:15]	**Extended Format Identifier —** The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

	7	6	5	4	3	2	1	0
R W	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
Reset:	x	x	x	x	x	x	x	x

Figure 15-26. Identifier Register 2 (IDR2) — Extended Identifier Mapping

Table 15-27. IDR2 Register Field Descriptions — Extended

Field	Description
7:0 ID[14:7]	**Extended Format Identifier** — The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

	7	6	5	4	3	2	1	0
R W	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
Reset:	x	x	x	x	x	x	x	x

Figure 15-27. Identifier Register 3 (IDR3) — Extended Identifier Mapping

Table 15-28. IDR3 Register Field Descriptions — Extended

Field	Description
7:1 ID[6:0]	**Extended Format Identifier** — The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.
0 RTR	**Remote Transmission Request** — This flag reflects the status of the remote transmission request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame

15.4.2 IDR0–IDR3 for Standard Identifier Mapping

	7	6	5	4	3	2	1	0
R W	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
Reset:	x	x	x	x	x	x	x	x

Figure 15-28. Identifier Register 0 — Standard Mapping

Table 15-29. IDR0 Register Field Descriptions — Standard

Field	Description
7:0 ID[10:3]	**Standard Format Identifier** — The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in Table 15-30.

	7	6	5	4	3	2	1	0
R W	ID2	ID1	ID0	RTR	IDE[1]	▨	▨	▨
Reset:	x	x	x	x	x	x	x	x

▨ = Unused; always read 'x'

Figure 15-29. Identifier Register 1 — Standard Mapping

[1] IDE is 0.

Table 15-30. IDR1 Register Field Descriptions

Field	Description
7:5 ID[2:0]	**Standard Format Identifier —** The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in Table 15-29.
4 RTR	**Remote Transmission Request** — This flag reflects the status of the Remote Transmission Request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame
3 IDE	**ID Extended** — This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the CPU how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)

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

▨ = Unused; always read 'x'

Figure 15-30. Identifier Register 2 — Standard Mapping

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

▨ = Unused; always read 'x'

Figure 15-31. Identifier Register 3 — Standard Mapping

15.4.3 Data Segment Registers (DSR0-7)

The eight data segment registers, each with bits DB[7:0], contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the corresponding DLR register.

	7	6	5	4	3	2	1	0
R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Reset:	x	x	x	x	x	x	x	x

Figure 15-32. Data Segment Registers (DSR0–DSR7) — Extended Identifier Mapping

Table 15-31. DSR0–DSR7 Register Field Descriptions

Field	Description
7:0 DB[7:0]	**Data bits 7:0**

15.4.4 Data Length Register (DLR)

This register keeps the data length field of the CAN frame.

	7	6	5	4	3	2	1	0
R W	▨	▨	▨	▨	DLC3	DLC2	DLC1	DLC0
Reset:	x	x	x	x	x	x	x	x

▨ = Unused; always read “x”

Figure 15-33. Data Length Register (DLR) — Extended Identifier Mapping

Table 15-32. DLR Register Field Descriptions

Field	Description
3:0 DLC[3:0]	**Data Length Code Bits** — The data length code contains the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0. The data byte count ranges from 0 to 8 for a data frame. Table 15-33 shows the effect of setting the DLC bits.

Table 15-33. Data Length Codes

Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

15.4.5 Transmit Buffer Priority Register (TBPR)

This register defines the local priority of the associated message transmit buffer. The local priority is used for the internal prioritization process of the MSCAN and is defined to be highest for the smallest binary number. The MSCAN implements the following internal prioritization mechanisms:

- All transmission buffers with a cleared TXEx flag participate in the prioritization immediately before the SOF (start of frame) is sent.
- The transmission buffer with the lowest local priority field wins the prioritization.

In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins.

	7	6	5	4	3	2	1	0
R W	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
Reset:	0	0	0	0	0	0	0	0

Figure 15-34. Transmit Buffer Priority Register (TBPR)

Read: Anytime when TXEx flag is set (see Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)") and the corresponding transmit buffer is selected in CANTBSEL (see Section 15.3.10, "MSCAN Transmit Buffer Selection Register (CANTBSEL)").

Write: Anytime when TXEx flag is set (see Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)") and the corresponding transmit buffer is selected in CANTBSEL (see Section 15.3.10, "MSCAN Transmit Buffer Selection Register (CANTBSEL)").

15.4.6 Time Stamp Register (TSRH–TSRL)

If the TIME bit is enabled, the MSCAN will write a time stamp to the respective registers in the active transmit or receive buffer as soon as a message has been acknowledged on the CAN bus (see

Section 15.3.1, “MSCAN Control Register 0 (CANCTL0)”). In case of a transmission, the CPU can only read the time stamp after the respective transmit buffer has been flagged empty.

The timer value, which is used for stamping, is taken from a free running internal CAN bit clock. A timer overrun is not indicated by the MSCAN. The timer is reset (all bits set to 0) during initialization mode. The CPU can only read the time stamp registers.

	7	6	5	4	3	2	1	0
R	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
W								
Reset:	x	x	x	x	x	x	x	x

Figure 15-35. Time Stamp Register — High Byte (TSRH)

	7	6	5	4	3	2	1	0
R	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
W								
Reset:	x	x	x	x	x	x	x	x

Figure 15-36. Time Stamp Register — Low Byte (TSRL)

Read: Anytime when TXEx flag is set (see Section 15.3.6, “MSCAN Transmitter Flag Register (CANTFLG)”) and the corresponding transmit buffer is selected in CANTBSEL (see Section 15.3.10, “MSCAN Transmit Buffer Selection Register (CANTBSEL)”).

Write: Unimplemented

15.5 Functional Description

15.5.1 General

This section provides a complete functional description of the MSCAN. It describes each of the features and modes listed in the introduction.

15.5.2 Message Storage

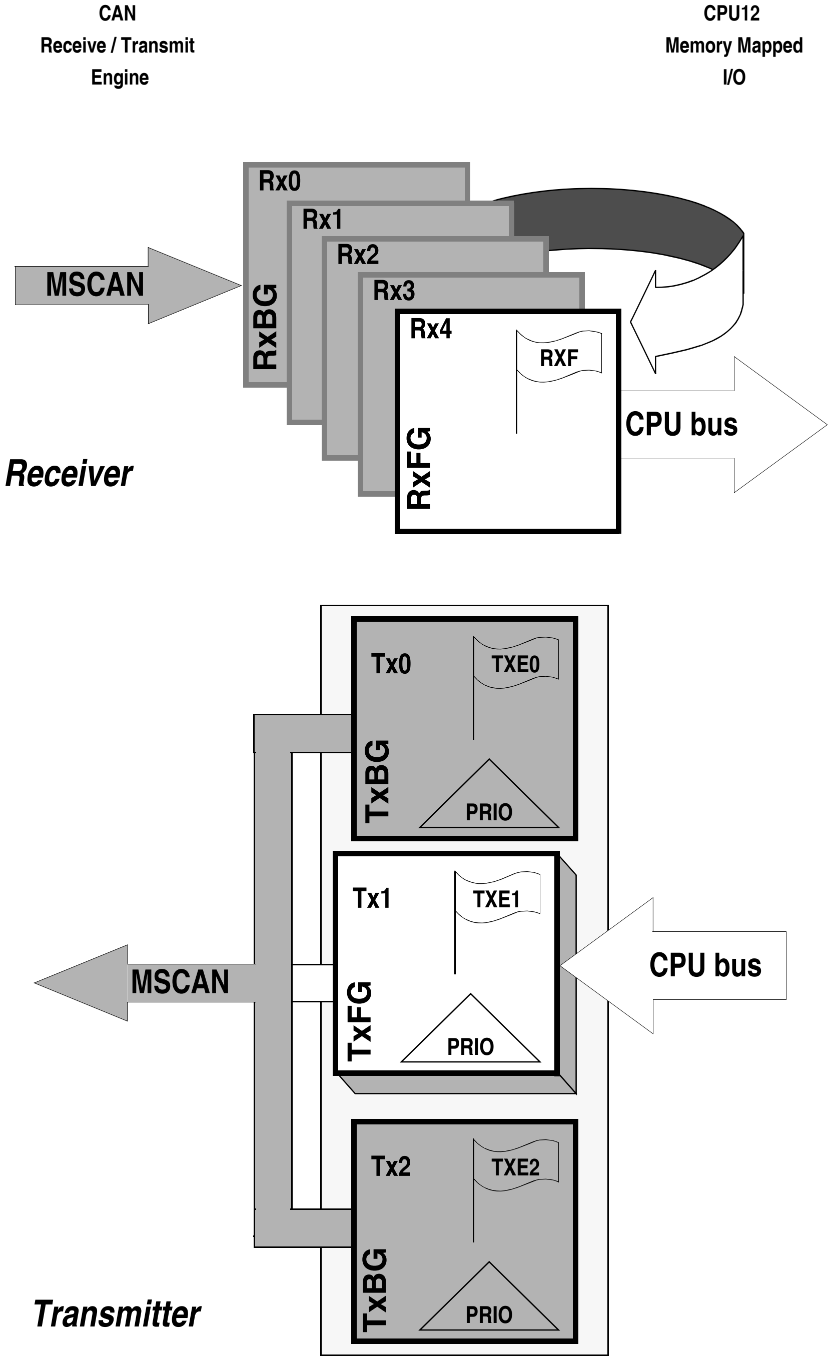


Figure 15-37. User Model for Message Buffer Organization

MSCAN facilitates a sophisticated message storage system which addresses the requirements of a broad range of network applications.

15.5.2.1 Message Transmit Background

Modern application layer software is built upon two fundamental assumptions:

- Any CAN node is able to send out a stream of scheduled messages without releasing the CAN bus between the two messages. Such nodes arbitrate for the CAN bus immediately after sending the previous message and only release the CAN bus in case of lost arbitration.
- The internal message queue within any CAN node is organized such that the highest priority message is sent out first, if more than one message is ready to be sent.

The behavior described in the bullets above cannot be achieved with a single transmit buffer. That buffer must be reloaded immediately after the previous message is sent. This loading process lasts a finite amount of time and must be completed within the inter-frame sequence (IFS) to be able to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires that the CPU reacts with short latencies to the transmit interrupt.

A double buffer scheme de-couples the reloading of the transmit buffer from the actual message sending and, therefore, reduces the reactiveness requirements of the CPU. Problems can arise if the sending of a message is finished while the CPU re-loads the second buffer. No buffer would then be ready for transmission, and the CAN bus would be released.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The MSCAN has three transmit buffers.

The second requirement calls for some sort of internal prioritization which the MSCAN implements with the "local priority" concept described in Section 15.5.2.2, "Transmit Structures."

15.5.2.2 Transmit Structures

The MSCAN triple transmit buffer scheme optimizes real-time performance by allowing multiple messages to be set up in advance. The three buffers are arranged as shown in Figure 15-37.

All three buffers have a 13-byte data structure similar to the outline of the receive buffers (see Section 15.4, "Programmer's Model of Message Storage"). An additional Section 15.4.5, "Transmit Buffer Priority Register (TBPR) contains an 8-bit local priority field (PRIO) (see Section 15.4.5, "Transmit Buffer Priority Register (TBPR)"). The remaining two bytes are used for time stamping of a message, if required (see Section 15.4.6, "Time Stamp Register (TSRH–TSRL)").

To transmit a message, the CPU must identify an available transmit buffer, which is indicated by a set transmitter buffer empty (TXEx) flag (see Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)"). If a transmit buffer is available, the CPU must set a pointer to this buffer by writing to the CANTBSEL register (see Section 15.3.10, "MSCAN Transmit Buffer Selection Register (CANTBSEL)"). This makes the respective buffer accessible within the CANTXFG address space (see Section 15.4, "Programmer's Model of Message Storage"). The algorithmic feature associated with the CANTBSEL register simplifies the transmit buffer selection. In addition, this scheme makes the handler software simpler because only one address area is applicable for the transmit process, and the required address space is minimized.

The CPU then stores the identifier, the control bits, and the data content into one of the transmit buffers. Finally, the buffer is flagged as ready for transmission by clearing the associated TXE flag.

The MSCAN then schedules the message for transmission and signals the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt (see Section 15.5.7.2, "Transmit Interrupt") is generated[1] when TXEx is set and can be used to drive the application software to re-load the buffer.

If more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the MSCAN uses the local priority setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit local priority field (PRIO). The application software programs this field when the message is set up. The local priority reflects the priority of this particular message relative to the set of messages being transmitted from this node. The lowest binary value of the PRIO field is defined to be the highest priority. The internal scheduling process takes place whenever the MSCAN arbitrates for the CAN bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message in one of the three transmit buffers. Because messages that are already in transmission cannot be aborted, the user must request the abort by setting the corresponding abort request bit (ABTRQ) (see Section 15.3.8, "MSCAN Transmitter Message Abort Request Register (CANTARQ)".) The MSCAN then grants the request, if possible, by:

1. Setting the corresponding abort acknowledge flag (ABTAK) in the CANTAAK register.
2. Setting the associated TXE flag to release the buffer.
3. Generating a transmit interrupt. The transmit interrupt handler software can determine from the setting of the ABTAK flag whether the message was aborted (ABTAK = 1) or sent (ABTAK = 0).

15.5.2.3 Receive Structures

The received messages are stored in a five stage input FIFO. The five message buffers are alternately mapped into a single memory area (see Figure 15-37). The background receive buffer (RxBG) is exclusively associated with the MSCAN, but the foreground receive buffer (RxFG) is addressable by the CPU (see Figure 15-37). This scheme simplifies the handler software because only one address area is applicable for the receive process.

All receive buffers have a size of 15 bytes to store the CAN control bits, the identifier (standard or extended), the data contents, and a time stamp, if enabled (see Section 15.4, "Programmer's Model of Message Storage").

The receiver full flag (RXF) (see Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG)") signals the status of the foreground receive buffer. When the buffer contains a correctly received message with a matching identifier, this flag is set.

On reception, each message is checked to see whether it passes the filter (see Section 15.5.3, "Identifier Acceptance Filter") and simultaneously is written into the active RxBG. After successful reception of a valid message, the MSCAN shifts the content of RxBG into the receiver FIFO[2], sets the RXF flag, and generates a receive interrupt (see Section 15.5.7.3, "Receive Interrupt") to the CPU[3]. The user's receive handler must read the received message from the RxFG and then reset the RXF flag to acknowledge the interrupt and to release the foreground buffer. A new message, which can follow immediately after the IFS

[1] The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.

[2] Only if the RXF flag is not set.

[3] The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.

field of the CAN frame, is received into the next available RxBG. If the MSCAN receives an invalid message in its RxBG (wrong identifier, transmission errors, etc.) the actual contents of the buffer will be over-written by the next message. The buffer will then not be shifted into the FIFO.

When the MSCAN module is transmitting, the MSCAN receives its own transmitted messages into the background receive buffer, RxBG, but does not shift it into the receiver FIFO, generate a receive interrupt, or acknowledge its own messages on the CAN bus. The exception to this rule is in loopback mode (see Section 15.3.2, "MSCAN Control Register 1 (CANCTL1)") where the MSCAN treats its own messages exactly like all other incoming messages. The MSCAN receives its own transmitted messages in the event that it loses arbitration. If arbitration is lost, the MSCAN must be prepared to become a receiver.

An overrun condition occurs when all receive message buffers in the FIFO are filled with correctly received messages with accepted identifiers and another message is correctly received from the CAN bus with an accepted identifier. The latter message is discarded and an error interrupt with overrun indication is generated if enabled (see Section 15.5.7.5, "Error Interrupt"). The MSCAN remains able to transmit messages while the receiver FIFO is full, but all incoming messages are discarded. As soon as a receive buffer in the FIFO is available again, new valid messages will be accepted.

15.5.3 Identifier Acceptance Filter

The MSCAN identifier acceptance registers (see Section 15.3.11, "MSCAN Identifier Acceptance Control Register (CANIDAC)") define the acceptable patterns of the standard or extended identifier (ID[10:0] or ID[28:0]). Any of these bits can be marked 'don't care' in the MSCAN identifier mask registers (see Section 15.3.16, "MSCAN Identifier Mask Registers (CANIDMR0–CANIDMR7)").

A filter hit is indicated to the application software by a set receive buffer full flag (RXF = 1) and three bits in the CANIDAC register (see Section 15.3.11, "MSCAN Identifier Acceptance Control Register (CANIDAC)"). These identifier hit flags (IDHIT[2:0]) clearly identify the filter section that caused the acceptance. They simplify the application software's task to identify the cause of the receiver interrupt. If more than one hit occurs (two or more filters match), the lower hit has priority.

A very flexible programmable generic identifier acceptance filter has been introduced to reduce the CPU interrupt loading. The filter is programmable to operate in four different modes (see Bosch CAN 2.0A/B protocol specification):

- Two identifier acceptance filters, each to be applied to:
  - — The full 29 bits of the extended identifier and to the following bits of the CAN 2.0B frame:
    - – Remote transmission request (RTR)
    - – Identifier extension (IDE)
    - – Substitute remote request (SRR)
  - — The 11 bits of the standard identifier plus the RTR and IDE bits of the CAN 2.0A/B messages[1]. This mode implements two filters for a full length CAN 2.0B compliant extended identifier. Figure 15-38 shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces a filter 0 hit. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces a filter 1 hit.

[1] Although this mode can be used for standard identifiers, it is recommended to use the four or eight identifier acceptance filters for standard identifiers

- Four identifier acceptance filters, each to be applied to
  - a) the 14 most significant bits of the extended identifier plus the SRR and IDE bits of CAN 2.0B messages or
  - b) the 11 bits of the standard identifier, the RTR and IDE bits of CAN 2.0A/B messages. Figure 15-39 shows how the first 32-bit filter bank (CANIDAR0–CANIDA3, CANIDMR0–3CANIDMR) produces filter 0 and 1 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 2 and 3 hits.
- Eight identifier acceptance filters, each to be applied to the first 8 bits of the identifier. This mode implements eight independent filters for the first 8 bits of a CAN 2.0A/B compliant standard identifier or a CAN 2.0B compliant extended identifier. Figure 15-40 shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces filter 0 to 3 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 4 to 7 hits.
- Closed filter. No CAN message is copied into the foreground buffer RxFG, and the RXF flag is never set.

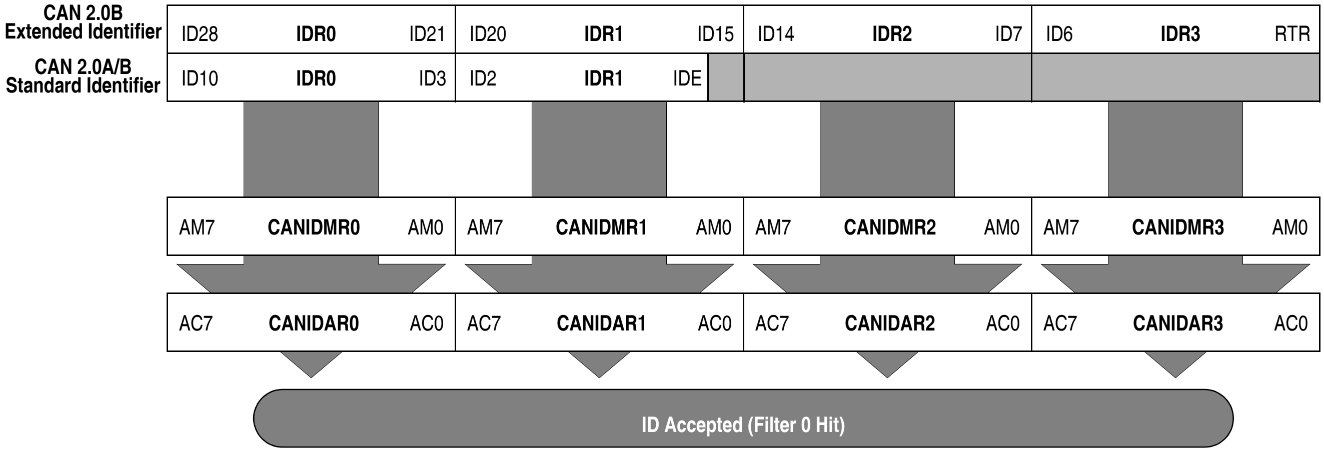


Figure 15-38. 32-bit Maskable Identifier Acceptance Filter

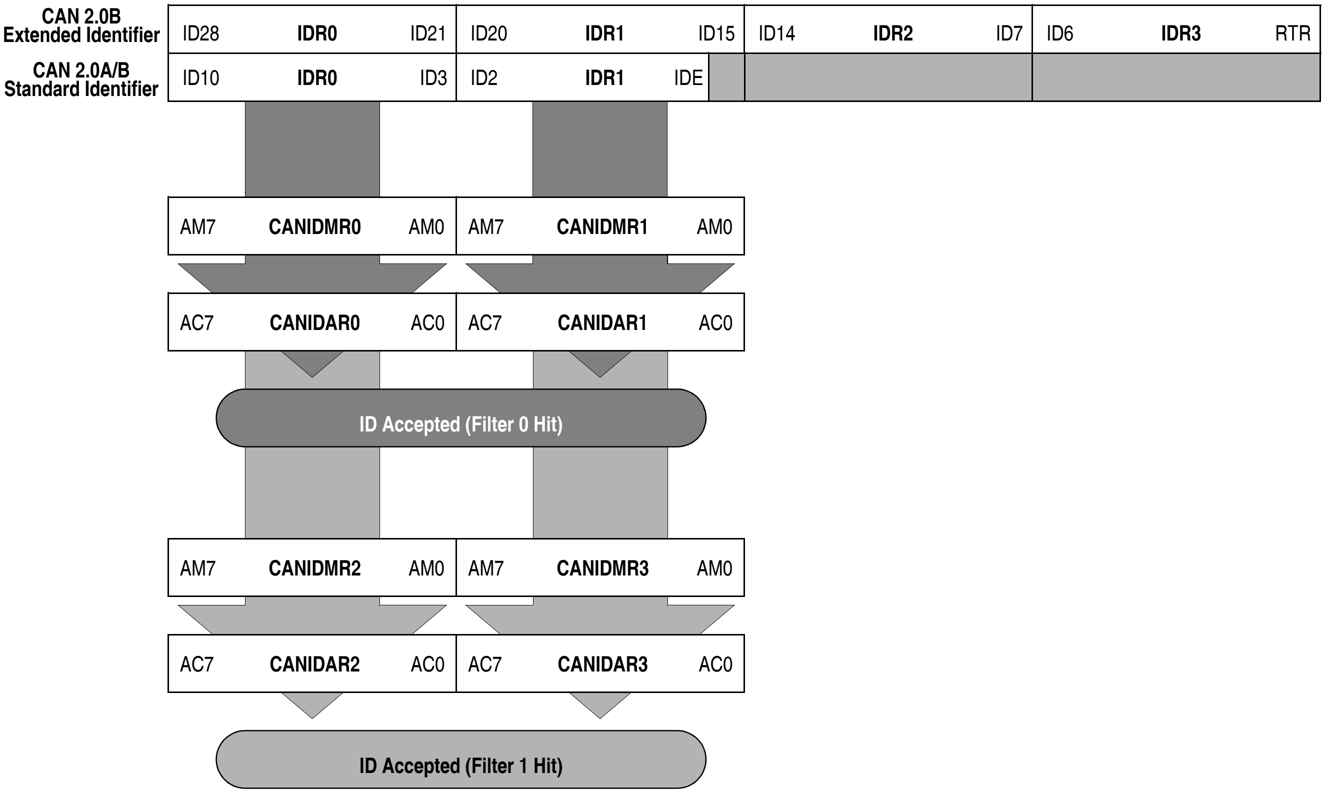


Figure 15-39. 16-bit Maskable Identifier Acceptance Filters

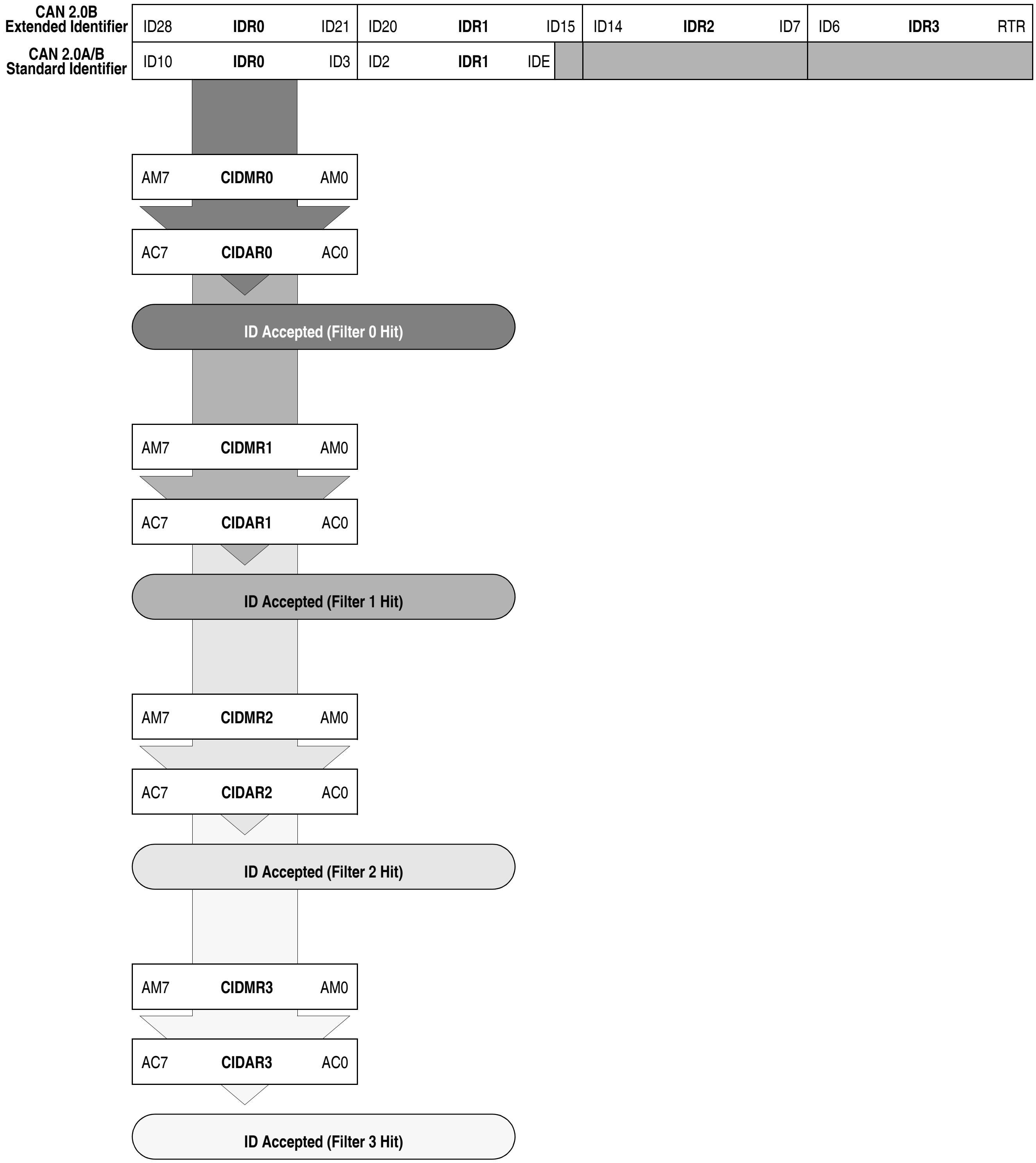


Figure 15-40. 8-bit Maskable Identifier Acceptance Filters

MSCAN filter uses three sets of registers to provide the filter configuration. Firstly, the CANIDAC register determines the configuration of the banks into filter sizes and number of filters. Secondly, registers CANIDMR0/1/2/3 determine those bits on which the filter will operate by placing a ‘0’ at the appropriate bit position in the filter register. Finally, registers CANIDAR0/1/2/3 determine the value of those bits determined by CANIDMR0/1/2/3.

For instance in the case of the filter value of:

0001x1001x0

The CANIDMR0/1/2/3 register would be configured as:

00001000010

and so all message identifier bits except bit 1 and bit 6 would be compared against the CANIDAR0/1/2/3 registers. These would be configured as:

00010100100

In this case bits 1 and 6 are set to '0', but since they are ignored it is equally valid to set them to '1'.

15.5.3.1 Identifier Acceptance Filters example

As described above, filters work by comparisons to individual bits in the CAN message identifier field. The filter will check each one of the eleven bits of a standard CAN message identifier. Suppose a filter value of 0001x1001x0. In this simple example, there are only three possible CAN messages.

Filter value: 0001x1001x0

Message 1: 00011100110

Message 2: 00110100110

Message 3: 00010100100

Message 2 will be rejected since its third most significant bit is not '0' - 001. The filter is simply a convenient way of defining the set of messages that the CPU must receive. For full 29-bits of an extended CAN message identifier, the filter identifies two sets of messages: one set that it receives and one set that it rejects. Alternatively, the filter may be split into two. This allows the MSCAN to examine only the first 16 bits of a message identifier, but allows two separate filters to perform the checking. See the example below:

Filter value A: 0001x1001x0

Filter value B: 00x101x01x0

Message 1: 00011100110

Message 2: 00110100110

Message 3: 00010100100

MSCAN will accept all three messages. Filter A will accept messages 1 and 3 as before and filter B will accept message 2. In practice, it is unimportant which filter accepts the message - messages accepted by either will be placed in the input buffer. A message may be accepted by more than one filter.

15.5.3.2 Protocol Violation Protection

The MSCAN protects the user from accidentally violating the CAN protocol through programming errors. The protection logic implements the following features:

- The receive and transmit error counters cannot be written or otherwise manipulated.
- All registers which control the configuration of the MSCAN cannot be modified while the MSCAN is on-line. The MSCAN has to be in Initialization Mode. The corresponding INITRQ/INITAK handshake bits in the CANCTL0/CANCTL1 registers (see Section 15.3.1, “MSCAN Control Register 0 (CANCTL0)”) serve as a lock to protect the following registers:
  - — MSCAN control 1 register (CANCTL1)
  - — MSCAN bus timing registers 0 and 1 (CANBTR0, CANBTR1)
  - — MSCAN identifier acceptance control register (CANIDAC)
  - — MSCAN identifier acceptance registers (CANIDAR0–CANIDAR7)
  - — MSCAN identifier mask registers (CANIDMR0–CANIDMR7)
- The TXCAN pin is immediately forced to a recessive state when the MSCAN goes into the power down mode or initialization mode (see Section 15.5.5.6, “MSCAN Power Down Mode,” and Section 15.5.5.5, “MSCAN Initialization Mode”).
- The MSCAN enable bit (CANE) is writable only once in normal system operation modes, which provides further protection against inadvertently disabling the MSCAN.

15.5.3.3 Clock System

Figure 15-41 shows the structure of the MSCAN clock generation circuitry.

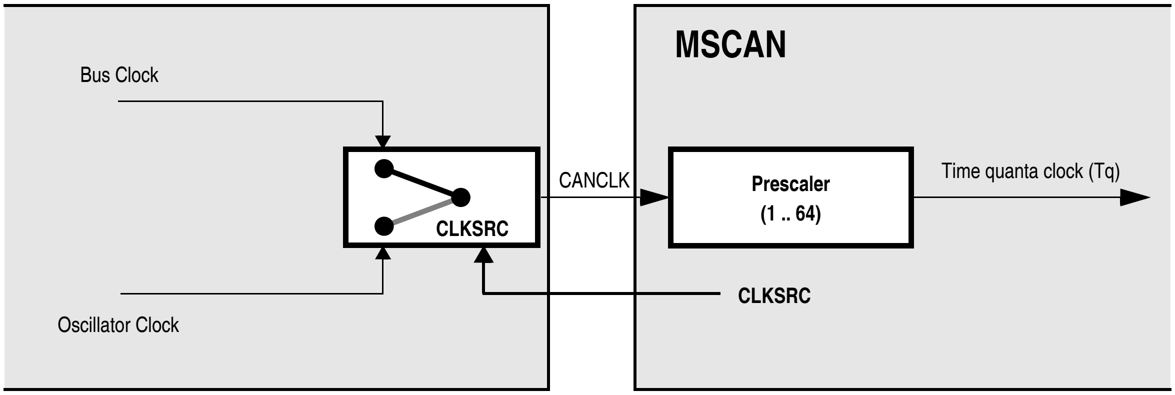


Figure 15-41. MSCAN Clocking Scheme

The clock source bit (CLKSRC) in the CANCTL1 register (15.3.2/15-6) defines whether the internal CANCLK is connected to the output of a crystal oscillator (oscillator clock) or to the bus clock.

The clock source has to be chosen such that the tight oscillator tolerance requirements (up to 0.4%) of the CAN protocol are met. Additionally, for high CAN bus rates (1 Mbps), a 45% to 55% duty cycle of the clock is required.

If the bus clock is generated from a PLL, it is recommended to select the oscillator clock rather than the bus clock due to jitter considerations, especially at the faster CAN bus rates. PLL lock may also be too wide to ensure adequate clock tolerance.

For microcontrollers without a clock and reset generator (CRG), CANCLK is driven from the crystal oscillator (oscillator clock).

A programmable prescaler generates the time quanta (Tq) clock from CANCLK. A time quantum is the atomic unit of time handled by the MSCAN.

Eqn. 15-2

$$f_{Tq} = \frac{f_{CANCLK}}{(\text{Prescaler value})}$$

A bit time is subdivided into three segments as described in the Bosch CAN specification. (see Figure 15-42):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the PROP_SEG and the PHASE_SEG1 of the CAN standard. It can be programmed by setting the parameter TSEG1 to consist of 4 to 16 time quanta.
- Time Segment 2: This segment represents the PHASE_SEG2 of the CAN standard. It can be programmed by setting the TSEG2 parameter to be 2 to 8 time quanta long.

Eqn. 15-3

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})}$$

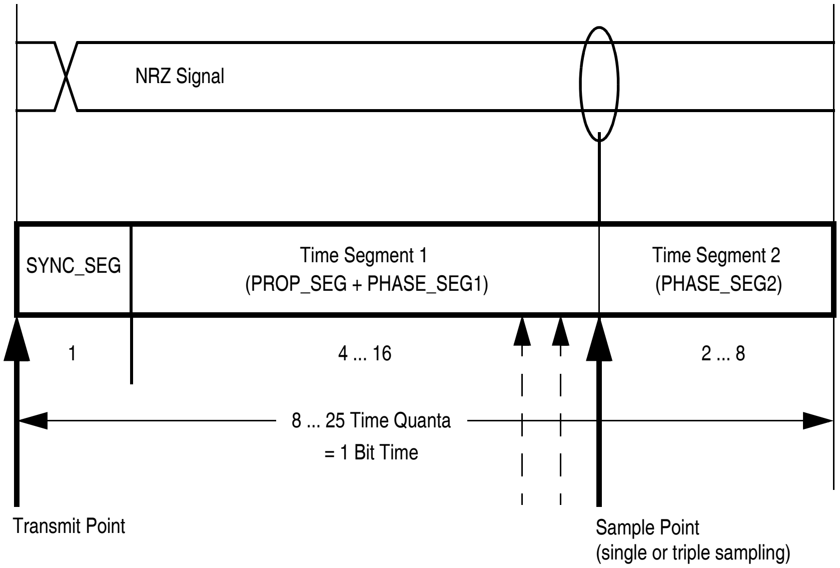


Figure 15-42. Segments within the Bit Time

Table 15-34. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the CAN bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the CAN bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

The synchronization jump width (see the Bosch CAN specification for details) can be programmed in a range of 1 to 4 time quanta by setting the SJW parameter.

The SYNC_SEG, TSEG1, TSEG2, and SJW parameters are set by programming the MSCAN bus timing registers (CANBTR0, CANBTR1) (see Section 15.3.3, "MSCAN Bus Timing Register 0 (CANBTR0)" and Section 15.3.4, "MSCAN Bus Timing Register 1 (CANBTR1)").

Table 15-35 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard.

Table 15-35. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	TSEG1	Time Segment 2	TSEG2	Synchronization Jump Width	SJW
5 .. 10	4 .. 9	2	1	1 .. 2	0 .. 1
4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
5 .. 12	4 .. 11	4	3	1 .. 4	0 .. 3
6 .. 13	5 .. 12	5	4	1 .. 4	0 .. 3
7 .. 14	6 .. 13	6	5	1 .. 4	0 .. 3
8 .. 15	7 .. 14	7	6	1 .. 4	0 .. 3
9 .. 16	8 .. 15	8	7	1 .. 4	0 .. 3

15.5.4 Modes of Operation

15.5.4.1 Normal Modes

The MSCAN module behaves as described within this specification in all normal system operation modes.

15.5.4.2 Special Modes

The MSCAN module behaves as described within this specification in all special system operation modes.

15.5.4.3 Emulation Modes

In all emulation modes, the MSCAN module behaves just like normal system operation modes as described within this specification.

15.5.4.4 Listen-Only Mode

In an optional CAN bus monitoring mode (listen-only), the CAN node is able to receive valid data frames and valid remote frames, but it sends only "recessive" bits on the CAN bus. In addition, it cannot start a transmision. If the MAC sub-layer is required to send a "dominant" bit (ACK bit, overload flag, or active error flag), the bit is rerouted internally so that the MAC sub-layer monitors this "dominant" bit, although the CAN bus may remain in recessive state externally.

15.5.4.5 Security Modes

The MSCAN module has no security features.

15.5.4.6 Loopback Self Test Mode

Loopback self test mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input. The RXCAN input pin is ignored and the TXCAN output goes to the recessive state (logic 1). The MSCAN behaves as it does normally when transmitting and treats its own transmitted message as a message received from a remote node. In this state, the MSCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

15.5.5 Low-Power Options

If the MSCAN is disabled (CANE = 0), the MSCAN clocks are stopped for power saving.

If the MSCAN is enabled (CANE = 1), the MSCAN has two additional modes with reduced power consumption, compared to normal mode: sleep and power down mode. In sleep mode, power consumption is reduced by stopping all clocks except those to access the registers from the CPU side. In power down mode, all clocks are stopped and no power is consumed.

Table 15-36 summarizes the combinations of MSCAN and CPU modes. A particular combination of modes is entered by the given settings on the CSWAI and SLPRQ/SLPAK bits.

For all modes, an MSCAN wake-up interrupt can occur only if the MSCAN is in sleep mode (SLPRQ = 1 and SLPAK = 1), wake-up functionality is enabled (WUPE = 1), and the wake-up interrupt is enabled (WUPIE = 1).

Table 15-36. CPU vs. MSCAN Operating Modes

CPU Mode	MSCAN Mode			
	Normal	Reduced Power Consumption		
		Sleep	Power Down	Disabled (CANE=0)

Run	CSWAI = X[1]
	SLPRQ = 0
	SLPAK = 0
		CSWAI = X
		SLPRQ = 1
		SLPAK = 1
				CSWAI = X
				SLPRQ = X
				SLPAK = X

Wait	CSWAI = 0
	SLPRQ = 0
	SLPAK = 0
		CSWAI = 0
		SLPRQ = 1
		SLPAK = 1
			CSWAI = 1
			SLPRQ = X
			SLPAK = X
				CSWAI = X
				SLPRQ = X
				SLPAK = X

Stop3		CSWAI = X[2]
		SLPRQ = 1
		SLPAK = 1
			CSWAI = X
			SLPRQ = 0
			SLPAK = 0
				CSWAI = X
				SLPRQ = X
				SLPAK = X

Stop1 or 2			CSWAI = X
			SLPRQ = X
			SLPAK = X
				CSWAI = X
				SLPRQ = X
				SLPAK = X

[1] 'X' means don't care.

[2] For a safe wake up from Sleep mode, SLPRQ and SLPAK must be set to 1 before going into stop3 mode.

15.5.5.1 Operation in Run Mode

As shown in Table 15-36, only MSCAN sleep mode is available as low power option when the CPU is in run mode.

15.5.5.2 Operation in Wait Mode

The WAIT instruction puts the MCU in a low power consumption stand-by mode. If the CSWAI bit is set, additional power can be saved in power down mode because the CPU clocks are stopped. After leaving this power down mode, the MSCAN restarts its internal controllers and enters normal mode again.

While the CPU is in wait mode, the MSCAN can be operated in normal mode and generate interrupts (registers can be accessed via background debug mode). The MSCAN can also operate in any of the low-power modes depending on the values of the SLPRQ/SLPAK and CSWAI bits as seen in Table 15-36.

15.5.5.3 Operation in Stop Mode

The STOP instruction puts the MCU in a low power consumption stand-by mode. In stop1 or stop2 modes, the MSCAN is set in power down mode regardless of the value of the SLPRQ/SLPAK. In stop3 mode, power down or sleep modes are determined by the SLPRQ/SLPAK values set prior to entering stop3. CSWAI bit has no function in any of the stop modes.Table 15-36.

15.5.5.4 MSCAN Sleep Mode

The CPU can request the MSCAN to enter this low power mode by asserting the SLPRQ bit in the CANCTL0 register. The time when the MSCAN enters sleep mode depends on a fixed synchronization delay and its current activity:

- If there are one or more message buffers scheduled for transmission (TXEx = 0), the MSCAN will continue to transmit until all transmit message buffers are empty (TXEx = 1, transmitted successfully or aborted) and then goes into sleep mode.
- If the MSCAN is receiving, it continues to receive and goes into sleep mode as soon as the CAN bus next becomes idle.
- If the MSCAN is neither transmitting nor receiving, it immediately goes into sleep mode.

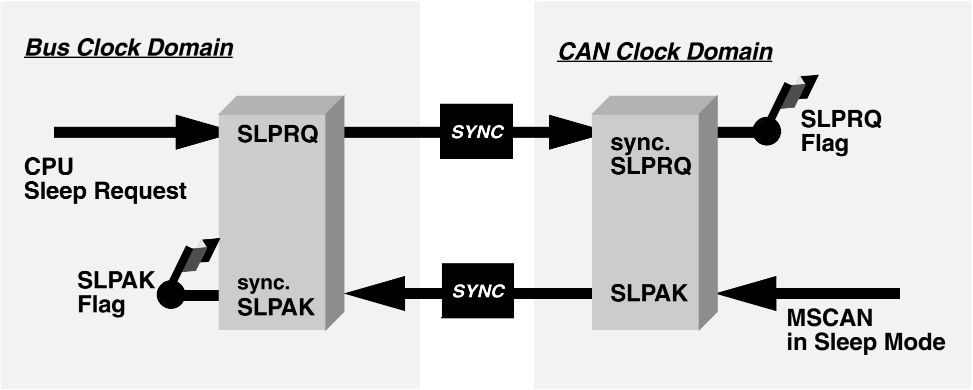


Figure 15-43. Sleep Request / Acknowledge Cycle

NOTE

The application software must avoid setting up a transmission (by clearing one or more TXEx flag(s)) and immediately request sleep mode (by setting SLPRQ). Whether the MSCAN starts transmitting or goes into sleep mode directly depends on the exact sequence of operations.

If sleep mode is active, the SLPRQ and SLPAK bits are set (Figure 15-43). The application software must use SLPAK as a handshake indication for the request (SLPRQ) to go into sleep mode.

When in sleep mode (SLPRQ = 1 and SLPAK = 1), the MSCAN stops its internal clocks. However, clocks that allow register accesses from the CPU side continue to run.

If the MSCAN is in bus-off state, it stops counting the 128 occurrences of 11 consecutive recessive bits due to the stopped clocks. The TXCAN pin remains in a recessive state. If RXF = 1, the message can be read and RXF can be cleared. Shifting a new message into the foreground buffer of the receiver FIFO (RxFG) does not take place while in sleep mode.

It is possible to access the transmit buffers and to clear the associated TXE flags. No message abort takes place while in sleep mode.

If the WUPE bit in CANCLT0 is not asserted, the MSCAN will mask any activity it detects on CAN. The RXCAN pin is therefore held internally in a recessive state. This locks the MSCAN in sleep mode (Figure 15-44). WUPE must be set before entering sleep mode to take effect.

The MSCAN is able to leave sleep mode (wake up) only when:

- CAN bus activity occurs and WUPE = 1

  or
- the CPU clears the SLPRQ bit

NOTE

The CPU cannot clear the SLPRQ bit before sleep mode (SLPRQ = 1 and SLPAK = 1) is active.

After wake-up, the MSCAN waits for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, if the MSCAN is woken-up by a CAN frame, this frame is not received.

The receive message buffers (RxFG and RxBG) contain messages if they were received before sleep mode was entered. All pending actions will be executed upon wake-up; copying of RxBG into RxFG, message aborts and message transmissions. If the MSCAN remains in bus-off state after sleep mode was exited, it continues counting the 128 occurrences of 11 consecutive recessive bits.

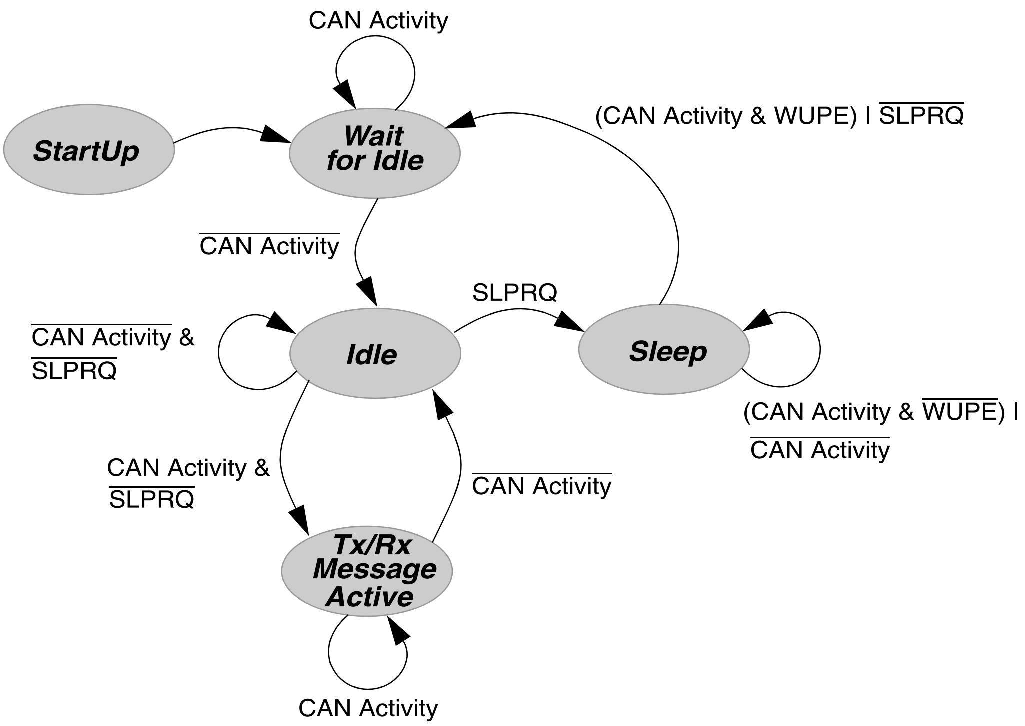

CAN Activity
StartUp
Wait for Idle
(CAN Activity & WUPE) | $\overline{\text{SLPRQ}}$
$\overline{\text{CAN Activity}}$
SLPRQ
$\overline{\text{CAN Activity}}$ & $\overline{\text{SLPRQ}}$
Idle
Sleep
(CAN Activity & $\overline{\text{WUPE}}$) | $\overline{\text{CAN Activity}}$
CAN Activity & $\overline{\text{SLPRQ}}$
$\overline{\text{CAN Activity}}$
Tx/Rx Message Active
CAN Activity

Figure 15-44. Simplified State Transitions for Entering/Leaving Sleep Mode

15.5.5.5 MSCAN Initialization Mode

In initialization mode, any on-going transmission or reception is immediately aborted and synchronization to the CAN bus is lost, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE

> The user is responsible for ensuring that the MSCAN is not active when initialization mode is entered. The recommended procedure is to bring the MSCAN into sleep mode (SLPRQ = 1 and SLPAK = 1) before setting the INITRQ bit in the CANCTL0 register. Otherwise, the abort of an on-going message can cause an error condition and can impact other CAN bus devices.

In initialization mode, the MSCAN is stopped. However, interface registers remain accessible. This mode is used to reset the CANCTL0, CANRFLG, CANRIER, CANTFLG, CANTIER, CANTARQ, CANTAAK, and CANTBSEL registers to their default values. In addition, the MSCAN enables the configuration of the CANBTR0, CANBTR1 bit timing registers; CANIDAC; and the CANIDAR, CANIDMR message filters. See Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)," for a detailed description of the initialization mode.

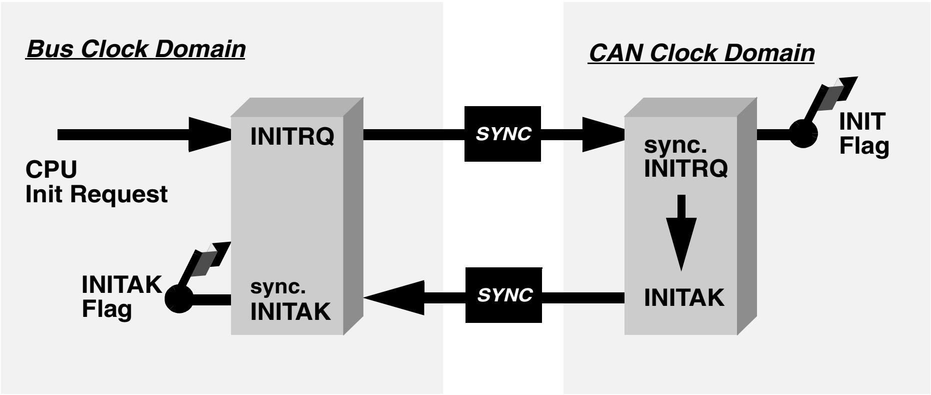


Figure 15-45. Initialization Request/Acknowledge Cycle

Due to independent clock domains within the MSCAN, INITRQ must be synchronized to all domains by using a special handshake mechanism. This handshake causes additional synchronization delay (see Section Figure 15-45., "Initialization Request/Acknowledge Cycle").

If there is no message transfer ongoing on the CAN bus, the minimum delay will be two additional bus clocks and three additional CAN clocks. When all parts of the MSCAN are in initialization mode, the INITAK flag is set. The application software must use INITAK as a handshake indication for the request (INITRQ) to go into initialization mode.

NOTE

> The CPU cannot clear INITRQ before initialization mode (INITRQ = 1 and INITAK = 1) is active.

15.5.5.6 MSCAN Power Down Mode

The MSCAN is in power down mode (Table 15-36) when

- CPU is in stop mode

  or

- CPU is in wait mode and the CSWAI bit is set

When entering the power down mode, the MSCAN immediately stops all ongoing transmissions and receptions, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations to the above rule, the MSCAN immediately drives the TXCAN pin into a recessive state.

NOTE

> The user is responsible for ensuring that the MSCAN is not active when power down mode is entered. The recommended procedure is to bring the MSCAN into Sleep mode before the STOP or WAIT instruction (if CSWAI is set) is executed. Otherwise, the abort of an ongoing message can cause an error condition and impact other CAN bus devices.

In power down mode, all clocks are stopped and no registers can be accessed. If the MSCAN was not in sleep mode before power down mode became active, the module performs an internal recovery cycle after powering up. This causes some fixed delay before the module enters normal mode again.

15.5.5.7 Programmable Wake-Up Function

The MSCAN can be programmed to wake up the MSCAN as soon as CAN bus activity is detected (see control bit WUPE in Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)"). The sensitivity to existing CAN bus action can be modified by applying a low-pass filter function to the RXCAN input line while in sleep mode (see control bit WUPM in Section 15.3.2, "MSCAN Control Register 1 (CANCTL1)").

This feature can be used to protect the MSCAN from wake-up due to short glitches on the CAN bus lines. Such glitches can result from—for example—electromagnetic interference within noisy environments.

15.5.6 Reset Initialization

The reset state of each individual bit is listed in Section 15.3, "Register Definition," which details all the registers and their bit-fields.

15.5.7 Interrupts

This section describes all interrupts originated by the MSCAN. It documents the enable bits and generated flags. Each interrupt is listed and described separately.

15.5.7.1 Description of Interrupt Operation

The MSCAN supports four interrupt vectors (see Table 15-37), any of which can be individually masked (for details see sections from Section 15.3.5, "MSCAN Receiver Interrupt Enable Register (CANRIER)," to Section 15.3.7, "MSCAN Transmitter Interrupt Enable Register (CANTIER)").

NOTE

The dedicated interrupt vector addresses are defined in the Resets and Interrupts chapter.

Table 15-37. Interrupt Vectors

Interrupt Source	CCR Mask	Local Enable
Wake-Up Interrupt (WUPIF)	I bit	CANRIER (WUPIE)
Error Interrupts Interrupt (CSCIF, OVRIF)	I bit	CANRIER (CSCIE, OVRIE)
Receive Interrupt (RXF)	I bit	CANRIER (RXFIE)
Transmit Interrupts (TXE[2:0])	I bit	CANTIER (TXEIE[2:0])

15.5.7.2 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. The TXEx flag of the empty message buffer is set.

15.5.7.3 Receive Interrupt

A message is successfully received and shifted into the foreground buffer (RxFG) of the receiver FIFO. This interrupt is generated immediately after receiving the EOF symbol. The RXF flag is set. If there are multiple messages in the receiver FIFO, the RXF flag is set as soon as the next message is shifted to the foreground buffer.

15.5.7.4 Wake-Up Interrupt

A wake-up interrupt is generated if activity on the CAN bus occurs during MSCAN internal sleep mode. WUPE (see Section 15.3.1, "MSCAN Control Register 0 (CANCTL0)") must be enabled.

15.5.7.5 Error Interrupt

An error interrupt is generated if an overrun of the receiver FIFO, error, warning, or bus-off condition occurs. Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG) indicates one of the following conditions:

- **Overrun** — An overrun condition of the receiver FIFO as described in Section 15.5.2.3, "Receive Structures," occurred.
- **CAN Status Change** — The actual value of the transmit and receive error counters control the CAN bus state of the MSCAN. As soon as the error counters skip into a critical range (Tx/Rx-warning, Tx/Rx-error, bus-off) the MSCAN flags an error condition. The status change, which caused the error condition, is indicated by the TSTAT and RSTAT flags (see

Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG)" and Section 15.3.5, "MSCAN Receiver Interrupt Enable Register (CANRIER)").

15.5.7.6 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in either the Section 15.3.4.1, "MSCAN Receiver Flag Register (CANRFLG)" or the Section 15.3.6, "MSCAN Transmitter Flag Register (CANTFLG)." Interrupts are pending as long as one of the corresponding flags is set. The flags in CANRFLG and CANTFLG must be reset within the interrupt handler to handshake the interrupt. The flags are reset by writing a 1 to the corresponding bit position. A flag cannot be cleared if the respective condition prevails.

NOTE

It must be guaranteed that the CPU clears only the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

15.5.7.7 Recovery from Stop or Wait

The MSCAN can recover from stop or wait via the wake-up interrupt. This interrupt can only occur if the MSCAN was in sleep mode (SLPRQ = 1 and SLPAK = 1) before entering power down mode, the wake-up option is enabled (WUPE = 1), and the wake-up interrupt is enabled (WUPIE = 1).

15.6 Initialization/Application Information

15.6.1 MSCAN initialization

The procedure to initially start up the MSCAN module out of reset is as follows:

1. Assert CANE
2. Write to the configuration registers in initialization mode
3. Clear INITRQ to leave initialization mode and enter normal mode

If the configuration of registers which are writable in initialization mode needs to be changed only when the MSCAN module is in normal mode:

1. Bring the module into sleep mode by setting SLPRQ and awaiting SLPAK to assert after the CAN bus becomes idle.
2. Enter initialization mode: assert INITRQ and await INITAK
3. Write to the configuration registers in initialization mode
4. Clear INITRQ to leave initialization mode and continue in normal mode

15.6.2 Bus-Off Recovery

The bus-off recovery is user configurable. The bus-off state can either be exited automatically or on user request.

For reasons of backwards compatibility, the MSCAN defaults to automatic recovery after reset. In this case, the MSCAN will become error active again after counting 128 occurrences of 11 consecutive recessive bits on the CAN bus (See the Bosch CAN specification for details).

If the MSCAN is configured for user request (BORM set in Section 15.3.2, “MSCAN Control Register 1 (CANCTL1)”), the recovery from bus-off starts after both independent events have become true:

- 128 occurrences of 11 consecutive recessive bits on the CAN bus have been monitored
- BOHOLD in Section 15.3.12, “MSCAN Miscellaneous Register (CANMISC) has been cleared by the user

These two events may occur in any order.

Chapter 16
Multipurpose Clock Generator (MCGV3)

16.1 Introduction

The multipurpose clock generator (MCG) module provides several clock source choices for this device. The module contains a frequency-locked loop (FLL) and a phase-locked loop (PLL) that are controllable by either an internal or an external reference clock. The module can select either of the FLL or PLL 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 which allows a lower output clock frequency to be derived. The MCG also controls a crystal oscillator (XOSC), which allows an external crystal, ceramic resonator, or another external clock source to produce the external reference clock.

16.1.1 Features

Key features of the MCG module are:

- Frequency-locked loop (FLL)
  - Internal or external reference clock can be used to control the FLL
- Phase-locked loop (PLL)
  - Voltage-controlled oscillator (VCO)
  - Modulo VCO frequency divider
  - Phase/Frequency detector
  - Integrated loop filter
  - Lock detector with interrupt capability
- Internal reference clock
  - Nine trim bits for accuracy
  - Can be selected as the clock source for the MCU
- External reference clock
  - Control for a separate crystal oscillator
  - Clock monitor with reset capability
  - Can be selected as the clock source for the MCU
- Reference divider is provided
- Clock source selected can be divided down by 1, 2, 4, or 8
- BDC clock (MCGLCLK) is provided as a constant divide-by-2 of the DCO output whether in an FLL or PLL mode.Three selectable digitally controlled oscillators (DCOs) optimized for different frequency ranges.
- Option to maximize DCO output frequency for a 32,768 Hz external reference clock source.
- The PLL can be used to drive MCGPLLSCLK even when MCGOUT is driven from one of the reference clocks (PBE mode).

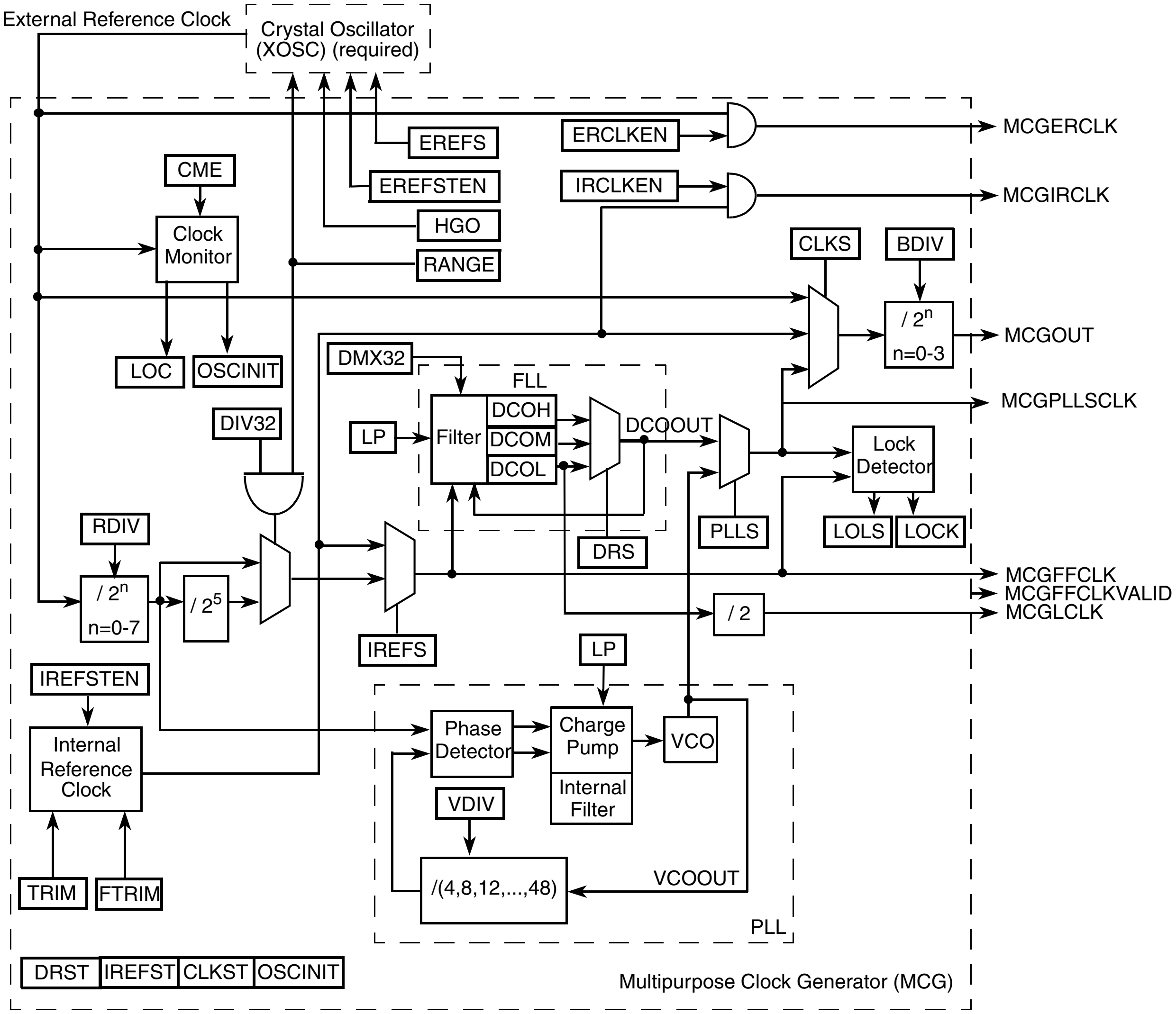


Figure 16-1. Multipurpose Clock Generator (MCG) Block Diagram

NOTE

The MCG requires the attachment of a crystal oscillator (XOSC) module, which provides an external reference clock.

16.1.2 Modes of Operation

There are several modes of operation for the MCG. For details, see Section 16.4.1, “MCG Modes of Operation.”

16.2 External Signal Description

There are no MCG signals that connect off chip.

16.3 Register Definition

16.3.1 MCG Control Register 1 (MCGC1)

	7	6	5	4	3	2	1	0
R W	CLKS		RDIV			IREFS	IRCLKEN	IREFSTEN
Reset:	0	0	0	0	0	1	0	0

Figure 16-2. MCG Control Register 1 (MCGC1)

Table 16-1. MCG Control Register 1 Field Descriptions

Field	Description
7:6 CLKS	**Clock Source Select** — Selects the system clock source. 00 Encoding 0 — Output of FLL or PLL is selected. 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Reserved, defaults to 00.
5:3 RDIV	**External Reference Divider** — Selects the amount to divide down the external reference clock. If the FLL is selected, the resulting frequency must be in the range 31.25 kHz to 39.0625 kHz. If the PLL is selected, the resulting frequency must be in the range 1 MHz to 2 MHz. See Table 16-2 and Table 16-3 for the divide-by factors.
2 IREFS	**Internal Reference Select** — Selects the reference clock source. 1 Internal reference clock selected 0 External reference clock selected
1 IRCLKEN	**Internal Reference Clock Enable** — Enables the internal reference clock for use as MCGIRCLK. 1 MCGIRCLK active 0 MCGIRCLK inactive
0 IREFSTEN	**Internal Reference Stop Enable** — Controls whether or not the internal reference clock remains enabled when the MCG enters stop mode. 1 Internal reference clock stays enabled in stop if IRCLKEN is set or if MCG is in FEI, FBI, or BLPI mode before entering stop 0 Internal reference clock is disabled in stop

Table 16-2. FLL External Reference Divide Factor

RDIV	Divide Factor		
	RANGE:DIV32 0:X	RANGE:DIV32 1:0	RANGE:DIV32 1:1
0	1	1	32
1	2	2	64
2	4	4	128
3	8	8	256
4	16	16	512
5	32	32	1024
6	64	64	Reserved
7	128	128	Reserved

Table 16-3. PLL External Reference Divide Factor

RDIV	Divide Factor
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

16.3.2 MCG Control Register 2 (MCGC2)

	7	6	5	4	3	2	1	0
R W	BDIV		RANGE	HGO	LP	EREFS	ERCLKEN	EREFSTEN
Reset:	0	1	0	0	0	0	0	0

Figure 16-3. MCG Control Register 2 (MCGC2)

Table 16-4. MCG Control Register 2 Field Descriptions

Field	Description
7:6 BDIV	**Bus Frequency Divider** — Selects the amount to divide down the clock source selected by the CLKS bits in the MCGC1 register. 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	**Frequency Range Select** — Selects the frequency range for the crystal oscillator or external clock source. 1 High frequency range selected for the crystal oscillator of 1 MHz to 16 MHz (1 MHz to 40 MHz for external clock source) 0 Low frequency range selected for the crystal oscillator of 32 kHz to 100 kHz (32 kHz to 1 MHz for external clock source)
4 HGO	**High Gain Oscillator Select** — Controls the crystal oscillator mode of operation. 1 Configure crystal oscillator for high gain operation 0 Configure crystal oscillator for low power operation
3 LP	**Low Power Select** — Controls whether the FLL (or PLL) is disabled in bypassed modes. 1 FLL (or PLL) is disabled in bypass modes (lower power). 0 FLL (or PLL) is not disabled in bypass modes.
2 EREFS	**External Reference Select** — Selects the source for the external reference clock. 1 Oscillator requested 0 External Clock Source requested
1 ERCLKEN	**External Reference Enable** — Enables the external reference clock for use as MCGERCLK. 1 MCGERCLK active 0 MCGERCLK inactive
0 EREFSTEN	**External Reference Stop Enable** — Controls whether or not the external reference clock remains enabled when the MCG enters stop mode. 1 External reference clock stays enabled in stop if ERCLKEN is set or if MCG is in FEE, FBE, PEE, PBE, or BLPE mode before entering stop 0 External reference clock is disabled in stop

16.3.3 MCG Trim Register (MCGTRM)

	7	6	5	4	3	2	1	0
R	TRIM[1]							
W								

Figure 16-4. MCG Trim Register (MCGTRM)

[1] A value for TRIM is loaded during reset from a factory programmed location when not in any BDM mode. If in a BDM mode, a default value of 0x80 is loaded.

Table 16-5. MCG Trim Register Field Descriptions

Field	Description
7:0 TRIM	MCG **Trim Setting** — Controls the internal reference clock frequency by controlling the internal reference clock period. The TRIM bits are binary weighted (i.e., bit 1 will adjust twice as much as bit 0). Increasing the binary value in TRIM will increase the period, and decreasing the value will decrease the period. An additional fine trim bit is available in MCGSC as the FTRIM bit. If a TRIM[7:0] value stored in nonvolatile memory is to be used, it's the user's responsibility to copy that value from the nonvolatile memory location to this register.

16.3.4 MCG Status and Control Register (MCGSC)

	7	6	5	4	3	2	1	0
R	LOLS	LOCK	PLLST	IREFST	CLKST		OSCINIT	FTRIM[1]
W								
Reset:	0	0	0	1	0	0	0	

Figure 16-5. MCG Status and Control Register (MCGSC)

[1] A value for FTRIM is loaded during reset from a factory programmed location when not in any BDM mode. If in a BDM mode, a default value of 0x0 is loaded.

Table 16-6. MCG Status and Control Register Field Description

Field	Description
7 LOLS	**Loss of Lock Status** — This bit is a sticky indication of lock status for the FLL or PLL. LOLS is set when lock detection is enabled and after acquiring lock, the FLL or PLL output frequency has fallen outside the lock exit frequency tolerance, D_{unl}. LOLIE determines whether an interrupt request is made when set. LOLS is cleared by reset or by writing a logic 1 to LOLS when LOLS is set. Writing a logic 0 to LOLS has no effect. 0 FLL or PLL has not lost lock since LOLS was last cleared. 1 FLL or PLL has lost lock since LOLS was last cleared.
6 LOCK	**Lock Status** — Indicates whether the FLL or PLL has acquired lock. Lock detection is disabled when both the FLL and PLL are disabled. If the lock status bit is set, changing the value of DMX32, DRS[1:0] and IREFS bits in FBE, FBI, FEE and FEI modes; DIV32 bit in FBE and FEE modes; TRIM[7:0] bits in FBI and FEI modes; RDIV[2:0] bits in FBE, FEE, PBE and PEE modes; VDIV[3:0] bits in PBE and PEE modes; and PLLS bit, causes the lock status bit to clear and stay clear until the FLL or PLL has reacquired lock. Entry into BLPI, BLPE or stop mode also causes the lock status bit to clear and stay cleared until the exit of these modes and the FLL or PLL has reacquired lock. 0 FLL or PLL is currently unlocked. 1 FLL or PLL is currently locked.
5 PLLST	**PLL Select Status** — The PLLST bit indicates the current source for the PLLS clock. The PLLST bit does not update immediately after a write to the PLLS bit due to internal synchronization between clock domains. 0 Source of PLLS clock is FLL clock. 1 Source of PLLS clock is PLL clock.
4 IREFST	**Internal Reference Status** — 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. 0 Source of reference clock is external reference clock (oscillator or external clock source as determined by the EREFS bit in the MCGC2 register). 1 Source of reference clock is internal reference clock.
3:2 CLKST	**Clock Mode Status** — The CLKST bits indicate the current clock mode. The CLKST bits do not update immediately after a write to the CLKS bits due to internal synchronization between clock domains. 00 Encoding 0 — Output of FLL is selected. 01 Encoding 1 — Internal reference clock is selected. 10 Encoding 2 — External reference clock is selected. 11 Encoding 3 — Output of PLL is selected.

Table 16-6. MCG Status and Control Register Field Description (continued)

Field	Description
1 OSCINIT	**OSC Initialization** — If the external reference clock is selected by ERCLKEN or by the MCG being in FEE, FBE, PEE, PBE, or BLPE mode, and if EREFS is set, then this bit is set after the initialization cycles of the crystal oscillator clock have completed. This bit is only cleared when either EREFS is cleared or when the MCG is in either FEI, FBI, or BLPI mode and ERCLKEN is cleared.
0 FTRIM	**MCG Fine Trim** — 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. If an FTRIM value stored in nonvolatile memory is to be used, it’s the user’s responsibility to copy that value from the nonvolatile memory location to this register’s FTRIM bit.

16.3.5 MCG Control Register 3 (MCGC3)

	7	6	5	4	3	2	1	0
R W	LOLIE	PLLS	CME	DIV32	VDIV			
Reset:	0	0	0	0	0	0	0	1

Figure 16-6. MCG PLL Register (MCGPLL)

Table 16-7. MCG PLL Register Field Descriptions

Field	Description
7 LOLIE	**Loss of Lock Interrupt Enable** — Determines if an interrupt request is made following a loss of lock indication. The LOLIE bit only has an effect when LOLS is set. 0 No request on loss of lock. 1 Generate an interrupt request on loss of lock.
6 PLLS	**PLL Select** — Controls whether the PLL or FLL is selected. If the PLLS bit is clear, the PLL is disabled in all modes. If the PLLS is set, the FLL is disabled in all modes. 1 PLL is selected 0 FLL is selected
5 CME	**Clock Monitor Enable** — 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 MCG is in an operational mode that uses the external clock (FEE, FBE, PEE, PBE, or BLPE) or the external reference is enabled (ERCLKEN=1 in the MCGC2 register). Whenever the CME bit is set to a logic 1, the value of the RANGE bit in the MCGC2 register should not be changed. If the external reference clock is set to be disabled when the MCG enters STOP mode (EREFSTEN=0), then the CME bit should be set to a logic 0 before the MCG enters STOP mode. Otherwise a reset request may occur while in STOP mode. 0 Clock monitor is disabled. 1 Generate a reset request on loss of external clock.

Table 16-7. MCG PLL Register Field Descriptions (continued)

Field	Description
4 DIV32	**Divide-by-32 Enable** — Controls an additional divide-by-32 factor to the external reference clock for the FLL when RANGE bit is set. When the RANGE bit is 0, this bit has no effect. Writes to this bit are ignored if PLLS bit is set. 0 Divide-by-32 is disabled. 1 Divide-by-32 is enabled when RANGE=1.
3:0 VDIV	**VCO Divider** — Selects the amount to divide down the VCO output of PLL. The VDIV bits establish the multiplication factor (M) applied to the reference clock frequency. 0000 Encoding 0 — Reserved. 0001 Encoding 1 — Multiply by 4. 0010 Encoding 2 — Multiply by 8. 0011 Encoding 3 — Multiply by 12. 0100 Encoding 4 — Multiply by 16. 0101 Encoding 5 — Multiply by 20. 0110 Encoding 6 — Multiply by 24. 0111 Encoding 7 — Multiply by 28. 1000 Encoding 8 — Multiply by 32. 1001 Encoding 9 — Multiply by 36. 1010 Encoding 10 — Multiply by 40. 1011 Encoding 11 — Multiply by 44. 1100 Encoding 12 — Multiply by 48. 1101 Encoding 13 — Reserved (default to M=48). 111x Encoding 14-15 — Reserved (default to M=48).

16.3.6 MCG Control Register 4 (MCGC4)

	7	6	5	4	3	2	1	0
R	0	0	DMX32	0	0	0	DRST	
W							DRS	
Reset:	0	0	0	0	0	0	0	0

Figure 16-7. MCG Control Register 4 (MCGC4)

Table 16-8. MCG Test and Control Register Field Descriptions

Field	Description
7:6	Reserved for test, user code should not write 1’s to these bits.
5 DMX32	**DCO Maximum frequency with 32.768 kHz reference** — The DMX32 bit controls whether or not the DCO frequency range is narrowed to its maximum frequency with a 32.768 kHz reference. See Table 16-9. 0 DCO has default range of 25%. 1 DCO is fined tuned for maximum frequency with 32.768 kHz reference.
4:2	Reserved for test, user code should not write 1’s to these bits.
1:0 DRST DRS	**DCO Range Status** — The DRST read bits indicate the current frequency range for the FLL output, DCOOUT. See Table 16-9. The DRST bits do not update immediately after a write to the DRS field due to internal synchronization between clock domains. The DRST bits are not valid in BLPI, BLPE, PBE or PEE mode and it reads zero regardless of the DCO range selected by the DRS bits. **DCO Range Select** — The DRS bits select the frequency range for the FLL output, DCOOUT. Writes to the DRS bits while either the LP or PLLS bit is set are ignored. 00 Low range. 01 Mid range. 10 High range. 11 Reserved

Table 16-9. DCO frequency range[1]

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			

[1] The resulting bus clock frequency should not exceed the maximum specified bus clock frequency of the device.

16.3.7 MCG Test Register (MCGT)

Table 16-10. MCG Test Register Field Descriptions

Field	Description
7:6	Reserved for test, user code should not write 1’s to these bits.
5	Reserved, user code should not write 1’s to these bits
4:1	Reserved for test, user code should not write 1’s to these bits.
0	Reserved, user code should not write 1’s to these bits

16.4 Functional Description

16.4.1 MCG Modes of Operation

The MCG operates in one of the modes described in Table 16-11.

NOTE

The MCG restricts transitions between modes. For the permitted transitions, see Section 16.4.2, "MCG Mode State Diagram."

Table 16-11. MCG Modes of Operation

Mode	Related field values	Description
FLL Engaged Internal (FEI)	• MCGC1[IREFS] = 1 • MCGC1[CLKS] = 00 • MCGC3[PLLS] = 0	Default. MCGOUT is derived from the FLL clock, which is controlled by the internal reference clock. The FLL clock frequency locks to a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, times the internal reference frequency. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Engaged External (FEE)	• MCGC1[IREFS] = 0 • MCGC1[CLKS] = 00 • MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 31.2500 to 39.0625 kHz. • MCGC3[PLLS] = 0	MCGOUT is derived from the FLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The FLL clock frequency locks to a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Bypassed Internal (FBI)	• MCGC1[IREFS] = 1 • MCGC1[CLKS] = 01 • MCGC2[LP] = 0 (or the BDM is enabled) • MCGC3[PLLS] = 0	MCGOUT is derived from the internal reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while the MCGOUT clock is driven from the internal reference clock. MCGOUT is derived from the internal reference clock. The FLL clock is controlled by the internal reference clock, and the FLL clock frequency locks to a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, times the internal reference frequency. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.
FLL Bypassed External (FBE)	• MCGC1[IREFS] = 0 • MCGC1[CLKS] = 10 • MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 31.2500 to 39.0625 kHz • MCGC2[LP] = 0 (or the BDM is enabled) • MCGC3[PLLS] = 0	MCGOUT is derived from the external reference clock; the FLL is operational, but its output clock is not used. This mode is useful to allow the FLL to acquire its target frequency while MCGOUT is driven from the external reference clock. MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC).The FLL clock is controlled by the external reference clock, and the FLL clock frequency locks to a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, times the external reference frequency, as selected by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. MCGLCLK is derived from the FLL, and the PLL is disabled in a low-power state.

Table 16-11. MCG Modes of Operation (continued)

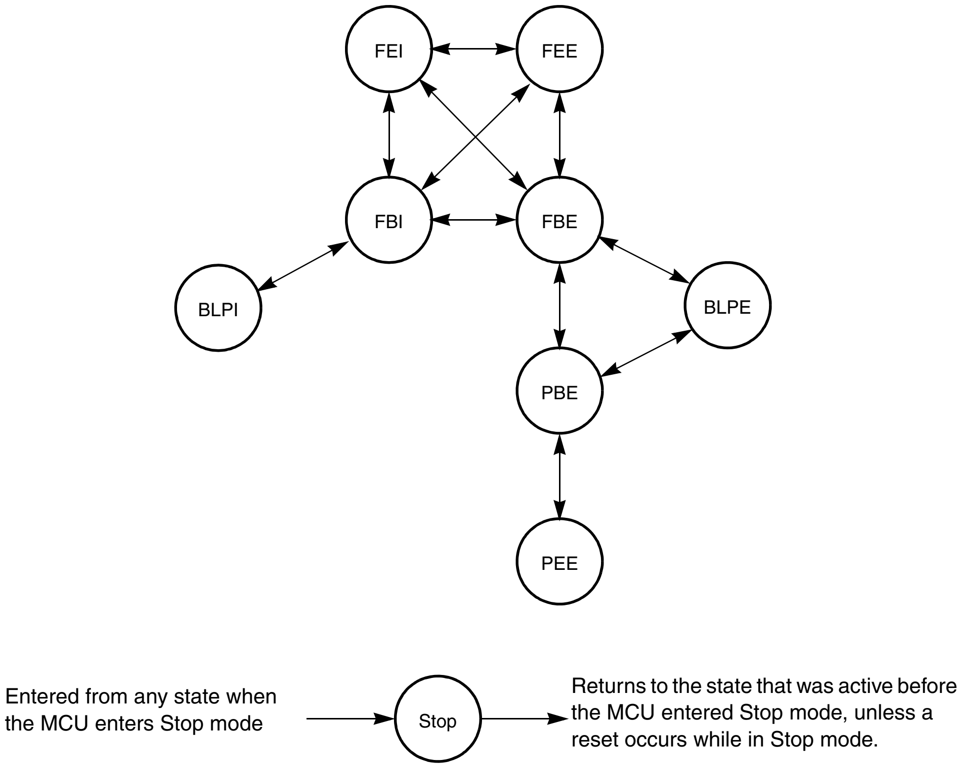
Mode	Related field values	Description
PLL Engaged External (PEE)	• MCGC1[IREFS] = 0 • MCGC1[CLKS] = 00 • MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 1 to 2 MHz. • PLLS = 1	MCGOUT is derived from the PLL clock, which is controlled by the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL clock frequency locks to a multiplication factor, as specified by MCGC3[VDIV], times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. If the BDM is enabled, MCGLCLK is derived from the DCO (open-loop mode) divided by two. If the BDM is not enabled, the FLL is disabled in a low-power state. In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].
PLL Bypassed External (PBE)	• MCGC1[IREFS] = 0 • MCGC1[CLKS] = 10 • MCGC1[RDIV] is programmed to divide the reference clock to be within the range of 1 to 2 MHz. • MCGC2[LP] = 0 • MCGC3[PLLS] = 1	MCGOUT is derived from the external reference clock; the PLL is operational, but its output clock is not used. This mode is useful to allow the PLL to acquire its target frequency while MCGOUT is driven from the external reference clock. MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL clock frequency locks to a multiplication factor, as specified by MCGC3[VDIV], times the external reference frequency, as specified by MCGC1[RDIV], MCGC2[RANGE], and MCGC3[DIV32]. If the BDM is enabled, MCGLCLK is derived from the DCO (open-loop mode) divided by two. If the BDM is not enabled, the FLL is disabled in a low-power state. In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].
Bypassed Low Power Internal (BLPI)	• MCGC1[IREFS] = 1 • MCGC1[CLKS] = 01MCGC3[PLLS] = 0 • MCGC2[LP] = 1 (and the BDM is disabled)	MCGOUT is derived from the internal reference clock. The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to FLL bypassed internal (FBI) mode. In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].
Bypassed Low Power External (BLPE)	• MCGC1[IREFS] = 0 • MCGC1[CLKS] = 10 • MCGC2[LP] = 1 (and the BDM is disabled)	MCGOUT is derived from the external reference clock. The external reference clock that is enabled can be produced by an external crystal, ceramic resonator, or another external clock source connected to the required crystal oscillator (XOSC). The PLL and FLL are disabled, and MCGLCLK is not available for BDC communications. If the BDM becomes enabled, the mode switches to one of the bypassed external modes as determined by the state of MCGC3[PLLS]. In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].

Table 16-11. MCG Modes of Operation (continued)

Mode	Related field values	Description
Stop	—	Entered whenever the MCU enters a Stop state. The FLL and PLL are disabled, and all MCG clock signals are static except in the following cases: MCGIRCLK is active in Stop mode when all the following conditions become true: • MCGC1[IRCLKEN] = 1 • MCGC1[IREFSTEN] = 1 MCGERCLK is active in Stop mode when all the following conditions become true: • MCGC2[ERCLKEN] = 1 • MCGC2[EREFSTEN] = 1

16.4.2 MCG Mode State Diagram

Figure 16-9 shows the MCG's mode state diagram. The arrows indicate the permitted mode transitions.



Entered from any state when the MCU enters Stop mode

Returns to the state that was active before the MCU entered Stop mode, unless a reset occurs while in Stop mode.

Figure 16-9. MCG Mode State Diagram

16.4.3 Mode Switching

The IREFS bit can be changed at anytime, but the actual switch to the newly selected clock is shown by the IREFST bit. When switching between engaged internal and engaged external modes, the FLL or PLL will begin 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 will remain 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.

For details see Figure 16-9.

16.4.4 Bus Frequency Divider

The BDIV bits can be changed at anytime and the actual switch to the new frequency will occur immediately.

16.4.5 Low Power Bit Usage

The low power bit (LP) is provided to allow the FLL or PLL to be disabled and thus conserve power when these systems are not being used. The DRS bit can not be written while LP bit is 1.However, in some applications it may be desirable to enable the FLL or PLL and allow it to lock for maximum accuracy before switching to an engaged mode. Do this by writing the LP bit to 0.

16.4.6 Internal Reference Clock

When IRCLKEN is set the internal reference clock signal will be presented as MCGIRCLK, which can be used as an additional clock source. The MCGIRCLK frequency can be re-targeted by trimming the period of the internal reference clock. This can be done by writing a new value to the TRIM bits in the MCGTRM register. Writing a larger value will decrease the MCGIRCLK frequency, and writing a smaller value to the MCGTRM register will increase the MCGIRCLK frequency. The TRIM bits will effect the MCGOUT frequency if the MCG is in FLL engaged internal (FEI), FLL bypassed internal (FBI), or bypassed low power internal (BLPI) mode. The TRIM and FTRIM value is initialized by POR but is not affected by other resets.

Until MCGIRCLK is trimmed, programming low reference divider (RDIV) factors may result in MCGOUT frequencies that exceed the maximum chip-level frequency and violate the chip-level clock timing specifications (see the Device Overview chapter).

If IREFSTEN and IRCLKEN bits are both set, the internal reference clock will keep running during stop mode in order to provide a fast recovery upon exiting stop.

16.4.7 External Reference Clock

The MCG module can support an external reference clock with frequencies between 31.25 kHz to 40 MHz in all modes. When ERCLKEN is set, the external reference clock signal will be presented as MCGERCLK, which can be used as an additional clock source. When IREFS = 1, the external reference clock will not be used by the FLL or PLL and will only be used as MCGERCLK. In these modes, the frequency can be equal to the maximum frequency the chip-level timing specifications will support (see the Device Overview chapter).

If EREFSTEN and ERCLKEN bits are both set or the MCG is in FEE, FBE, PEE, PBE or BLPE mode, the external reference clock will keep running during stop mode in order to provide a fast recovery upon exiting stop.

If CME bit is written to 1, the clock monitor is enabled. If the external reference falls below a certain frequency (f_{loc_high} or f_{loc_low} depending on the RANGE bit in the MCGC2), the MCU will reset. The LOC bit in the System Reset Status (SRS) register will be set to indicate the error.

16.4.8 Fixed Frequency Clock

The MCG presents the divided reference clock as MCGFFCLK for use as an additional clock source. The MCGFFCLK frequency must be no more than 1/4 of the MCGOUT frequency to be valid. When MCGFFCLK is valid then MCGFFCLKVALID is set to 1. When MCGFFCLK is not valid then MCGFFCLKVALID is set to 0.

This clock is intended for use in systems which include a USB interface. It allows the MCG to supply a 48MHz clock to the USB. This same clock can be used to derive MCGOUT. Alternately, MCGOUT can be derived from either internal or external reference clock. This allows the CPU to run at a lower frequency (to conserve power) while the USB continues to monitor traffic.

Note that the FLL can not be used for generation of the system clocks while the PLL is supplying MCGPLLSCLK.

16.5 Initialization / Application Information

This section describes how to initialize and configure the MCG module in application. The following sections include examples on how to initialize the MCG and properly switch between the various available modes.

16.5.1 MCG Module Initialization Sequence

The MCG comes out of reset configured for FEI mode with the BDIV set for divide-by-2. The internal reference will stabilize in t_{irefst} microseconds before the FLL can acquire lock. As soon as the internal reference is stable, the FLL will acquire lock in $t_{fll\_acquire}$ milliseconds.

NOTE

> If the internal reference is not already trimmed, the BDIV value should not be changed to divide-by-1 without first trimming the internal reference. Failure to do so could result in the MCU running out of specification.

16.5.1.1 Initializing the MCG

Because the MCG comes out of reset in FEI mode, the only MCG modes which can be directly switched to upon reset are FEE, FBE, and FBI modes (see Figure 16-9). Reaching any of the other modes requires first configuring the MCG for one of these three initial modes. Care must be taken to check relevant status bits in the MCGSC register reflecting all configuration changes within each mode.

To change from FEI mode to FEE or FBE modes, follow this procedure:

1. Enable the external clock source by setting the appropriate bits in MCGC2.
2. If the RANGE bit (bit 5) in MCGC2 is set, set DIV32 in MCGC3 to allow access to the proper RDIV values.
3. Write to MCGC1 to select the clock mode.
   - If entering FEE mode, set RDIV appropriately, clear the IREFS bit to switch to the external reference, and leave the CLKS bits at %00 so that the output of the FLL is selected as the system clock source.
   - If entering FBE, clear the IREFS bit to switch to the external reference and change the CLKS bits to %10 so that the external reference clock is selected as the system clock source. The RDIV bits should also be set appropriately here according to the external reference frequency because although the FLL is bypassed, it is still on in FBE mode.
   - The internal reference can optionally be kept running by setting the IRCLKEN bit. This is useful if the application will switch back and forth between internal and external modes. For minimum power consumption, leave the internal reference disabled while in an external clock mode.
4. Once the proper configuration bits have been set, wait for the affected bits in the MCGSC register to be changed appropriately, reflecting that the MCG has moved into the proper mode.
   - If ERCLKEN was set in step 1 or the MCG is in FEE, FBE, PEE, PBE, or BLPE mode, and EREFS was also set in step 1, wait here for the OSCINIT bit to become set indicating that the

external clock source has finished its initialization cycles and stabilized. Typical crystal startup times are given in Appendix A, “Electrical Characteristics”.

- If in FEE mode, check to make sure the IREFST bit is cleared and the LOCK bit is set before moving on.
- If in FBE mode, check to make sure the IREFST bit is cleared, the LOCK bit is set, and the CLKST bits have changed to %10 indicating the external reference clock has been appropriately selected. Although the FLL is bypassed in FBE mode, it is still on and will lock in FBE mode.

5. Write to the MCGC4 register to determine the DCO output (MCGOUT) frequency range. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.
   - By default, with DMX32 cleared to 0, the FLL multiplier for the DCO output is 512. For greater flexibility, if a mid-range FLL multiplier of 1024 is desired instead, set the DRS[1:0] bits to %01 for a DCO output frequency of 33.55 MHz. If a high-range FLL multiplier of 1536 is desired instead, set the DRS[1:0] bits to %10 for a DCO output frequency of 50.33 MHz.
   - When using a 32.768 kHz external reference, if the maximum low-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %00 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 608 will be 19.92 MHz.
   - When using a 32.768 kHz external reference, if the maximum mid-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %01 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 1216 will be 39.85 MHz.
   - When using a 32.768 kHz external reference, if the maximum high-range DCO frequency that can be achieved with a 32.768 kHz reference is desired, set the DRS[1:0] bits to %10 and set the DMX32 bit to 1. The resulting DCO output (MCGOUT) frequency with the new multiplier of 1824 will be 59.77 MHz.
6. Wait for the LOCK bit in MCGSC to become set, indicating that the FLL has locked to the new multiplier value designated by the DRS and DMX32 bits.

NOTE

Setting DIV32 (bit 4) in MCGC3 is strongly recommended for FLL external modes when using a high frequency range (RANGE = 1) external reference clock. The DIV32 bit is ignored in all other modes.

To change from FEI clock mode to FBI clock mode, follow this procedure:

1. Change the CLKS bits in MCGC1 to %01 so that the internal reference clock is selected as the system clock source.
2. Wait for the CLKST bits in the MCGSC register to change to %01, indicating that the internal reference clock has been appropriately selected.

16.5.2 Using a 32.768 kHz Reference

In FEE and FBE modes, if using a 32.768 kHz external reference, at the default FLL multiplication factor of 512, the DCO output (MCGOUT) frequency is 16.78 MHz at high-range. If the DRS[1:0] bits are set to %01, the multiplication factor is doubled to 1024, and the resulting DCO output frequency is 33.55 Mhz at mid-range. If the DRS[1:0] bits are set to %10, the multiplication factor is set to 1536, and the resulting DCO output frequency is 50.33 Mhz at high-range. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.

Setting the DMX32 bit in MCGC4 to 1 increases the FLL multiplication factor to allow the 32.768 kHz reference to achieve its maximum DCO output frequency. When the DRS[1:0] bits are set to %00, the 32.768 kHz reference can achieve a high-range maximum DCO output of 19.92 MHz with a multiplier of 608. When the DRS[1:0] bits are set to %01, the 32.768 kHz reference can achieve a mid-range maximum DCO output of 39.85 MHz with a multiplier of 1216. When the DRS[1:0] bits are set to %10, the 32.768 kHz reference can achieve a high-range maximum DCO output of 59.77 MHz with a multiplier of 1824. Make sure that the resulting bus clock frequency does not exceed the maximum specified bus clock frequency of the device.

In FBI and FEI modes, setting the DMX32 bit is not recommended. If the internal reference is trimmed to a frequency above 32.768 kHz, the greater FLL multiplication factor could potentially push the microcontroller system clock out of specification and damage the part.

16.5.3 MCG Mode Switching

When switching between operational modes of the MCG, certain configuration bits must be changed in order to properly move from one mode to another. Each time any of these bits are changed (PLLS, IREFS, CLKS, or EREFS), the corresponding bits in the MCGSC register (PLLST, IREFST, CLKST, or OSCINIT) must be checked before moving on in the application software.

Additionally, care must be taken to ensure that the reference clock divider (RDIV) is set properly for the mode being switched to. For instance, in PEE mode, if using a 4 MHz crystal, RDIV must be set to %001 (divide-by-2) or %010 (divide -by-4) in order to divide the external reference down to the required frequency between 1 and 2 MHz.

If switching to FBE or FEE mode, first setting the DIV32 bit will ensure a proper reference frequency is sent to the FLL clock at all times.

In FBE, FEE, FBI, and FEI modes, at any time, the application can switch the FLL multiplication factor between 512, 1024, and 1536 with the DRS[1:0] bits in MCGC4. Writes to the DRS[1:0] bits will be ignored if LP=1 or PLLS=1.

The RDIV and IREFS bits should always be set properly before changing the PLLS bit so that the FLL or PLL clock has an appropriate reference clock frequency to switch to. The table below shows MCGOUT frequency calculations using RDIV, BDIV, and VDIV settings for each clock mode. The bus frequency is equal to MCGOUT divided by 2.

Table 16-12. MCGOUT Frequency Calculation Options

Clock Mode	f_{MCGOUT}[1]	Note
FEI (FLL engaged internal)	$(f_{int} * F) / B$	Typical f_{MCGOUT} = 16 MHz immediately after reset.
FEE (FLL engaged external)	$(f_{ext} / R * F) / B$	f_{ext} / R must be in the range of 31.25 kHz to 39.0625 kHz
FBE (FLL bypassed external)	f_{ext} / B	f_{ext} / R must be in the range of 31.25 kHz to 39.0625 kHz
FBI (FLL bypassed internal)	f_{int} / B	Typical f_{int} = 32 kHz
PEE (PLL engaged external)	$[(f_{ext} / R) * M] / B$	f_{ext} / R must be in the range of 1 MHz to 2 MHz
PBE (PLL bypassed external)	f_{ext} / B	f_{ext} / R must be in the range of 1 MHz to 2 MHz
BLPI (Bypassed low power internal)	f_{int} / B	
BLPE (Bypassed low power external)	f_{ext} / B	

[1] R is the reference divider selected by the RDIV bits, B is the bus frequency divider selected by the BDIV bits, F is the FLL factor selected by the DRS[1:0] and DMX32 bits, and M is the multiplier selected by the VDIV bits.

This section will include 3 mode switching examples using an 8 MHz external crystal. If using an external clock source less than 1 MHz, the MCG should not be configured for any of the PLL modes (PEE and PBE).

16.5.3.1 Example 1: Moving from FEI to PEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz

In this example, the MCG will move through the proper operational modes from FEI to PEE mode until the 8 MHz crystal reference frequency is set to achieve a bus frequency of 16 MHz. Because the MCG is in FEI mode out of reset, this example also shows how to initialize the MCG for PEE mode out of reset. First, the code sequence will be described. Then a flowchart will be included which illustrates the sequence.

1. First, FEI must transition to FBE mode:
   a) MCGC2 = 0x36 (%00110110)
      - BDIV (bits 7 and 6) set to %00, or divide-by-1
      - RANGE (bit 5) set to 1 because the frequency of 8 MHz is within the high frequency range
      - HGO (bit 4) set to 1 to configure the crystal oscillator for high gain operation
      - EREFS (bit 2) set to 1, because a crystal is being used
      - ERCLKEN (bit 1) set to 1 to ensure the external reference clock is active

b) Loop until OSCINIT (bit 1) in MCGSC is 1, indicating the crystal selected by the EREFS bit has been initialized.

c) Because RANGE = 1, set DIV32 (bit 4) in MCGC3 to allow access to the proper RDIV bits while in an FLL external mode.

d) MCGC1 = 0x98 (%10011000)
   - CLKS (bits 7 and 6) set to %10 in order to select external reference clock as system clock source
   - RDIV (bits 5-3) set to %011, or divide-by-256 because 8MHz / 256 = 31.25 kHz which is in the 31.25 kHz to 39.0625 kHz range required by the FLL
   - IREFS (bit 2) cleared to 0, selecting the external reference clock

e) Loop until IREFST (bit 4) in MCGSC is 0, indicating the external reference is the current source for the reference clock

f) Loop until CLKST (bits 3 and 2) in MCGSC is %10, indicating that the external reference clock is selected to feed MCGOUT

2. Then, FBE must transition either directly to PBE mode or first through BLPE mode and then to PBE mode:

   a) BLPE: If a transition through BLPE mode is desired, first set LP (bit 3) in MCGC2 to 1.

   b) BLPE/PBE: MCGC3 = 0x58 (%01011000)
      - PLLS (bit 6) set to 1, selects the PLL. At this time, with an RDIV value of %011, the FLL reference divider of 256 is switched to the PLL reference divider of 8 (see Table 16-3), resulting in a reference frequency of 8 MHz/ 8 = 1 MHz. In BLPE mode,changing the PLLS bit only prepares the MCG for PLL usage in PBE mode
      - DIV32 (bit 4) still set at 1. Because the MCG is in a PLL mode, the DIV32 bit is ignored. Keeping it set at 1 makes transitions back into an FLL external mode easier.
      - VDIV (bits 3-0) set to %1000, or multiply-by-32 because 1 MHz reference * 32= 32MHz. In BLPE mode, the configuration of the VDIV bits does not matter because the PLL is disabled. Changing them only sets up the multiply value for PLL usage in PBE mode

   c) BLPE: If transitioning through BLPE mode, clear LP (bit 3) in MCGC2 to 0 here to switch to PBE mode

   d) PBE: Loop until PLLST (bit 5) in MCGSC is set, indicating that the current source for the PLLS clock is the PLL

   e) PBE: Then loop until LOCK (bit 6) in MCGSC is set, indicating that the PLL has acquired lock

3. Lastly, PBE mode transitions into PEE mode:

   a) MCGC1 = 0x18 (%00011000)
      - CLKS (bits7 and 6) in MCGSC1 set to %00 in order to select the output of the PLL as the system clock source

b) Loop until CLKST (bits 3 and 2) in MCGSC are %11, indicating that the PLL output is selected to feed MCGOUT in the current clock mode

   – Now, With an RDIV of divide-by-8, a BDIV of divide-by-1, and a VDIV of multiply-by-32, MCGOUT = [(8 MHz / 8) * 32] / 1 = 32 MHz, and the bus frequency is MCGOUT / 2, or 16 MHz

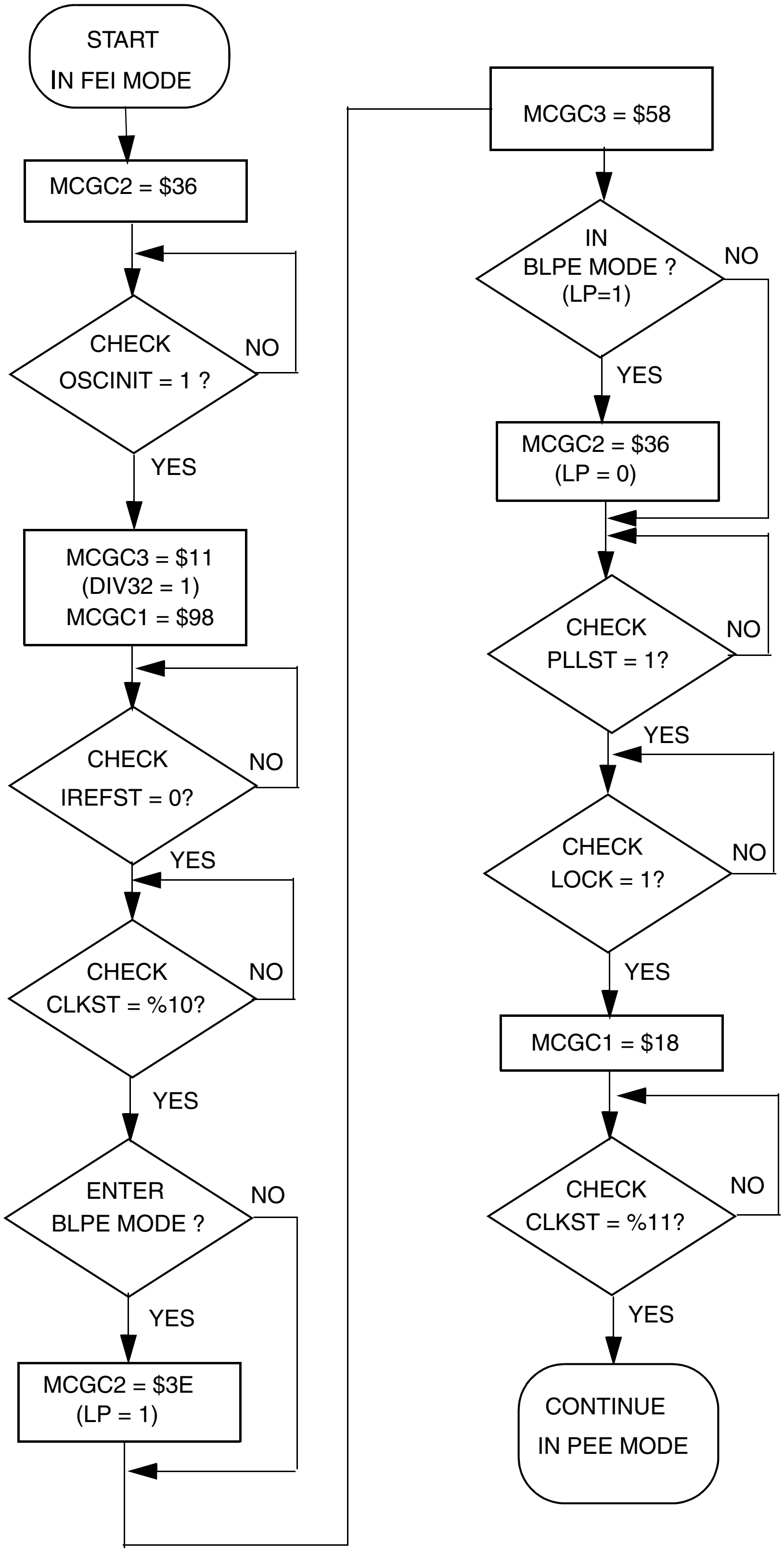


Figure 16-10. Flowchart of FEI to PEE Mode Transition using an 8 MHz crystal

16.5.3.2 Example 2: Moving from PEE to BLPI Mode: Bus Frequency =16 kHz

In this example, the MCG will move through the proper operational modes from PEE mode with an 8MHz crystal configured for an 16 MHz bus frequency (see previous example) to BLPI mode with a 16 kHz bus frequency.First, the code sequence will be described. Then a flowchart will be included which illustrates the sequence.

1. First, PEE must transition to PBE mode:

   a) MCGC1 = 0x98 (%10011000)
      – CLKS (bits 7 and 6) set to %10 in order to switch the system clock source to the external reference clock

   b) Loop until CLKST (bits 3 and 2) in MCGSC are %10, indicating that the external reference clock is selected to feed MCGOUT

2. Then, PBE must transition either directly to FBE mode or first through BLPE mode and then to FBE mode:

   a) BLPE: If a transition through BLPE mode is desired, first set LP (bit 3) in MCGC2 to 1

   b) BLPE/FBE: MCGC3 = 0x18(%00011000)
      – PLLS (bit 6) clear to 0 to select the FLL. At this time, with an RDIV value of %011, the PLL reference divider of 8 is switched to an FLL divider of 256 (see Table 16-2), resulting in a reference frequency of 8 MHz / 256 = 31.25 kHz. If RDIV was not previously set to %011 (necessary to achieve required 31.25-39.06 kHz FLL reference frequency with an 8 MHz external source frequency), it must be changed prior to clearing the PLLS bit. In BLPE mode,changing this bit only prepares the MCG for FLL usage in FBE mode. With PLLS = 0, the VDIV value does not matter.
      – DIV32 (bit 4) set to 1 (if previously cleared), automatically switches RDIV bits to the proper reference divider for the FLL clock (divide-by-256)

   c) BLPE: If transitioning through BLPE mode, clear LP (bit 3) in MCGC2 to 0 here to switch to FBE mode

   d) FBE: Loop until PLLST (bit 5) in MCGSC is clear, indicating that the current source for the PLLS clock is the FLL

   e) FBE: Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has acquired lock. Although the FLL is bypassed in FBE mode, it is still enabled and running.

3. Next, FBE mode transitions into FBI mode:

   a) MCGC1 = 0x5C (%01011100)
      – CLKS (bits7 and 6) in MCGSC1 set to %01 in order to switch the system clock to the internal reference clock

- – IREFS (bit 2) set to 1 to select the internal reference clock as the reference clock source
- – RDIV (bits 5-3) remain unchanged because the reference divider does not affect the internal reference.

b) Loop until IREFST (bit 4) in MCGSC is 1, indicating the internal reference clock has been selected as the reference clock source

c) Loop until CLKST (bits 3 and 2) in MCGSC are %01, indicating that the internal reference clock is selected to feed MCGOUT

4. Lastly, FBI transitions into BLPI mode.

a) MCGC2 = 0x08 (%00001000)

- – LP (bit 3) in MCGSC is 1
- – RANGE, HGO, EREFS, ERCLKEN, and EREFSTEN bits are ignored when the IREFS bit (bit2) in MCGC is set. They can remain set, or be cleared at this point.

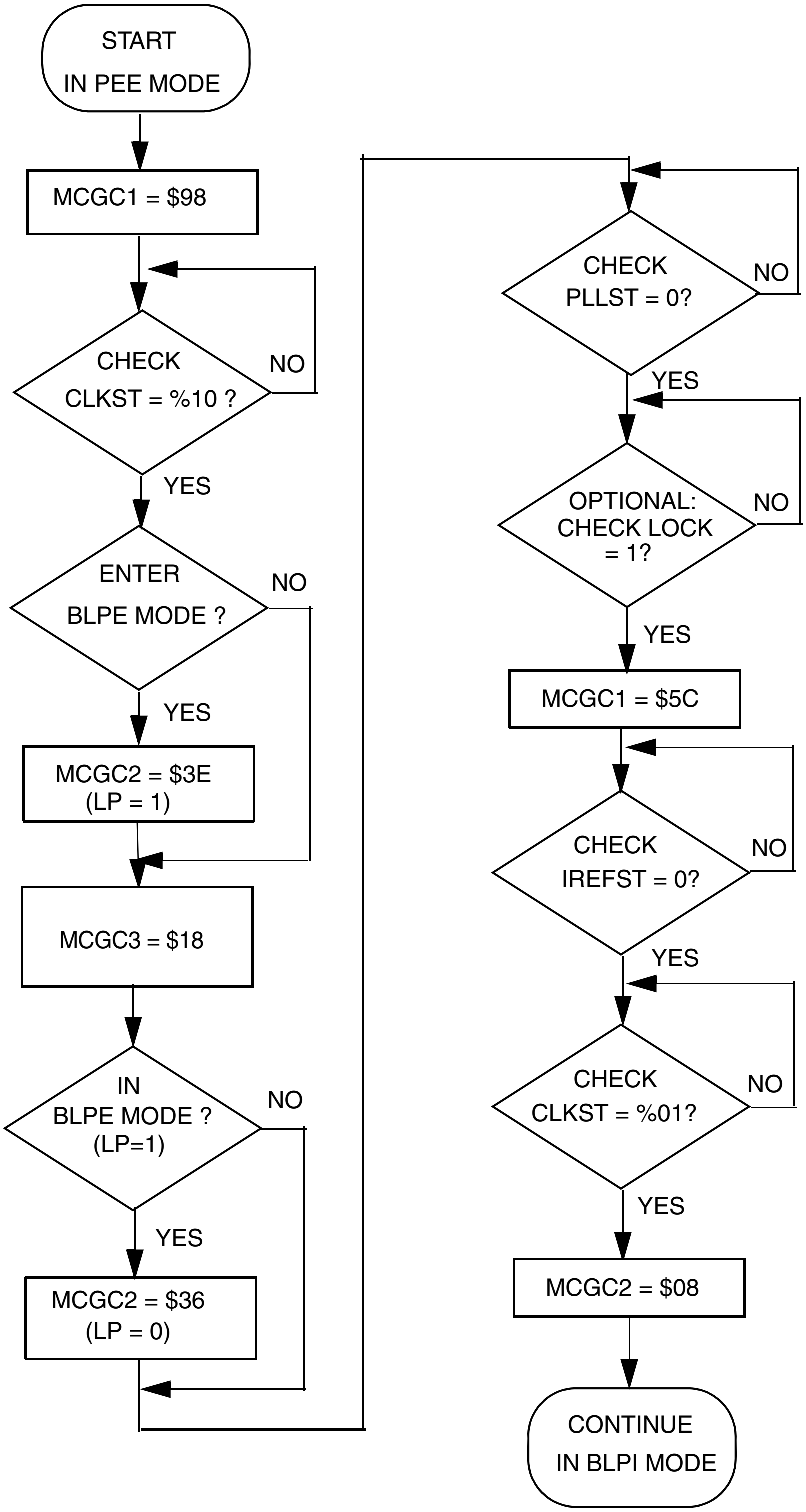


Figure 16-11. Flowchart of PEE to BLPI Mode Transition using an 8 MHz crystal

16.5.3.3 Example 3: Moving from BLPI to FEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz

In this example, the MCG will move through the proper operational modes from BLPI mode at a 16 kHz bus frequency running off of the internal reference clock (see previous example) to FEE mode using an 8MHz crystal configured for a 16 MHz bus frequency. First, the code sequence will be described. Then a flowchart will be included which illustrates the sequence.

1. First, BLPI must transition to FBI mode.
   a) MCGC2 = 0x00 (%00000000)
      - LP (bit 3) in MCGSC is 0

   b) Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has acquired lock. Although the FLL is bypassed in FBI mode, it is still enabled and running.
2. Next, FBI will transition to FEE mode.
   a) MCGC2 = 0x36 (%00110110)
      - RANGE (bit 5) set to 1 because the frequency of 8 MHz is within the high frequency range
      - HGO (bit 4) set to 1 to configure the crystal oscillator for high gain operation
      - EREFS (bit 2) set to 1, because a crystal is being used
      - ERCLKEN (bit 1) set to 1 to ensure the external reference clock is active

   b) Loop until OSCINIT (bit 1) in MCGSC is 1, indicating the crystal selected by the EREFS bit has been initialized.

   c) MCGC1 = 0x18 (%00011000)
      - CLKS (bits 7 and 6) set to %00 in order to select the output of the FLL as system clock source
      - RDIV (bits 5-3) remain at %011, or divide-by-256 for a reference of 8 MHz / 256 = 31.25 kHz.
      - IREFS (bit 1) cleared to 0, selecting the external reference clock

   d) Loop until IREFST (bit 4) in MCGSC is 0, indicating the external reference clock is the current source for the reference clock

   e) Optionally, loop until LOCK (bit 6) in the MCGSC is set, indicating that the FLL has reacquired lock.

   f) Loop until CLKST (bits 3 and 2) in MCGSC are %00, indicating that the output of the FLL is selected to feed MCGOUT

   g) Now, with a 31.25 kHz reference frequency, a fixed DCO multiplier of 512, and a bus divider of 1, MCGOUT = 31.25 kHz * 512 / 1 = 16 MHz. Therefore, the bus frequency is 8 MHz.

   h) At this point, by default, the DRS[1:0] bits in MCGC4 are set to %00 and DMX32 in MCGC4 is cleared to 0. If a bus frequency of 16MHz is desired instead, set the DRS[1:0] bits to $01 to switch the FLL multiplication factor from 512 to 1024 and loop until LOCK (bit 6) in MCGSC is set, indicating that the FLL has reacquired LOCK. To return the bus frequency to 8 MHz, set the DRS[1:0] bits to %00 again, and the FLL multiplication factor will switch back to 512. Then loop again until the LOCK bit is set.

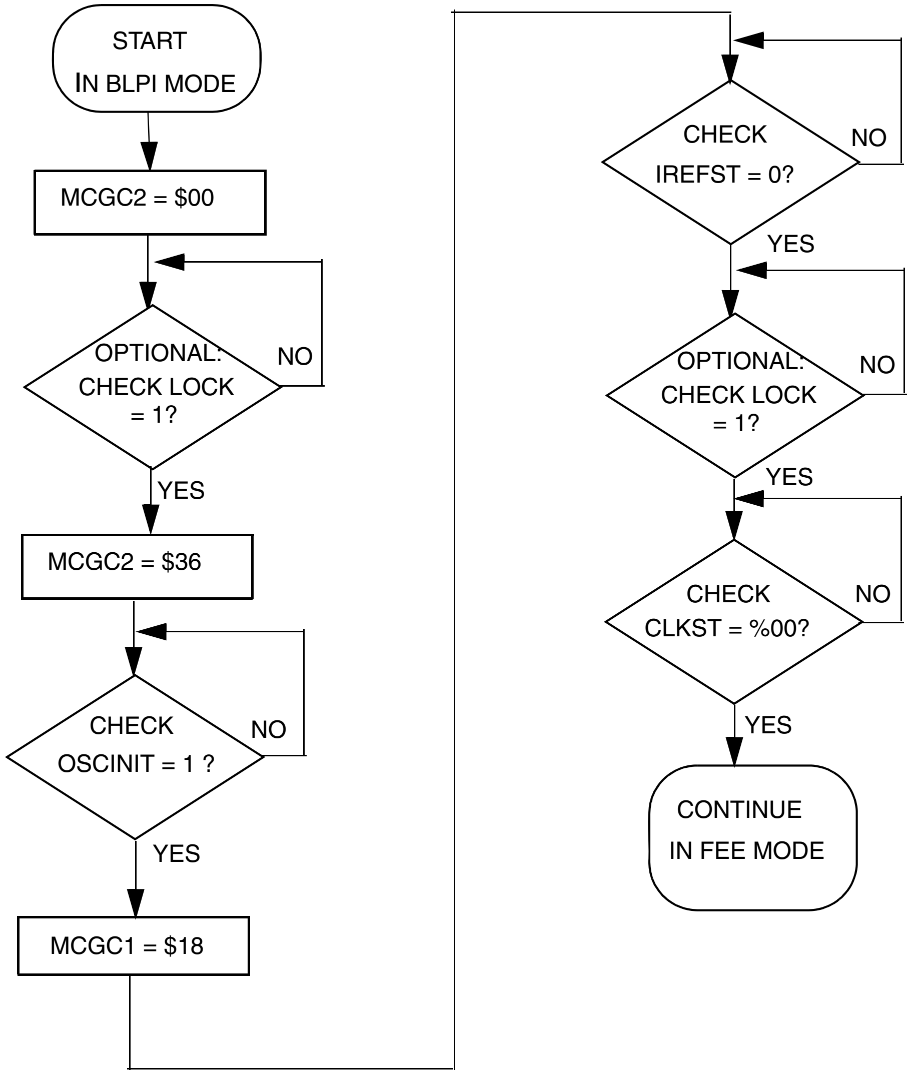


Figure 16-12. Flowchart of BLPI to FEE Mode Transition using an 8 MHz crystal

16.5.4 Calibrating the Internal Reference Clock (IRC)

The IRC is calibrated by writing to the MCGTRM register first, then using the FTRIM bit to “fine tune” the frequency. We will refer to this total 9-bit value as the trim value, ranging from 0x000 to 0x1FF, where the FTRIM bit is the LSB.

The trim value after reset is the factory trim value unless the device resets into any BDM mode in which case it is 0x800. Writing a larger value will decrease the frequency and smaller values will increase the frequency. The trim value is linear with the period, except that slight variations in wafer fab processing produce slight non-linearities between trim value and period. These non-linearities are why an iterative trimming approach to search for the best trim value is recommended. In Example 4: Internal Reference Clock Trim later in this section, this approach will be demonstrated.

If a user specified trim value has been found for a device (to replace the factory trim value), this value can be stored in FLASH memory to save the value. If power is removed from the device, the IRC can easily be re-trimmed to the user specified value by copying the saved value from FLASH to the MCG registers. Freescale identifies recommended FLASH locations for storing the trim value for each MCU. Consult the memory map in the data sheet for these locations.

16.5.4.1 Example 4: Internal Reference Clock Trim

For applications that require a user specified tight frequency tolerance, a trimming procedure is provided that will allow a very accurate internal clock source. This section outlines one example of trimming the internal oscillator. Many other possible trimming procedures are valid and can be used.

In the example below, the MCG trim will be calibrated for the 9-bit MCGTRM and FTRIM collective value. This value will be referred to as TRMVAL.

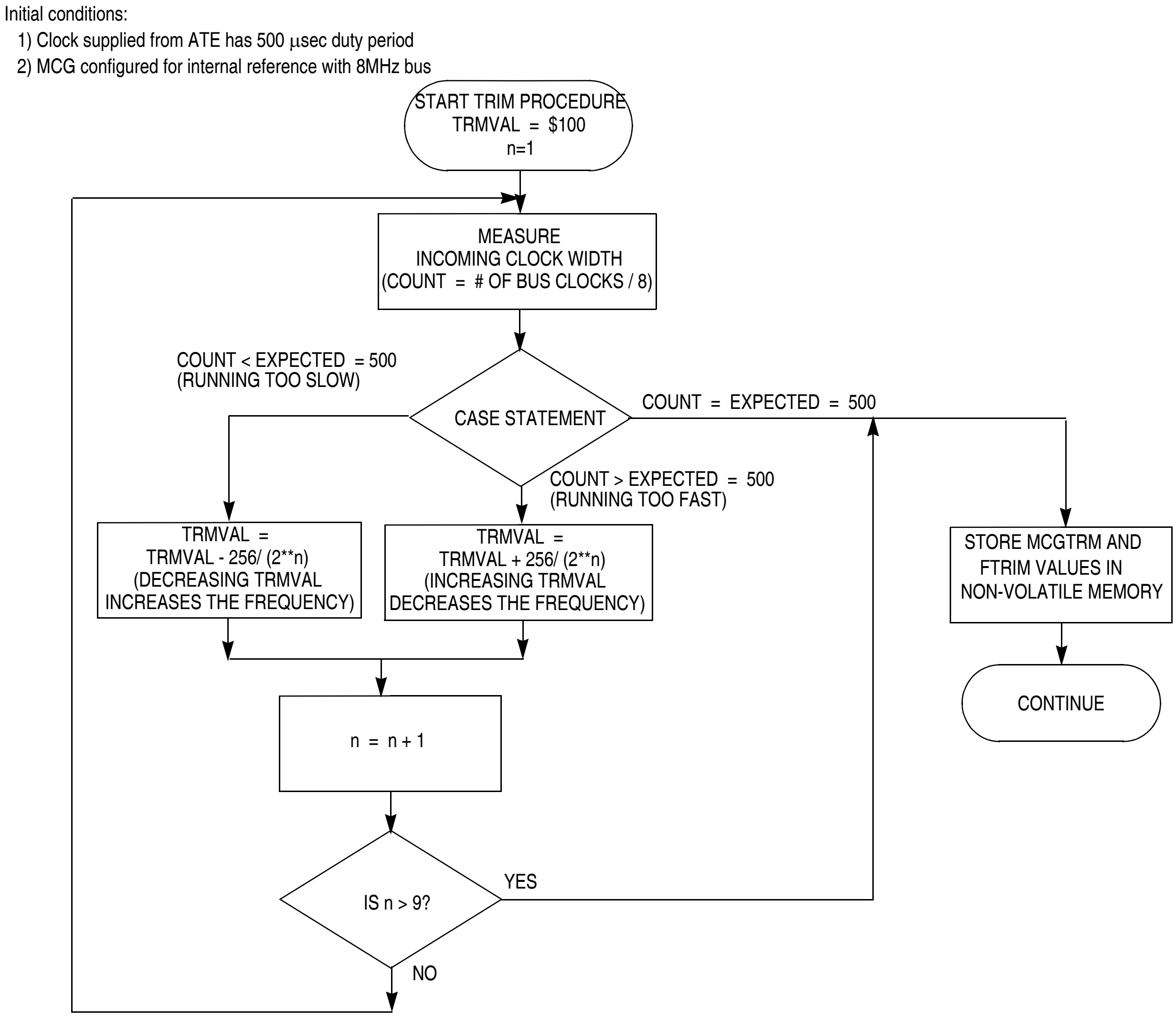


Figure 16-13. Trim Procedure

In this particular case, the MCU has been attached to a PCB and the entire assembly is undergoing final test with automated test equipment. A separate signal or message is provided to the MCU operating under user provided software control. The MCU initiates a trim procedure as outlined in Figure 16-13 while the tester supplies a precision reference signal.

If the intended bus frequency is near the maximum allowed for the device, it is recommended to trim using a reference divider value (RDIV setting) of twice the final value. After the trim procedure is complete, the reference divider can be restored. This will prevent accidental overshoot of the maximum clock frequency.

Chapter 17
Rapid GPIO (RGPIO)

17.1 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.

17.1.1 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 17-1. The details of the pin muxing and pad logic are device-specific.

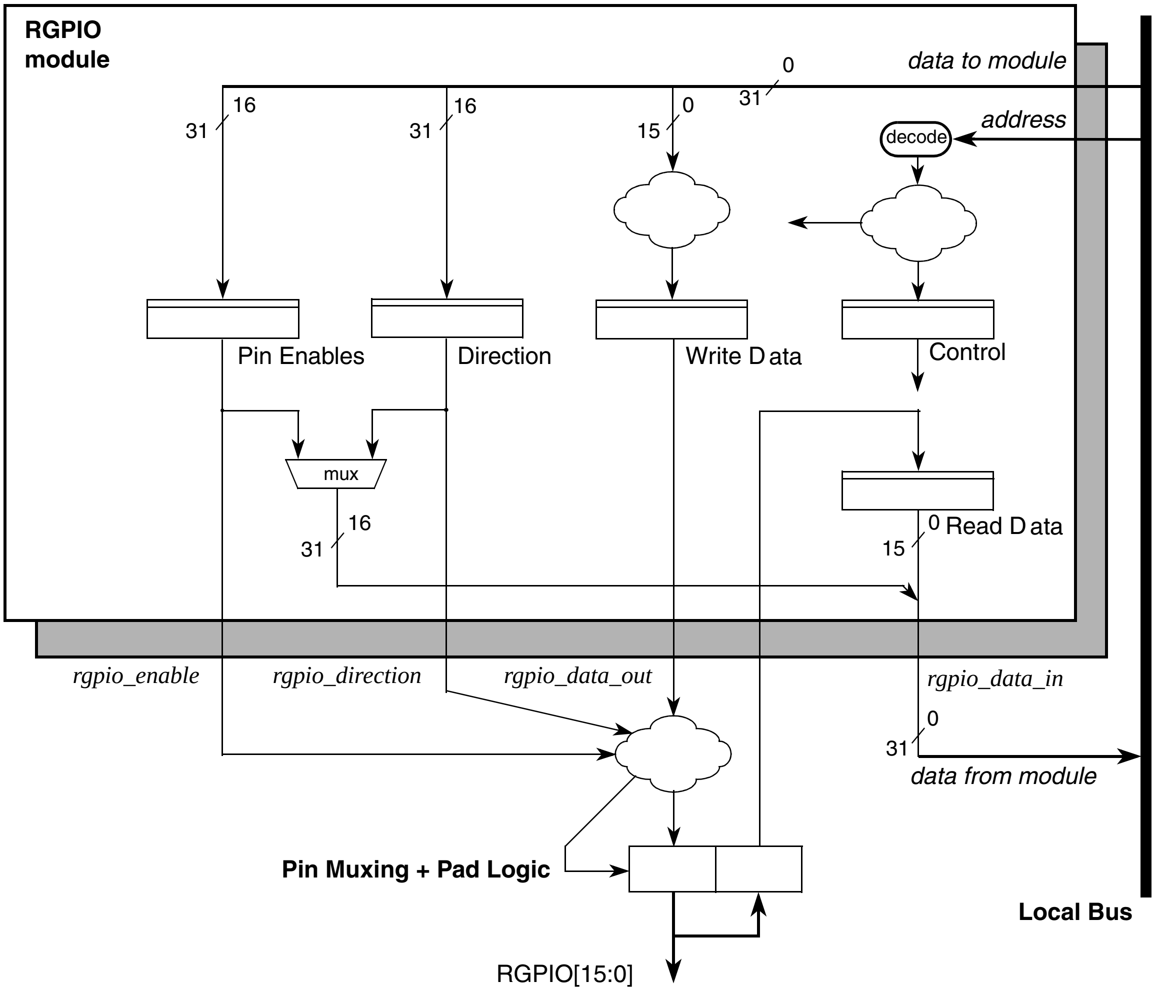


Figure 17-1. RGPIO Block Diagram

17.1.2 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)

17.1.3 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.

17.2 External Signal Description

17.2.1 Overview

As shown in Figure 17-1, 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 17-1.

Table 17-1. RGPIO Module External I/O Signals

Signal Name	Type	Description
RGPIO[15:0]	I/O	RGPIO Data Input/Output

17.2.2 Detailed Signal Descriptions

Table 17-2 provides descriptions of the RGPIO module's input and output signals.

Table 17-2. 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.	
		State Meaning	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.
		Timing	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.

17.3 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 programming model can be referenced using any operand size access (byte, word, longword). 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 17-3. RGPIO *Write* Memory Map

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	16	W	0x0000	17.3.1/17-4
0x02	RGPIO Write Data Register (RGPIO_DATA)	16	W	0x0000	17.3.2/17-5
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	16	W	0x0000	17.3.3/17-6
0x06	RGPIO Write Data Clear Register (RGPIO_CLR)	16	W	N/A	17.3.4/17-6
0x0A	RGPIO Write Data Set Register (RGPIO_SET)	16	W	N/A	17.3.5/17-7
0x0E	RGPIO Write Data Toggle Register (RGPIO_TOG)	16	W	N/A	17.3.6/17-7

Table 17-4. RGPIO *Read* Memory Map

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	RGPIO data direction register (RGPIO_DIR)	16	R	0x0000	17.3.1/17-4
0x02	RGPIO write data register (RGPIO_DATA)	16	R	0x0000	17.3.2/17-5
0x04	RGPIO pin enable register (RGPIO_ENB)	16	R	0x0000	17.3.3/17-6
0x06	RGPIO write data register (RGPIO_DATA)	16	R	0x0000	17.3.2/17-5
0x08	RGPIO data direction register (RGPIO_DIR)	16	R	0x0000	17.3.1/17-4
0x0A	RGPIO write data register (RGPIO_DATA)	16	R	0x0000	17.3.2/17-5
0x0C	RGPIO data direction register (RGPIO_DIR)	16	R	0x0000	17.3.1/17-4
0x0E	RGPIO write data register (RGPIO_DATA)	16	R	0x0000	17.3.2/17-5

17.3.1 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.

Offset: RGPIO_Base + 0x0 (RGPIO_DIR)
RGPIO_Base + 0x8
RGPIO_Base + 0xC

Access: Read/write
Read-only
Read-only

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

Figure 17-2. RGPIO Data Direction Register (RGPIO_DIR)

Table 17-5. RGPIO_DIR Field Descriptions

Field	Description
15–0 DIR	RGPIO data direction. 0 A properly-enabled RGPIO pin is configured as an input 1 A properly-enabled RGPIO pin is configured as an output

17.3.2 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.

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

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

Figure 17-3. RGPIO Data Register (RGPIO_DATA)

Table 17-6. 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

17.3.3 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

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

Figure 17-4. RGPIO Enable Register (RGPIO_ENB)

Table 17-7. RGPIO_ENB Field Descriptions

Field	Description
15–0 ENB	RGPIO enable. 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

17.3.4 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.

Offset: RGPIO_Base + 0x6 (RGPIO_CLR) Access: Write-only

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

Figure 17-5. RGPIO Clear Data Register (RGPIO_CLR)

Table 17-8. RGPIO_CLR Field Descriptions

Field	Description
15–0 CLR	RGPIO clear data. 0 Clears the corresponding bit in the RGPIO_DATA register 1 No effect

17.3.5 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.

Offset: RGPIO_Base + 0xA (RGPIO_SET) Access: Write-only

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

Figure 17-6. RGPIO Set Data Register (RGPIO_SET)

Table 17-9. RGPIO_SET Field Descriptions

Field	Description
15–0 SET	RGPIO set data. 0 No effect 1 Sets the corresponding bit in the RGPIO_DATA register

17.3.6 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.

Offset: RGPIO_Base + 0xE (RGPIO_TOG) Access: Write-only

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

Figure 17-7. RGPIO Toggle Data Register (RGPIO_TOG)

Table 17-10. RGPIO_TOG Field Descriptions

Field	Description
15–0 TOG	RGPIO toggle data. 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

17.4 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).

17.5 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)

17.6 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).

17.6.1 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 17-11. 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 17-11. 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
bchg	(1/24) × *f* MHz	2.083 MHz	1.00x	(1/14) × *f* MHz	3.571 MHz	1.71x
set+clr (+toggle)	(1/12) × *f* MHz	4.167 MHz	2.00x	(1/8) × *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.

17.6.2 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 17-8.

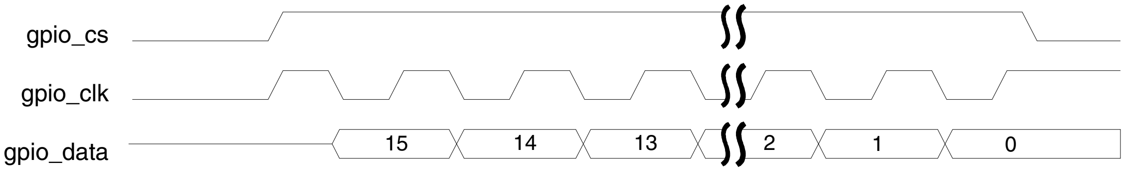


Figure 17-8. GPIO SPI Example Timing Diagram

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 17-9.

```
# 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
                              align   4

                      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 17-9. 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 17-12.

Table 17-12. 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 18
Serial Communications Interface (SCIV4)

18.1 Introduction

18.1.1 Features

Features of SCI module include:

- Full-duplex, standard non-return-to-zero (NRZ) format
- Double-buffered transmitter and receiver with separate enables
- Programmable baud rates (13-bit modulo divider)
- Interrupt-driven or polled operation:
    - Transmit data register empty and transmission complete
    - Receive data register full
    - Receive overrun, parity error, framing error, and noise error
    - Idle receiver detect
    - Active edge on receive pin
    - Break detect supporting LIN
- Hardware parity generation and checking
- Programmable 8-bit or 9-bit character length
- Receiver wakeup by idle-line or address-mark
- Optional 13-bit break character generation / 11-bit break character detection
- Selectable transmitter output polarity

18.1.2 Modes of Operation

See Section 18.3, “Functional Description,” for details concerning SCI operation in these modes:

- 8- and 9-bit data modes
- Stop mode operation
- Loop mode
- Single-wire mode

18.1.3 Block Diagram

Figure 18-1 shows the transmitter portion of the SCI.

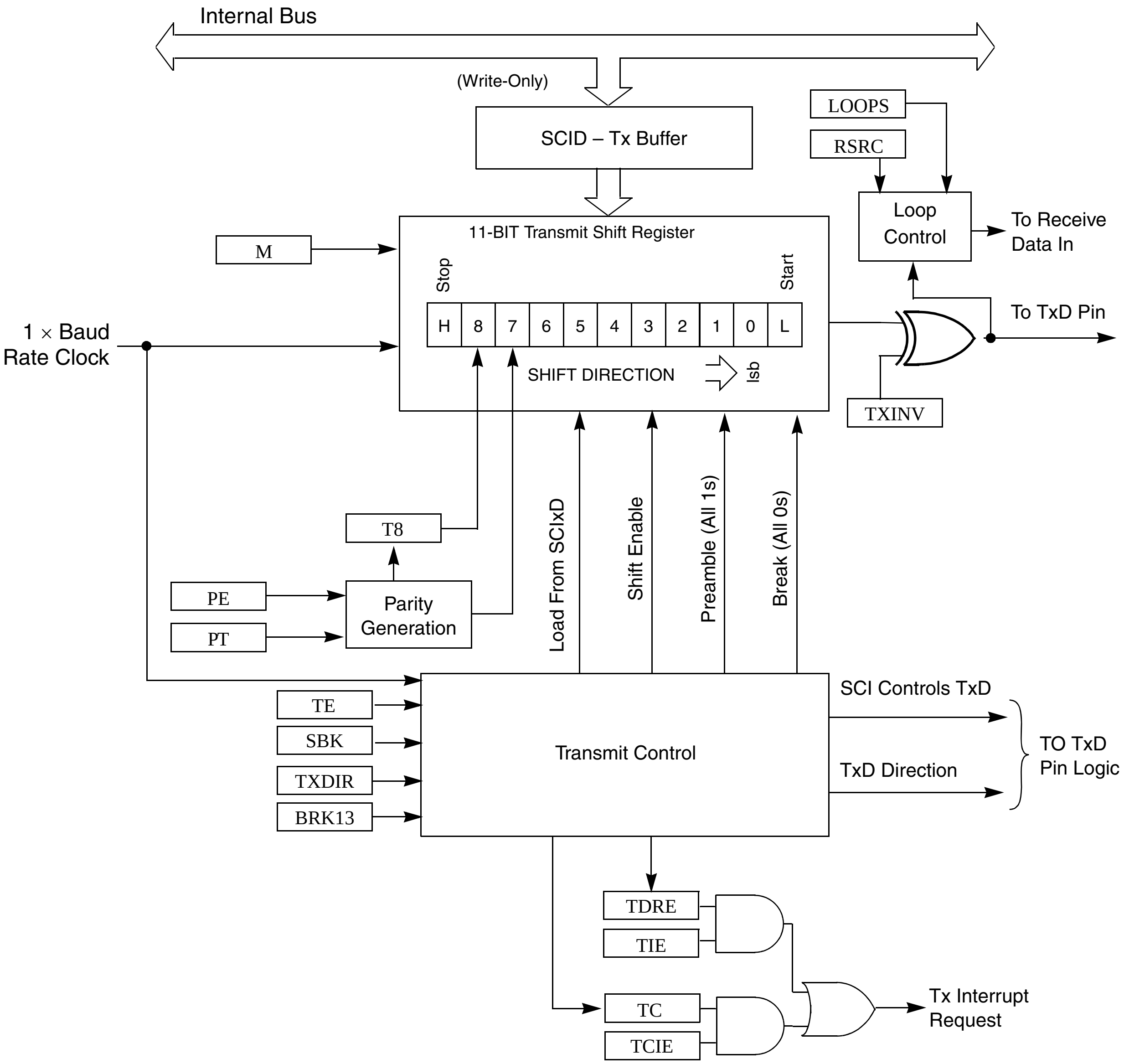


Figure 18-1. SCI Transmitter Block Diagram

Figure 18-2 shows the receiver portion of the SCI.

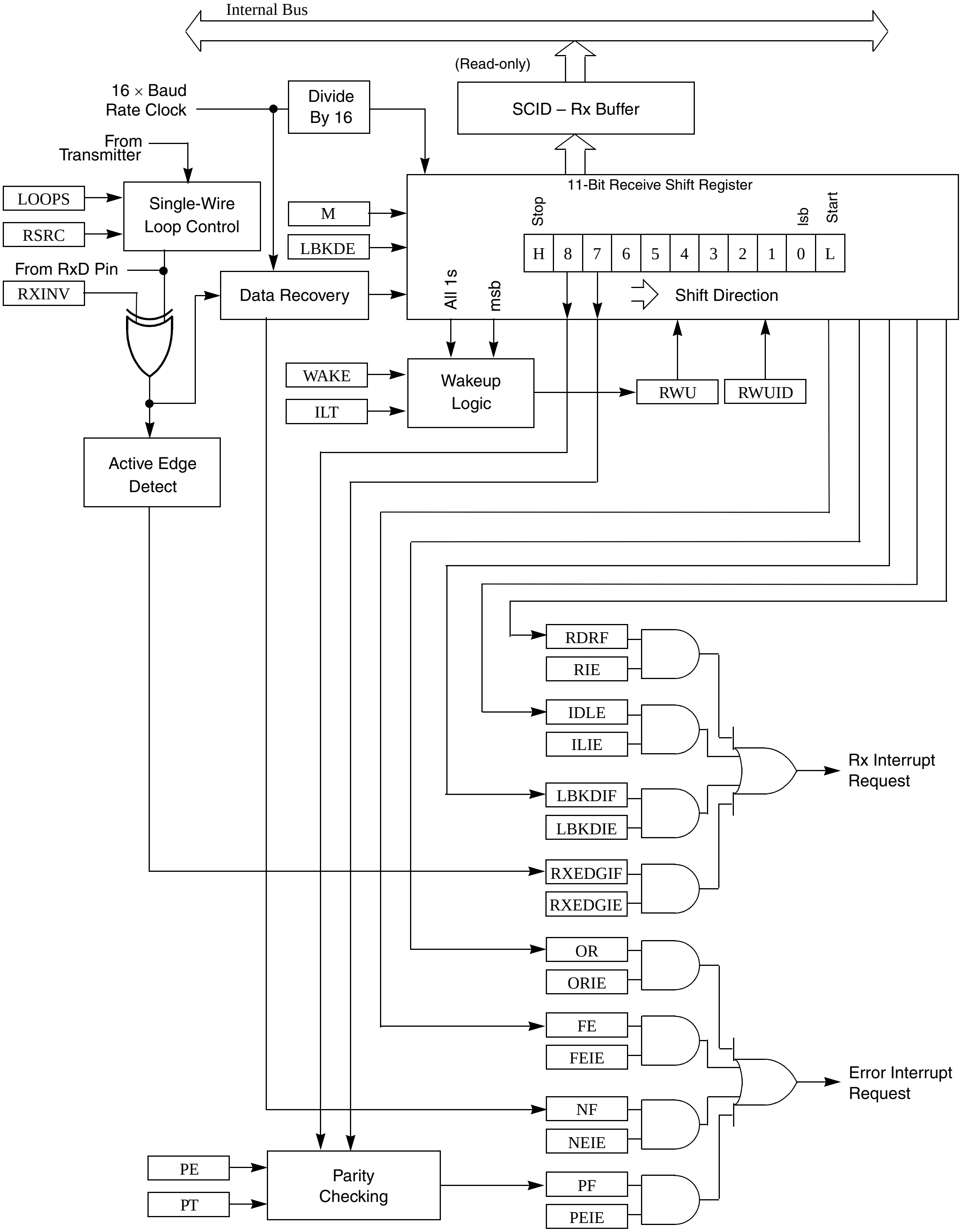


Figure 18-2. SCI Receiver Block Diagram

18.2 Register Definition

The SCI has eight 8-bit registers to control baud rate, select SCI options, report SCI status, and for transmit/receive data.

Refer to the direct-page register summary in the memory chapter of this document or the absolute address assignments for all SCI 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.

18.2.1 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 [SBR12:SBR0], 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).

	7	6	5	4	3	2	1	0
R	LBKDIE	RXEDGIE	0	SBR[12:8]				
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-3. SCI Baud Rate Register (SCIxBDH)

Table 18-1. SCIxBDH Field Descriptions

Field	Description
7 LBKDIE	LIN Break Detect Interrupt Enable (for LBKDIF) 0 Hardware interrupts from LBKDIF disabled (use polling). 1 Hardware interrupt requested when LBKDIF flag is 1.
6 RXEDGIE	RxD Input Active Edge Interrupt Enable (for RXEDGIF) 0 Hardware interrupts from RXEDGIF disabled (use polling). 1 Hardware interrupt requested when RXEDGIF flag is 1.
4–0 SBR[12:8]	Baud Rate Modulo Divisor. The 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 – 8191, the SCI baud rate equals BUSCLK/(16×BR). See also BR bits in Table 18-2.

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

Figure 18-4. SCI Baud Rate Register (SCIxBDL)

Table 18-2. SCIxBDL Field Descriptions

Field	Description
7–0 SBR[7:0]	Baud Rate Modulo Divisor. These 13 bits in SBR[12:0] are referred to collectively as BR, and they set the modulo divide rate for the SCI baud rate generator. When BR is cleared, the SCI baud rate generator is disabled to reduce supply current. When BR is 1 – 8191, the SCI baud rate equals BUSCLK/(16×BR). See also BR bits in Table 18-1.

18.2.2 SCI Control Register 1 (SCIxC1)

This read/write register controls various optional features of the SCI system.

	7	6	5	4	3	2	1	0
R W	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

Figure 18-5. SCI Control Register 1 (SCIxC1)

Table 18-3. SCIxC1 Field Descriptions

Field	Description
7 LOOPS	Loop Mode Select. Selects between loop back modes and normal 2-pin full-duplex modes. When LOOPS is set, the transmitter output is internally connected to the receiver input. 0 Normal operation — RxD and TxD use separate pins. 1 Loop mode or single-wire mode where transmitter outputs are internally connected to receiver input. (See RSRC bit.) RxD pin is not used by SCI.
6 SCISWAI	SCI Stops in Wait Mode 0 SCI clocks continue to run in wait mode so the SCI can be the source of an interrupt that wakes up the CPU. 1 SCI clocks freeze while CPU is in wait mode.
5 RSRC	Receiver Source Select. This bit has no meaning or effect unless the LOOPS bit is set to 1. When LOOPS is set, the receiver input is internally connected to the TxD pin and RSRC determines whether this connection is also connected to the transmitter output. 0 Provided LOOPS is set, RSRC is cleared, selects internal loop back mode and the SCI does not use the RxD pins. 1 Single-wire SCI mode where the TxD pin is connected to the transmitter output and receiver input.
4 M	9-Bit or 8-Bit Mode Select 0 Normal — start + 8 data bits (lsb first) + stop. 1 Receiver and transmitter use 9-bit data characters start + 8 data bits (lsb first) + 9th data bit + stop.
3 WAKE	Receiver Wakeup Method Select. Refer to Section 18.3.3.2, “Receiver Wakeup Operation” for more information. 0 Idle-line wakeup. 1 Address-mark wakeup.
2 ILT	Idle Line Type Select. Setting this bit to 1 ensures that the stop bit and logic 1 bits at the end of a character do not count toward the 10 or 11 bit times of logic high level needed by the idle line detection logic. Refer to Section 18.3.3.2.1, “Idle-Line Wakeup” for more information. 0 Idle character bit count starts after start bit. 1 Idle character bit count starts after stop bit.

Table 18-3. SCIxC1 Field Descriptions (continued)

Field	Description
1 PE	Parity Enable. Enables hardware parity generation and checking. When parity is enabled, the most significant bit (msb) of the data character (eighth or ninth data bit) is treated as the parity bit. 0 No hardware parity generation or checking. 1 Parity enabled.
0 PT	Parity Type. Provided parity is enabled (PE = 1), this bit selects even or odd parity. Odd parity means the total number of 1s in the data character, including the parity bit, is odd. Even parity means the total number of 1s in the data character, including the parity bit, is even. 0 Even parity. 1 Odd parity.

18.2.3 SCI Control Register 2 (SCIxC2)

This register can be read or written at any time.

	7	6	5	4	3	2	1	0
R W	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

Figure 18-6. SCI Control Register 2 (SCIxC2)

Table 18-4. SCIxC2 Field Descriptions

Field	Description
7 TIE	Transmit Interrupt Enable (for TDRE) 0 Hardware interrupts from TDRE disabled (use polling). 1 Hardware interrupt requested when TDRE flag is 1.
6 TCIE	Transmission Complete Interrupt Enable (for TC) 0 Hardware interrupts from TC disabled (use polling). 1 Hardware interrupt requested when TC flag is 1.
5 RIE	Receiver Interrupt Enable (for RDRF) 0 Hardware interrupts from RDRF disabled (use polling). 1 Hardware interrupt requested when RDRF flag is 1.
4 ILIE	Idle Line Interrupt Enable (for IDLE) 0 Hardware interrupts from IDLE disabled (use polling). 1 Hardware interrupt requested when IDLE flag is 1.

Table 18-4. SCIxC2 Field Descriptions (continued)

Field	Description
3 TE	Transmitter Enable 0 Transmitter off. 1 Transmitter on. TE must be 1 to use the SCI transmitter. When TE is set, the SCI forces the TxD pin to act as an output for the SCI system. When the SCI is configured for single-wire operation (LOOPS = RSRC = 1), TXDIR controls the direction of traffic on the single SCI communication line (TxD pin). TE can also queue an idle character by clearing TE then setting TE while a transmission is in progress. Refer to Section 18.3.2.1, “Send Break and Queued Idle” for more details. When TE is written to 0, the transmitter keeps control of the port TxD pin until any data, queued idle, or queued break character finishes transmitting before allowing the pin to revert to a general-purpose I/O pin.
2 RE	Receiver Enable. When the SCI receiver is off, the RxD pin reverts to being a general-purpose port I/O pin. If LOOPS is set the RxD pin reverts to being a general-purpose I/O pin even if RE is set. 0 Receiver off. 1 Receiver on.
1 RWU	Receiver Wakeup Control. This bit can be written to 1 to place the SCI receiver in a standby state where it waits for automatic hardware detection of a selected wakeup condition. The wakeup condition is an idle line between messages (WAKE = 0, idle-line wakeup) or a logic 1 in the most significant data bit in a character (WAKE = 1, address-mark wakeup). Application software sets RWU and (normally) a selected hardware condition automatically clears RWU. Refer to Section 18.3.3.2, “Receiver Wakeup Operation,” for more details. 0 Normal SCI receiver operation. 1 SCI receiver in standby waiting for wakeup condition.
0 SBK	Send Break. Writing a 1 and then a 0 to SBK queues a break character in the transmit data stream. Additional break characters of 10 or 11 (13 or 14 if BRK13 = 1) bit times of logic 0 are queued as long as SBK is set. Depending on the timing of the set and clear of SBK relative to the information currently being transmitted, a second break character may be queued before software clears SBK. Refer to Section 18.3.2.1, “Send Break and Queued Idle” for more details. 0 Normal transmitter operation. 1 Queue break character(s) to be sent.

18.2.4 SCI Status Register 1 (SCIxS1)

This register has eight read-only status flags. Writes have no effect. Special software sequences (which do not involve writing to this register) clear these status flags.

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	1	1	0	0	0	0	0	0

Figure 18-7. SCI Status Register 1 (SCIxS1)

Table 18-5. SCIxS1 Field Descriptions

Field	Description
7 TDRE	Transmit Data Register Empty Flag. TDRE is set out of reset and when a transmit data value transfers from the transmit data buffer to the transmit shifter, leaving room for a new character in the buffer. To clear TDRE, read SCIxS1 with TDRE set and then write to the SCI data register (SCIxD). 0 Transmit data register (buffer) full. 1 Transmit data register (buffer) empty.
6 TC	Transmission Complete Flag. TC is set out of reset and when TDRE is set and no data, preamble, or break character is being transmitted. 0 Transmitter active (sending data, a preamble, or a break). 1 Transmitter idle (transmission activity complete). TC is cleared automatically by reading SCIxS1 with TC set and then doing one of the following: • Write to the SCI data register (SCIxD) to transmit new data • Queue a preamble by changing TE from 0 to 1 • Queue a break character by writing 1 to SBK in SCIxC2
5 RDRF	Receive Data Register Full Flag. RDRF becomes set when a character transfers from the receive shifter into the receive data register (SCIxD). To clear RDRF, read SCIxS1 with RDRF set and then read the SCI data register (SCIxD). 0 Receive data register empty. 1 Receive data register full.
4 IDLE	Idle Line Flag. IDLE is set when the SCI receive line becomes idle for a full character time after a period of activity. When ILT is cleared, the receiver starts counting idle bit times after the start bit. If the receive character is all 1s, these bit times and the stop bit time count toward the full character time of logic high (10 or 11 bit times depending on the M control bit) needed for the receiver to detect an idle line. When ILT is set, the receiver doesn't start counting idle bit times until after the stop bit. The stop bit and any logic high bit times at the end of the previous character do not count toward the full character time of logic high needed for the receiver to detect an idle line. To clear IDLE, read SCIxS1 with IDLE set and then read the SCI data register (SCIxD). After IDLE has been cleared, it cannot become set again until after a new character has been received and RDRF has been set. IDLE is set only once even if the receive line remains idle for an extended period. 0 No idle line detected. 1 Idle line was detected.
3 OR	Receiver Overrun Flag. OR is set when a new serial character is ready to be transferred to the receive data register (buffer), but the previously received character has not been read from SCIxD yet. In this case, the new character (and all associated error information) is lost because there is no room to move it into SCIxD. To clear OR, read SCIxS1 with OR set and then read the SCI data register (SCIxD). 0 No overrun. 1 Receive overrun (new SCI data lost).
2 NF	Noise Flag. The advanced sampling technique used in the receiver takes seven samples during the start bit and three samples in each data bit and the stop bit. If any of these samples disagrees with the rest of the samples within any bit time in the frame, the flag NF is set at the same time as RDRF is set for the character. To clear NF, read SCIxS1 and then read the SCI data register (SCIxD). 0 No noise detected. 1 Noise detected in the received character in SCIxD.

Table 18-5. SCIxS1 Field Descriptions (continued)

Field	Description
1 FE	Framing Error Flag. FE is set at the same time as RDRF when the receiver detects a logic 0 where the stop bit was expected. This suggests the receiver was not properly aligned to a character frame. To clear FE, read SCIxS1 with FE set and then read the SCI data register (SCIxD). 0 No framing error detected. This does not guarantee the framing is correct. 1 Framing error.
0 PF	Parity Error Flag. PF is set at the same time as RDRF when parity is enabled (PE = 1) and the parity bit in the received character does not agree with the expected parity value. To clear PF, read SCIxS1 and then read the SCI data register (SCIxD). 0 No parity error. 1 Parity error.

18.2.5 SCI Status Register 2 (SCIxS2)

This register contains one read-only status flag.

	7	6	5	4	3	2	1	0
R	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-8. SCI Status Register 2 (SCIxS2)

Table 18-6. SCIxS2 Field Descriptions

Field	Description
7 LBKDIF	LIN Break Detect Interrupt Flag. LBKDIF is set when the LIN break detect circuitry is enabled and a LIN break character is detected. LBKDIF is cleared by writing a 1 to it. 0 No LIN break character has been detected. 1 LIN break character has been 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[1]	Receive Data Inversion. Setting this bit reverses the polarity of the received data input. 0 Receive data not inverted 1 Receive data inverted
3 RWUID	Receive Wake Up Idle Detect. RWUID controls whether the idle character that wakes up the receiver sets the IDLE bit. 0 During receive standby state (RWU = 1), the IDLE bit does not get set upon detection of an idle character. 1 During receive standby state (RWU = 1), the IDLE bit gets set upon detection of an idle character.
2 BRK13	Break Character Generation Length. BRK13 selects a longer transmitted break character length. Detection of a framing error is not affected by the state of this bit. 0 Break character is transmitted with length of 10 bit times (11 if M = 1) 1 Break character is transmitted with length of 13 bit times (14 if M = 1)

Table 18-6. SCIxS2 Field Descriptions (continued)

Field	Description
1 LBKDE	LIN Break Detection Enable. LBKDE selects a longer break character detection length. While LBKDE is set, framing error (FE) and receive data register full (RDRF) flags are prevented from setting. 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 the beginning of a valid start bit, and RAF is cleared automatically when the receiver detects an idle line. This status flag can be used to check whether an SCI character is being received before instructing the MCU to go to stop mode. 0 SCI receiver idle waiting for a start bit. 1 SCI receiver active (RxD input not idle).

[1] Setting RXINV inverts the RxD input for all cases: data bits, start and stop bits, break, and idle.

When using an internal oscillator in a LIN system, it is necessary to raise the break detection threshold one bit time. Under the worst case timing conditions allowed in LIN, it is possible that a 0x00 data character can appear to be 10.26 bit times long at a slave running 14% faster than the master. This would trigger normal break detection circuitry designed to detect a 10-bit break symbol. When the LBKDE bit is set, framing errors are inhibited and the break detection threshold changes from 10 bits to 11 bits, preventing false detection of a 0x00 data character as a LIN break symbol.

18.2.6 SCI Control Register 3 (SCIxC3)

	7	6	5	4	3	2	1	0
R	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 18-9. SCI Control Register 3 (SCIxC3)

Table 18-7. SCIxC3 Field Descriptions

Field	Description
7 R8	Ninth Data Bit for Receiver. When the SCI is configured for 9-bit data (M = 1), R8 can be thought of as a ninth receive data bit to the left of the msb of the buffered data in the SCIxD register. When reading 9-bit data, read R8 before reading SCIxD because reading SCIxD completes automatic flag clearing sequences that could allow R8 and SCIxD to be overwritten with new data.
6 T8	Ninth Data Bit for Transmitter. When the SCI is configured for 9-bit data (M = 1), T8 may be thought of as a ninth transmit data bit to the left of the msb of the data in the SCIxD register. When writing 9-bit data, the entire 9-bit value is transferred to the SCI shift register after SCIxD is written so T8 should be written (if it needs to change from its previous value) before SCIxD is written. If T8 does not need to change in the new value (such as when it is used to generate mark or space parity), it need not be written each time SCIxD is written.
5 TXDIR	TxD Pin Direction in Single-Wire Mode. When the SCI is configured for single-wire half-duplex operation (LOOPS = RSRC = 1), this bit determines the direction of data at the TxD pin. 0 TxD pin is an input in single-wire mode. 1 TxD pin is an output in single-wire mode.

Table 18-7. SCIxC3 Field Descriptions (continued)

Field	Description
4 TXINV[1]	Transmit Data Inversion. Setting this bit reverses the polarity of the transmitted data output. 0 Transmit data not inverted 1 Transmit data inverted
3 ORIE	Overrun Interrupt Enable. This bit enables the overrun flag (OR) to generate hardware interrupt requests. 0 OR interrupts disabled (use polling). 1 Hardware interrupt requested when OR is set.
2 NEIE	Noise Error Interrupt Enable. This bit enables the noise flag (NF) to generate hardware interrupt requests. 0 NF interrupts disabled (use polling). 1 Hardware interrupt requested when NF is set.
1 FEIE	Framing Error Interrupt Enable. This bit enables the framing error flag (FE) to generate hardware interrupt requests. 0 FE interrupts disabled (use polling). 1 Hardware interrupt requested when FE is set.
0 PEIE	Parity Error Interrupt Enable. This bit enables the parity error flag (PF) to generate hardware interrupt requests. 0 PF interrupts disabled (use polling). 1 Hardware interrupt requested when PF is set.

[1] Setting TXINV inverts the TxD output for all cases: data bits, start and stop bits, break, and idle.

18.2.7 SCI Data Register (SCIxD)

This register is actually two separate registers. Reads return the contents of the read-only receive data buffer and writes go to the write-only transmit data buffer. Reads and writes of this register are also involved in the automatic flag clearing mechanisms for the SCI status flags.

	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 18-10. SCI Data Register (SCIxD)

18.3 Functional Description

The SCI allows full-duplex, asynchronous, NRZ serial communication among the MCU and remote devices, including other MCUs. The SCI comprises a baud rate generator, transmitter, and receiver block. The transmitter and receiver operate independently, although they use the same baud rate generator. During normal operation, the MCU monitors the status of the SCI, writes the data to be transmitted, and processes received data. The following describes each of the blocks of the SCI.

18.3.1 Baud Rate Generation

As shown in Figure 18-11, the clock source for the SCI baud rate generator is the bus-rate clock.

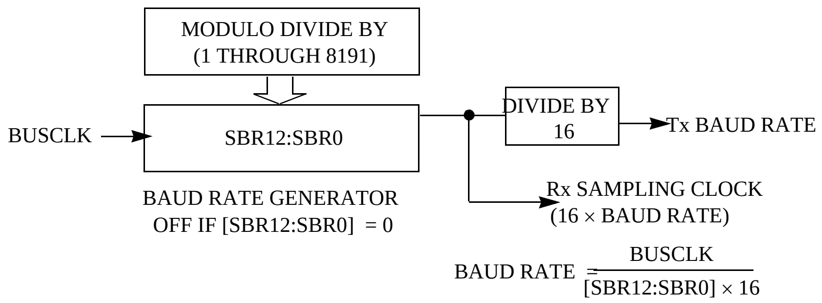


Figure 18-11. SCI Baud Rate Generation

SCI communications require the transmitter and receiver (which typically derive baud rates from independent clock sources) to use the same baud rate. Allowed tolerance on this baud frequency depends on the details of how the receiver synchronizes to the leading edge of the start bit and how bit sampling is performed.

The MCU resynchronizes to bit boundaries on every high-to-low transition. In the worst case, there are no such transitions in the full 10- or 11-bit time character frame so any mismatch in baud rate is accumulated for the whole character time. For a Freescale Semiconductor SCI system whose bus frequency is driven by a crystal, the allowed baud rate mismatch is about ±4.5 percent for 8-bit data format and about ±4 percent for 9-bit data format. Although baud rate modulo divider settings do not always produce baud rates that exactly match standard rates, it is normally possible to get within a few percent, which is acceptable for reliable communications.

18.3.2 Transmitter Functional Description

This section describes the overall block diagram for the SCI transmitter, as well as specialized functions for sending break and idle characters. The transmitter block diagram is shown in Figure 18-1.

The transmitter output (TxD) idle state defaults to logic high (TXINV is cleared following reset). The transmitter output is inverted by setting TXINV. The transmitter is enabled by setting the TE bit in SCIxC2. This queues a preamble character that is one full character frame of the idle state. The transmitter then remains idle until data is available in the transmit data buffer. Programs store data into the transmit data buffer by writing to the SCI data register (SCIxD).

The central element of the SCI transmitter is the transmit shift register that is 10 or 11 bits long depending on the setting in the M control bit. For the remainder of this section, assume M is cleared, selecting the normal 8-bit data mode. In 8-bit data mode, the shift register holds a start bit, eight data bits, and a stop bit. When the transmit shift register is available for a new SCI character, the value waiting in the transmit data register is transferred to the shift register (synchronized with the baud rate clock) and the transmit data register empty (TDRE) status flag is set to indicate another character may be written to the transmit data buffer at SCIxD.

If no new character is waiting in the transmit data buffer after a stop bit is shifted out the TxD pin, the transmitter sets the transmit complete flag and enters an idle mode, with TxD high, waiting for more characters to transmit.

Writing 0 to TE does not immediately release the pin to be a general-purpose I/O pin. Any transmit activity in progress must first be completed. This includes data characters in progress, queued idle characters, and queued break characters.

18.3.2.1 Send Break and Queued Idle

The SBK control bit in SCIxC2 sends break characters originally used to gain the attention of old teletype receivers. Break characters are a full character time of logic 0 (10 bit times including the start and stop bits). A longer break of 13 bit times can be enabled by setting BRK13. Normally, a program would wait for TDRE to become set to indicate the last character of a message has moved to the transmit shifter, then write 1 and then write 0 to the SBK bit. This action queues a break character to be sent as soon as the shifter is available. If SBK remains 1 when the queued break moves into the shifter (synchronized to the baud rate clock), an additional break character is queued. If the receiving device is another Freescale Semiconductor SCI, the break characters are received as 0s in all eight data bits and a framing error (FE = 1) occurs.

When idle-line wakeup is used, a full character time of idle (logic 1) is needed between messages to wake up any sleeping receivers. Normally, a program would wait for TDRE to become set to indicate the last character of a message has moved to the transmit shifter, then write 0 and then write 1 to the TE bit. This action queues an idle character to be sent as soon as the shifter is available. As long as the character in the shifter does not finish while TE is cleared, the SCI transmitter never actually releases control of the TxD pin. If there is a possibility of the shifter finishing while TE is cleard, set the general-purpose I/O controls so the pin shared with TxD is an output driving a logic 1. This ensures that the TxD line looks like a normal idle line even if the SCI loses control of the port pin between writing 0 and then 1 to TE.

The length of the break character is affected by the BRK13 and M bits as shown below.

Table 18-8. Break Character Length

BRK13	M	Break Character Length
0	0	10 bit times
0	1	11 bit times
1	0	13 bit times
1	1	14 bit times

18.3.3 Receiver Functional Description

In this section, the receiver block diagram (Figure 18-2) is a guide for the overall receiver functional description. Next, the data sampling technique used to reconstruct receiver data is described in more detail. Finally, two variations of the receiver wakeup function are explained.

The receiver input is inverted by setting RXINV. The receiver is enabled by setting the RE bit in SCIxC2. Character frames consist of a start bit of logic 0, eight (or nine) data bits (lsb first), and a stop bit of logic 1. For information about 9-bit data mode, refer to Section •, “8- and 9-bit data modes”. For the remainder of this discussion, assume the SCI is configured for normal 8-bit data mode.

After receiving the stop bit into the receive shifter, and provided the receive data register is not already full, the data character is transferred to the receive data register and the receive data register full (RDRF) status flag is set. If RDRF was already set indicating the receive data register (buffer) was already full, the overrun (OR) status flag is set and the new data is lost. Because the SCI receiver is double-buffered, the program has one full character time after RDRF is set before the data in the receive data buffer must be read to avoid a receiver overrun.

When a program detects that the receive data register is full (RDRF = 1), it gets the data from the receive data register by reading SCIxD. The RDRF flag is cleared automatically by a two-step sequence normally satisfied in the course of the user's program that manages receive data. Refer to Section 18.3.4, "Interrupts and Status Flags," for more details about flag clearing.

18.3.3.1 Data Sampling Technique

The SCI receiver uses a 16× baud rate clock for sampling. The receiver starts by taking logic level samples at 16 times the baud rate to search for a falling edge on the RxD serial data input pin. A falling edge is defined as a logic 0 sample after three consecutive logic 1 samples. The 16× baud rate clock divides the bit time into 16 segments labeled RT1 through RT16. When a falling edge is located, three more samples are taken at RT3, RT5, and RT7 to make sure this was a real start bit and not merely noise. If at least two of these three samples are 0, the receiver assumes it is synchronized to a receive character.

The receiver then samples each bit time, including the start and stop bits, at RT8, RT9, and RT10 to determine the logic level for that bit. The logic level is interpreted to be that of the majority of the samples taken during the bit time. In the case of the start bit, the bit is assumed to be 0 if at least two of the samples at RT3, RT5, and RT7 are 0 even if one or all of the samples taken at RT8, RT9, and RT10 are 1s. If any sample in any bit time (including the start and stop bits) in a character frame fails to agree with the logic level for that bit, the noise flag (NF) is set when the received character is transferred to the receive data buffer.

The falling edge detection logic continuously looks for falling edges. If an edge is detected, the sample clock is resynchronized to bit times. This improves the reliability of the receiver in the presence of noise or mismatched baud rates. It does not improve worst case analysis because some characters do not have any extra falling edges anywhere in the character frame.

In the case of a framing error, provided the received character was not a break character, the sampling logic that searches for a falling edge is filled with three logic 1 samples so that a new start bit can be detected almost immediately.

In the case of a framing error, the receiver is inhibited from receiving any new characters until the framing error flag is cleared. The receive shift register continues to function, but a complete character cannot transfer to the receive data buffer if FE remains set.

18.3.3.2 Receiver Wakeup Operation

Receiver wakeup is a hardware mechanism that allows an SCI receiver to ignore the characters in a message intended for a different SCI receiver. In such a system, all receivers evaluate the first character(s) of each message, and as soon as they determine the message is intended for a different receiver, they write logic 1 to the receiver wake up (RWU) control bit in SCIxC2. When RWU bit is set, the status flags associated with the receiver (with the exception of the idle bit, IDLE, when RWUID bit is set) are inhibited from setting, thus eliminating the software overhead for handling the unimportant message characters. At the end of a message, or at the beginning of the next message, all receivers automatically force RWU to 0 so all receivers wake up in time to look at the first character(s) of the next message.

18.3.3.2.1 Idle-Line Wakeup

When wake is cleared, the receiver is configured for idle-line wakeup. In this mode, RWU is cleared automatically when the receiver detects a full character time of the idle-line level. The M control bit selects 8-bit or 9-bit data mode that determines how many bit times of idle are needed to constitute a full character time (10 or 11 bit times because of the start and stop bits).

When RWU is one and RWUID is zero, the idle condition that wakes up the receiver does not set the IDLE flag. The receiver wakes up and waits for the first data character of the next message that sets the RDRF flag and generates an interrupt if enabled. When RWUID is one, any idle condition sets the IDLE flag and generates an interrupt if enabled, regardless of whether RWU is zero or one.

The idle-line type (ILT) control bit selects one of two ways to detect an idle line. When ILT is cleared, the idle bit counter starts after the start bit so the stop bit and any logic 1s at the end of a character count toward the full character time of idle. When ILT is set, the idle bit counter does not start until after a stop bit time, so the idle detection is not affected by the data in the last character of the previous message.

18.3.3.2.2 Address-Mark Wakeup

When wake is set, the receiver is configured for address-mark wakeup. In this mode, RWU is cleared automatically when the receiver detects a logic 1 in the most significant bit of a received character (eighth bit when M is cleared and ninth bit when M is set).

Address-mark wakeup allows messages to contain idle characters, but requires the msb be reserved for use in address frames. The logic 1 msb of an address frame clears the RWU bit before the stop bit is received and sets the RDRF flag. In this case, the character with the msb set is received even though the receiver was sleeping during most of this character time.

18.3.4 Interrupts and Status Flags

The SCI system has three separate interrupt vectors to reduce the amount of software needed to isolate the cause of the interrupt. One interrupt vector is associated with the transmitter for TDRE and TC events. Another interrupt vector is associated with the receiver for RDRF, IDLE, RXEDGIF, and LBKDIF events. A third vector is used for OR, NF, FE, and PF error conditions. Each of these ten interrupt sources can be separately masked by local interrupt enable masks. The flags can be polled by software when the local masks are cleared to disable generation of hardware interrupt requests.

The SCI transmitter has two status flags that can optionally generate hardware interrupt requests. Transmit data register empty (TDRE) indicates when there is room in the transmit data buffer to write another transmit character to SCIxD. If the transmit interrupt enable (TIE) bit is set, a hardware interrupt is requested when TDRE is set. Transmit complete (TC) indicates that the transmitter is finished transmitting all data, preamble, and break characters and is idle with TxD at the inactive level. This flag is often used in systems with modems to determine when it is safe to turn off the modem. If the transmit complete interrupt enable (TCIE) bit is set, a hardware interrupt is requested when TC is set. Instead of hardware interrupts, software polling may be used to monitor the TDRE and TC status flags if the corresponding TIE or TCIE local interrupt masks are cleared.

When a program detects that the receive data register is full (RDRF = 1), it gets the data from the receive data register by reading SCIxD. The RDRF flag is cleared by reading SCIxS1 while RDRF is set and then reading SCIxD.

When polling is used, this sequence is naturally satisfied in the normal course of the user program. If hardware interrupts are used, SCIxS1 must be read in the interrupt service routine (ISR). Normally, this is done in the ISR anyway to check for receive errors, so the sequence is automatically satisfied.

The IDLE status flag includes logic that prevents it from getting set repeatedly when the RxD line remains idle for an extended period of time. IDLE is cleared by reading SCIxS1 while IDLE is set and then reading SCIxD. After IDLE has been cleared, it cannot become set again until the receiver has received at least one new character and has set RDRF.

If the associated error was detected in the received character that caused RDRF to be set, the error flags — noise flag (NF), framing error (FE), and parity error flag (PF) — are set at the same time as RDRF. These flags are not set in overrun cases.

If RDRF was already set when a new character is ready to be transferred from the receive shifter to the receive data buffer, the overrun (OR) flag is set instead of the data along with any associated NF, FE, or PF condition is lost.

At any time, an active edge on the RxD serial data input pin causes the RXEDGIF flag to set. The RXEDGIF flag is cleared by writing a 1 to it. This function does depend on the receiver being enabled (RE = 1).

18.3.5 Additional SCI Functions

The following sections describe additional SCI functions.

18.3.5.1 8- and 9-Bit Data Modes

The SCI system (transmitter and receiver) can be configured to operate in 9-bit data mode by setting the M control bit in SCIxC1. In 9-bit mode, there is a ninth data bit to the left of the msb of the SCI data register. For the transmit data buffer, this bit is stored in T8 in SCIxC3. For the receiver, the ninth bit is held in R8 in SCIxC3.

For coherent writes to the transmit data buffer, write to the T8 bit before writing to SCIxD.

If the bit value to be transmitted as the ninth bit of a new character is the same as for the previous character, it is not necessary to write to T8 again. When data is transferred from the transmit data buffer to the transmit shifter, the value in T8 is copied at the same time data is transferred from SCIxD to the shifter.

The 9-bit data mode is typically used with parity to allow eight bits of data plus the parity in the ninth bit, or it is used with address-mark wakeup so the ninth data bit can serve as the wakeup bit. In custom protocols, the ninth bit can also serve as a software-controlled marker.

18.3.5.2 Stop Mode Operation

During all stop modes, clocks to the SCI module are halted.

In stop1 and stop2 modes, all SCI register data is lost and must be re-initialized upon recovery from these two stop modes. No SCI module registers are affected in stop3 mode.

The receive input active edge detect circuit remains active in stop3 mode, but not in stop2. An active edge on the receive input brings the CPU out of stop3 mode if the interrupt is not masked (RXEDGIE = 1).

Because the clocks are halted, the SCI module resumes operation upon exit from stop (only in stop3 mode). Software should ensure stop mode is not entered while there is a character being transmitted out of or received into the SCI module.

18.3.5.3 Loop Mode

When LOOPS is set, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Loop mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. In this mode, the transmitter output is internally connected to the receiver input and the RxD pin is not used by the SCI, so it reverts to a general-purpose port I/O pin.

18.3.5.4 Single-Wire Operation

When LOOPS is set, the RSRC bit in the same register chooses between loop mode (RSRC = 0) or single-wire mode (RSRC = 1). Single-wire mode implements a half-duplex serial connection. The receiver is internally connected to the transmitter output and to the TxD pin. The RxD pin is not used and reverts to a general-purpose port I/O pin.

In single-wire mode, the TXDIR bit in SCIxC3 controls the direction of serial data on the TxD pin. When TXDIR is cleared, the TxD pin is an input to the SCI receiver and the transmitter is temporarily disconnected from the TxD pin so an external device can send serial data to the receiver. When TXDIR is set, the TxD pin is an output driven by the transmitter. In single-wire mode, the internal loop back connection from the transmitter to the receiver causes the receiver to receive characters that are sent out by the transmitter.

Chapter 19
8-Bit Serial Peripheral Interface (SPIV3)

19.1 Introduction

The SPI1 in MCF51AC256 series MCUs is a 8-bit serial peripheral interface (SPI) module.

19.1.1 Features

Features of the SPI module include:

- Master or slave mode operation
- Full-duplex or single-wire bidirectional option
- Programmable transmit bit rate
- Double-buffered transmit and receive
- Serial clock phase and polarity options
- Slave select output
- Selectable msb-first or lsb-first shifting

19.1.2 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.

19.1.2.1 SPI System Block Diagram

Figure 19-1 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 ($\overline{\text{SS}}$ pin). In this system, the master device has configured its $\overline{\text{SS}}$ pin as an optional slave select output.

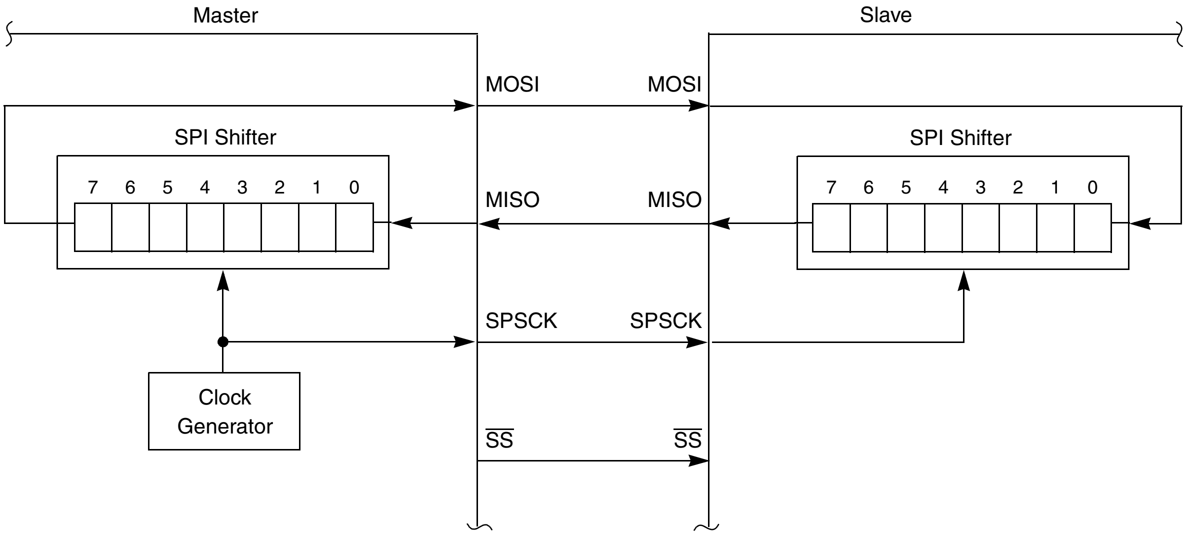


Figure 19-1. SPI System Connections

The most common uses of the SPI system include connecting simple shift registers for adding input or output ports or connecting small peripheral devices such as serial A/D or D/A converters. Although

Figure 19-1 shows a system where data is exchanged between two MCUs, many practical systems involve simpler connections where data is unidirectionally transferred from the master MCU to a slave or from a slave to the master MCU.

19.1.2.2 SPI Module Block Diagram

Figure 19-2 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 SPI1D) and gets transferred to the SPI shift register at the start of a data transfer. After shifting in a byte of data, the data is transferred into the double-buffered receiver where it can be read (read from SPI1D). 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 SPSCK 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 SPSCK 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 SPSCK pins to each other, all MISO pins together, and all MOSI pins together. Peripheral devices often use slightly different names for these pins.

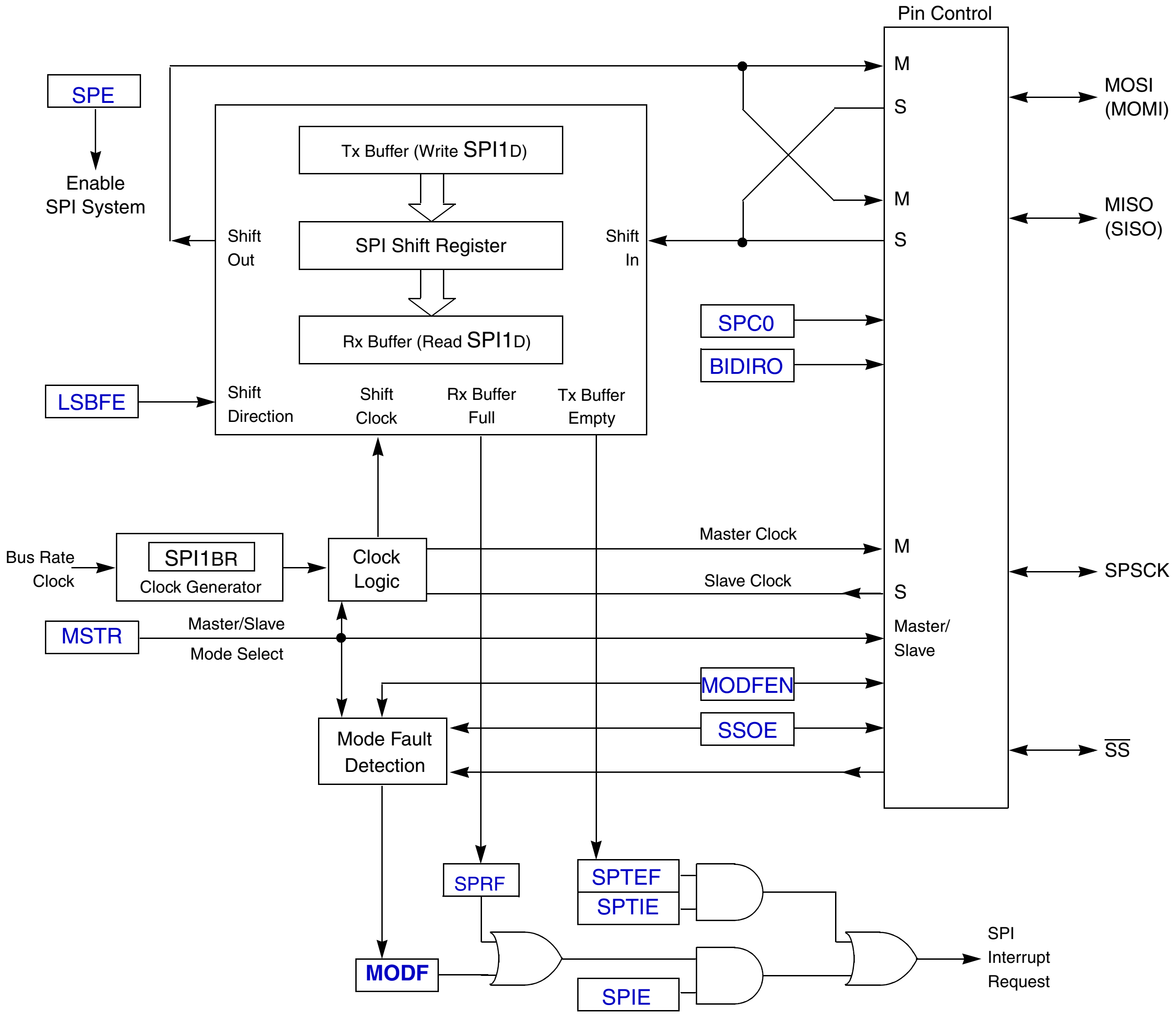


Figure 19-2. SPI Module Block Diagram

19.1.3 SPI Baud Rate Generation

As shown in Figure 19-3, the clock source for the SPI baud rate generator is the bus clock. The three prescale bits (SPI1BR[SPPR]) choose a prescale divisor of 1, 2, 3, 4, 5, 6, 7, or 8. The three rate select bits (SPI1BR[SPR]) divide the output of the prescaler stage by 2, 4, 8, 16, 32, 64, 128, or 256 to obtain the internal SPI master mode bit-rate clock.

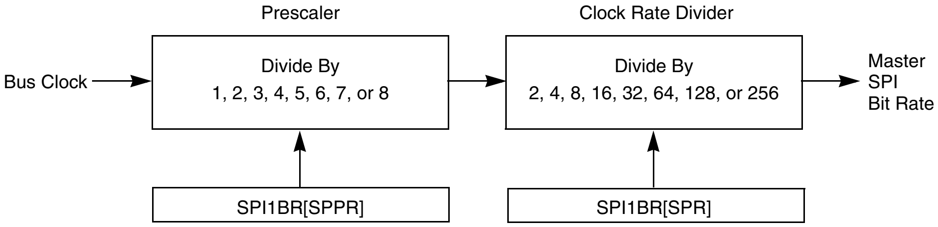


Figure 19-3. SPI Baud Rate Generation

19.2 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 (SPI1C1[SPE] is cleared), these four pins revert to being general-purpose port I/O pins that are not controlled by the SPI.

19.2.1 SPSCK — 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.

19.2.2 MOSI — Master Data Out, Slave Data In

When the SPI is enabled as a master and SPI pin control zero bit (SPI1C2[SPC0]) is cleared (not bidirectional mode), this pin is the serial data output. When the SPI is enabled as a slave and SPI1C2[SPC0] is cleared, this pin is the serial data input. If SPC0 is set 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 (SPI1C2[BIDIROE] = 0) or an output (BIDIROE = 1). If SPC0 is set and slave mode is selected, this pin is not used by the SPI and reverts to a general-purpose port I/O pin.

19.2.3 MISO — Master Data In, Slave Data Out

When the SPI is enabled as a master and SPI pin control zero (SPI1C2[SPC0]) is cleared (not bidirectional mode), this pin is the serial data input. When the SPI is enabled as a slave and SPC0 is cleared, this pin is the serial data output. If SPC0 is set 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 is set and master mode is selected, this pin is not used by the SPI and reverts to being a general-purpose port I/O pin.

19.2.4 $\overline{\text{SS}}$ — 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 (SPI1C2[MODFEN] = 0), this pin is not used by the SPI and reverts to a general-purpose port I/O pin. When the SPI is enabled as a master and MODFEN is set, the slave select output enable bit determines whether this pin acts as the mode fault input (SPI1C1[SSOE] = 0) or as the slave select output (SSOE = 1).

19.3 Modes of Operation

19.3.1 SPI in Stop Modes

The SPI is disabled in all stop modes, regardless of the settings before executing the STOP instruction. During stop2 mode, the SPI module is fully powered down. Upon wake-up from stop2 mode, the SPI module is in the reset state. During stop3 mode, clocks to the SPI module are halted. No registers are affected. If stop3 is exited with a reset, the SPI is placed into its reset state. If stop3 is exited with an interrupt, the SPI continues from the state it was in when stop3 was entered.

19.4 Register Definition

The SPI contains five 8-bit registers to select SPI options, control baud rate, report SPI status, and for transmit/receive data.

Refer to the direct-page register summary in the memory chapter of this document 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.

19.4.1 SPI Control Register 1 (SPI1C1)

This read/write register includes the SPI enable control, interrupt enables, and configuration options.

	7	6	5	4	3	2	1	0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0

Figure 19-4. SPI Control Register 1 (SPI1C1)

Table 19-1. SPI1C1 Field Descriptions

Field	Description
7 SPIE	**SPI Interrupt Enable (for SPRF and MODF).** 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	**SPI System Enable**. Disabling the SPI halts any transfer in progress, clears data buffers, and initializes internal state machines. SPRF is cleared and SPTEF is set to indicate the SPI transmit data buffer is empty. 0 SPI system inactive 1 SPI system enabled

Table 19-1. SPI1C1 Field Descriptions (continued)

Field	Description
5 SPTIE	**SPI Transmit Interrupt Enable**. This is the interrupt enable bit for SPI transmit buffer empty (SPTEF). 0 Interrupts from SPTEF inhibited (use polling) 1 When SPTEF is 1, hardware interrupt requested
4 MSTR	**Master/Slave Mode Select** 0 SPI module configured as a slave SPI device 1 SPI module configured as a master SPI device
3 CPOL	**Clock Polarity**. This bit effectively places an inverter in series with the clock signal from a master SPI or to a slave SPI device. Refer to Section 19.5.1, “SPI Clock Formats” for more details. 0 Active-high SPI clock (idles low) 1 Active-low SPI clock (idles high)
2 CPHA	**Clock Phase**. This bit selects one of two clock formats for different synchronous serial peripheral devices. Refer to Section 19.5.1, “SPI Clock Formats” for more details. 0 First edge on SPSCK occurs at the middle of the first cycle of an 8-cycle data transfer 1 First edge on SPSCK occurs at the start of the first cycle of an 8-cycle data transfer
1 SSOE	**Slave Select Output Enable**. This bit is used with the mode fault enable (SPI1C2[MODFEN]) bit and the master/slave (MSTR) control bit to determine the function of the SS pin as shown in Table 19-2.
0 LSBFE	**lsb First (Shifter Direction)** 0 SPI serial data transfers start with most significant bit 1 SPI serial data transfers start with least significant bit

Table 19-2. $\overline{\text{SS}}$ 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	$\overline{\text{SS}}$ input for mode fault	Slave select input
1	1	Automatic $\overline{\text{SS}}$ output	Slave select input

NOTE

Ensure that the SPI should not be disabled (SPE = 0) at the same time as a bit change to SPI1C1[CPHA]. These changes should be performed as separate operations or unexpected behavior may occur.

19.4.2 SPI Control Register 2 (SPI1C2)

This read/write register controls optional features of the SPI system. Bits 7, 6, 5, and 2 are reserved and always read 0.

	7	6	5	4	3	2	1	0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W								
Reset	0	0	0	0	0	0	0	0

Figure 19-5. SPI Control Register 2 (SPI1C2)

Table 19-3. SPI1C2 Register Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 MODFEN	**Master Mode-Fault Function Enable.** When the SPI is configured for slave mode, this bit has no meaning or effect. (The SS pin is the slave select input.) In master mode, this bit determines how the SS pin is used (refer to Table 19-2 for more details). 0 Mode fault function disabled, master SS pin reverts to general-purpose I/O not controlled by SPI 1 Mode fault function enabled, master SS pin acts as the mode fault input or the slave select output
3 BIDIROE	**Bidirectional Mode Output Enable.** When bidirectional mode is enabled by setting SPC0, 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 the MOSI (MOMI) or MISO (SISO) pin, respectively, as the single SPI data I/O pin. When SPC0 is cleared, 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	Reserved, should be cleared.
1 SPISWAI	**SPI Stop in Wait Mode** 0 SPI clocks continue to operate in wait mode 1 SPI clocks stop when the MCU enters wait mode
0 SPC0	**SPI Pin Control 0** — The SPC0 bit chooses single-wire bidirectional mode. If MSTR is cleared (slave mode), the SPI uses the MISO (SISO) pin for bidirectional SPI data transfers. If MSTR is set (master mode), the SPI uses the MOSI (MOMI) pin for bidirectional SPI data transfers. When SPC0 is set 1, BIDIROE enables or disables the output driver for the single bidirectional SPI I/O pin. 0 SPI uses separate pins for data input and data output 1 SPI configured for single-wire bidirectional operation

19.4.3 SPI Baud Rate Register (SPI1BR)

This register sets the prescaler and bit rate divisor for an SPI master. This register may be read or written at any time.

	7	6	5	4	3	2	1	0
R	0	SPPR			0	SPR		
W								
Reset	0	0	0	0	0	0	0	0

Figure 19-6. SPI Baud Rate Register (SPI1BR)

Table 19-4. SPI1BR Register Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, should be cleared.</td></tr>
<tr><td>6–4
SPPR</td><td><b>SPI Baud Rate Prescale Divisor.</b> This 3-bit field selects one of eight divisors for the SPI baud rate prescaler as shown below. 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 Figure 19-3).
<table>
<tr><th>SPPR</th><th>Prescaler Divisor</th><th>SPPR</th><th>Prescaler Divisor</th></tr>
<tr><td>000</td><td>1</td><td>100</td><td>5</td></tr>
<tr><td>001</td><td>2</td><td>101</td><td>6</td></tr>
<tr><td>010</td><td>3</td><td>110</td><td>7</td></tr>
<tr><td>011</td><td>4</td><td>111</td><td>8</td></tr>
</table></td></tr>
<tr><td>3</td><td>Reserved, should be cleared.</td></tr>
<tr><td>2–0
SPR</td><td><b>SPI Baud Rate Divisor.</b> This 3-bit field selects one of eight divisors for the SPI baud rate divider as shown below. The SPI baud rate prescaler supplies the input to this divider (see Figure 19-3). The output of this divider is the SPI bit rate clock for master mode.
<table>
<tr><th>SPR</th><th>Rate Divisor</th><th>SPR</th><th>Rate Divisor</th></tr>
<tr><td>000</td><td>2</td><td>100</td><td>32</td></tr>
<tr><td>001</td><td>4</td><td>101</td><td>64</td></tr>
<tr><td>010</td><td>8</td><td>110</td><td>128</td></tr>
<tr><td>011</td><td>16</td><td>111</td><td>256</td></tr>
</table></td></tr>
</table>

19.4.4 SPI Status Register (SPI1S)

This register has three read-only status bits. Bits 6, 3, 2, 1, and 0 are not reserved and always read 0. Writes have no meaning or effect.

	7	6	5	4	3	2	1	0
R	SPRF	0	SPTEF	MODF	0	0	0	0
W								
Reset	0	0	1	0	0	0	0	0

Figure 19-7. SPI Status Register (SPI1S)

Table 19-5. SPI1S Register Field Descriptions

Field	Description
7 SPRF	**SPI Read Buffer Full Flag.** SPRF is set at the completion of an SPI transfer to indicate that received data may be read from the SPI data register (SPI1D). SPRF is cleared by reading SPRF while it is set, then reading SPI1D. 0 No data available in the receive data buffer 1 Data available in the receive data buffer
6	Reserved, should be cleared.
5 SPTEF	**SPI Transmit Buffer Empty Flag.** This bit is set when there is room in the transmit data buffer. It is cleared by reading SPI1S with SPTEF set, followed by writing a data value to the transmit buffer at SPI1D. SPI1S must be read with SPTEF set before writing data to SPI1D or the SPI1D write is ignored. SPTEF generates a CPU interrupt request if SPI1C1[SPTIE] is also set. SPTEF is automatically set when a data byte transfers from the transmit buffer into the transmit shift register. For an idle SPI (no data in the transmit buffer or the shift register and no transfer in progress), data written to SPI1D is transferred to the shifter almost immediately so SPTEF is set within two bus cycles allowing a second 8-bit data value to be queued into the transmit buffer. After completion of the transfer of the value in the shift register, the queued value from the transmit buffer is automatically moved to the shifter and SPTEF is set, indicating there is room for new data in the transmit buffer. If no new data is waiting in the transmit buffer, SPTEF remains set and no data moves from the buffer to the shifter. 0 SPI transmit buffer not empty 1 SPI transmit buffer empty
4 MODF	**Master Mode Fault Flag.** MODF is set if the SPI is configured as a master and the slave select input asserts, indicating some other SPI device is also configured as a master. The SS pin acts as a mode fault error input only when MSTR and MODFEN are set and SSOE is cleared. Otherwise, MODF is never set. MODF is cleared by reading MODF while it is 1, then writing to the SPI1C1 register. 0 No mode fault error 1 Mode fault error detected
3–0	Reserved, should be cleared.

19.4.5 SPI Data Register (SPI1D)

Reads of this register returns the data read from the receive data buffer. Writes to this register write data to the transmit data buffer. When the SPI is configured as a master, writing data to the transmit data buffer initiates an SPI transfer.

Data should not be written to the transmit data buffer unless SPI1S[SPTEF] is set, indicating there is room in the transmit buffer to queue a new transmit byte.

Data may be read from SPI1D 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.

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

Figure 19-8. SPI Data Register (SPI1D)

19.5 Functional Description

An SPI transfer is initiated by checking for the SPI transmit buffer empty flag (SPI1S[SPTEF] = 1) and then writing a byte of data to the SPI data register (SPI1D) in the master SPI device. When the SPI shift register is available, this byte of data is moved from the transmit data buffer to the shifter, SPTEF is set to indicate there is room in the buffer to queue another transmit character if desired, and the SPI serial transfer starts.

During the SPI transfer, data is sampled (read) on the MISO pin at one SPSCK edge and shifted, changing the bit value on the MOSI pin, one-half SPSCK cycle later. After eight SPSCK cycles, the data that was in the shift register of the master has been shifted out the MOSI pin to the slave while eight bits of data were shifted in the MISO pin into the master's shift register. At the end of this transfer, the received data byte is moved from the shifter into the receive data buffer and SPRF is set to indicate the data can be read by reading SPI1D. If another byte of data is waiting in the transmit buffer at the end of a transfer, it is moved into the shifter, SPTEF is set, and a new transfer is started.

Normally, SPI data is transferred most significant bit (msb) first. If SPI1C1[LSBFE] is set, SPI data is shifted lsb first.

When the SPI is configured as a slave, its $\overline{\text{SS}}$ pin must be driven low before a transfer starts and $\overline{\text{SS}}$ must stay low throughout the transfer. If a clock format where CPHA is cleared is selected, $\overline{\text{SS}}$ must be driven to a logic 1 between successive transfers. If CPHA is set, $\overline{\text{SS}}$ may remain low between successive transfers. See Section 19.5.1, "SPI Clock Formats," for more details.

Because the transmitter and receiver are double buffered, a second byte, in addition to the byte currently being shifted out, can be queued into the transmit data buffer. A previously received character can be in the receive data buffer while a new character is being shifted in. The SPTEF flag indicates when the transmit buffer has room for a new character. The SPRF flag indicates when a received character is available in the receive data buffer. The received character must be read out of the receive buffer (read SPI1D) before the next transfer is finished or a receive overrun error results.

In the case of a receive overrun, the new data is lost because the receive buffer held the previous character and was not ready to accept the new data. There is no indication for such an overrun condition so the user must ensure that previous data has been read from the receive buffer before a new transfer is initiated.

19.5.1 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 19-9 shows the clock formats when CPHA is set. At the top of the figure, the eight bit times are shown for reference with bit 1 starting at the first SPSCK edge and bit 8 ending one-half SPSCK cycle after the sixteenth SPSCK edge. The msb first and lsb first lines show the order of SPI data bits depending on SPI1C1[LSBFE]. Both variations of SPSCK polarity are shown; however, 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 $\overline{\text{SS}}$ OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE are set). The master $\overline{\text{SS}}$ output asserts one-half SPSCK cycle before the start of the transfer and negates at the end of the eighth bit time of the transfer. The $\overline{\text{SS}}$ IN waveform applies to the slave select input of a slave.

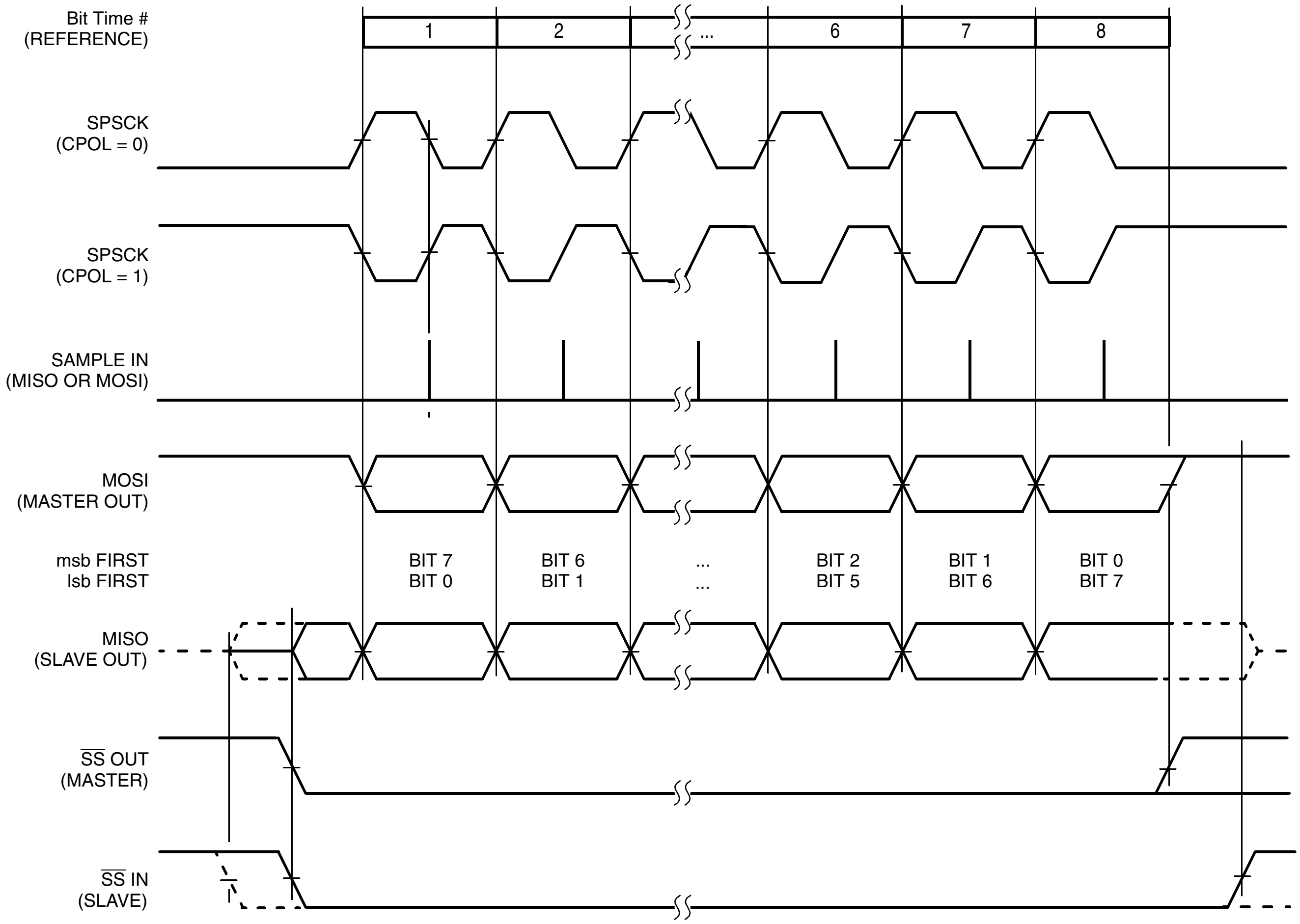


Figure 19-9. SPI Clock Formats (CPHA = 1)

When CPHA is set, the slave begins to drive its MISO output when $\overline{\text{SS}}$ asserts, but the data is not defined until the first SPSCK edge. The first SPSCK 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 SPSCK edge causes the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the third SPSCK edge, the SPI shifter shifts one bit position that shifts in the bit value that was 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 CHPA is set, the slave's $\overline{\text{SS}}$ input is not required to negate between transfers.

Figure 19-10 shows the clock formats when CPHA is cleared. At the top of the figure, the eight bit times are shown for reference with bit 1 starting as the slave is selected ($\overline{\text{SS}}$ IN asserts), and bit 8 ends at the last SPSCK edge. The msb first and lsb first lines show the order of SPI data bits depending on the setting of SPI1C1[LSBFE]. Both variations of SPSCK 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 $\overline{\text{SS}}$ OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE are set). The master $\overline{\text{SS}}$ output asserts at the start of the first bit time of the transfer and negates one-half SPSCK cycle after the end of the eighth bit time of the transfer. The $\overline{\text{SS}}$ IN waveform applies to the slave select input of a slave.

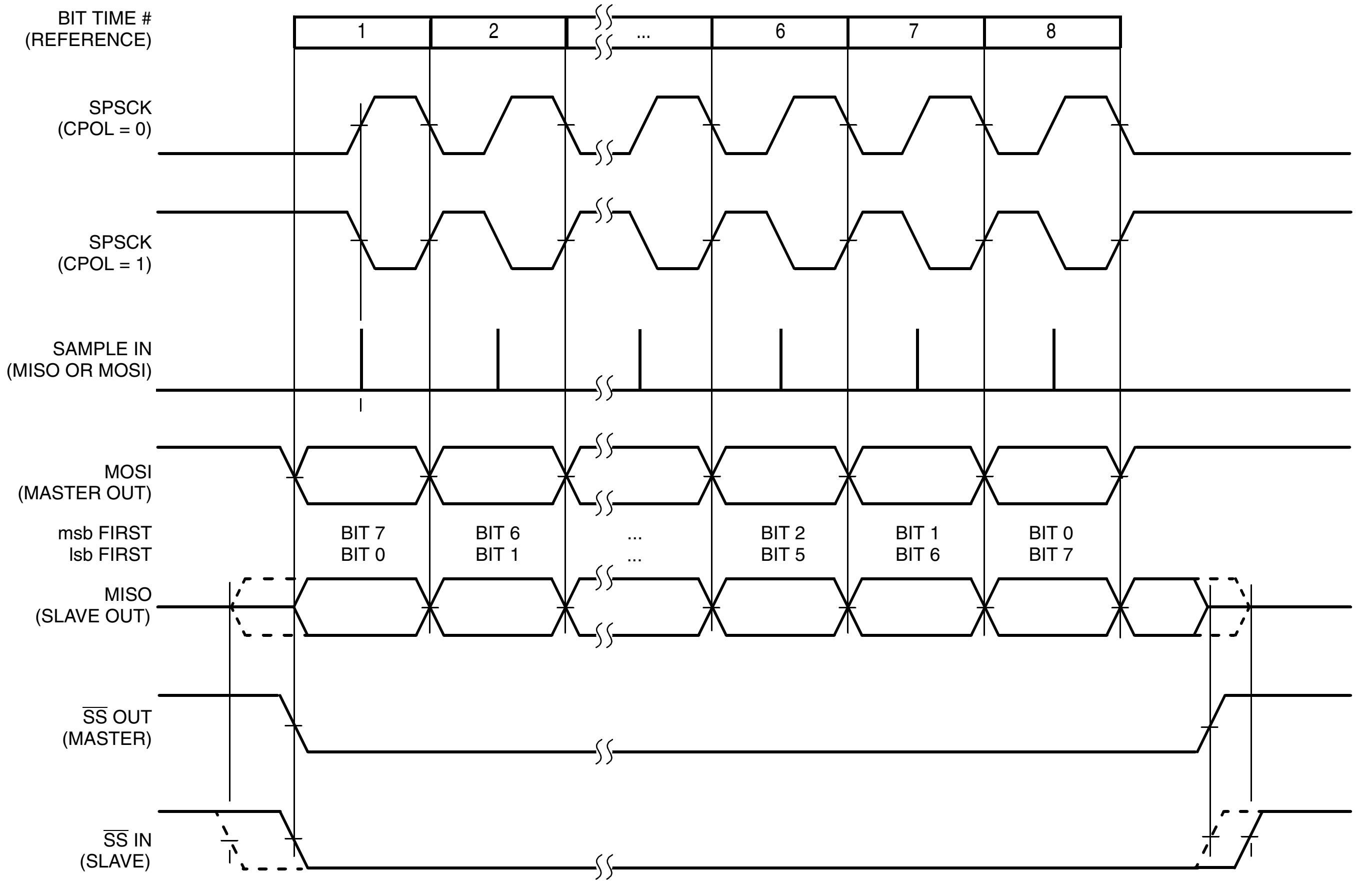


Figure 19-10. SPI Clock Formats (CPHA = 0)

When CPHA is cleared, the slave begins to drive its MISO output with the first data bit value (msb or lsb depending on LSBFE) when $\overline{\text{SS}}$ asserts. The first SPSCK edge causes the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the second SPSCK edge, the SPI shifter shifts one bit position that shifts in the bit value that was 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 is cleared, the slave's $\overline{\text{SS}}$ input must negate between transfers.

19.5.2 SPI Interrupts

There are three flag bits, two interrupt mask bits, and one interrupt vector associated with the SPI system. The SPIE bit enables interrupts from the SPI receiver full flag (SPRF) and mode fault flag (MODF). The SPTIE bit enables interrupts from the SPI transmit buffer empty flag (SPTEF). 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).

19.5.3 Mode Fault Detection

A mode fault occurs and the mode fault flag (MODF) sets when a master SPI device detects an error on the $\overline{\text{SS}}$ pin (provided the $\overline{\text{SS}}$ pin is configured as the mode fault input signal). The $\overline{\text{SS}}$ pin is configured as the mode fault input signal when MSTR and MODFEN is set, and SSOE is clear.

The mode fault detection feature is used in a system where more than one SPI device might become a master at the same time. The error is detected when a master's $\overline{\text{SS}}$ pin is low, indicating that some other SPI device is trying to address this master as if it were a slave. This could indicate a harmful output driver conflict, so the mode fault logic is designed to disable all SPI output drivers when such an error is detected.

When a mode fault is detected, MODF is set and MSTR is cleared to change the SPI configuration back to slave mode. The output drivers on the SPSCK, MOSI, and MISO (if not bidirectional mode) are disabled.

MODF is cleared by reading it while it is set, then writing to the SPI1C1 register. Software should verify the error condition has been corrected before changing the SPI back to master mode.

Chapter 20
16-Bit Serial Peripheral Interface (SPI16V2)

20.1 Introduction

The SPI2 in MCF51AC256 series MCUs is a 16-bit serial peripheral interface (SPI) module with FIFO.

20.1.1 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
- 64bit FIFO mode for high speed/large amounts of data transfers.

20.1.2 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 20.4.10, “Low Power Mode Options.”

20.1.3 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.

20.1.3.1 SPI System Block Diagram

Figure 20-1 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 ($\overline{\text{SS}}$ pin). In this system, the master device has configured its $\overline{\text{SS}}$ pin as an optional slave select output.

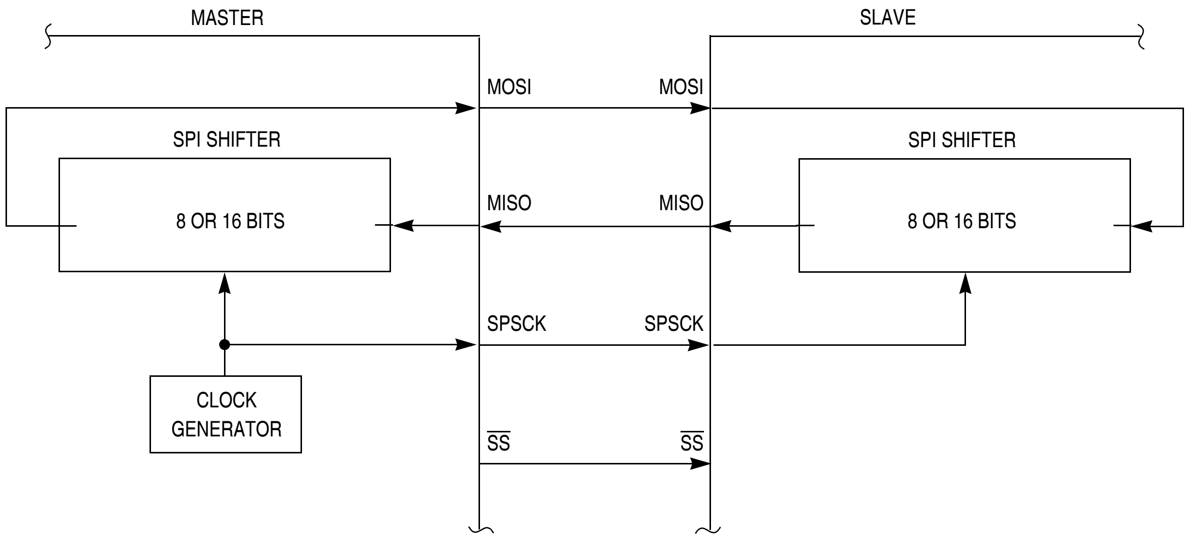


Figure 20-1. SPI System Connections

20.1.3.2 SPI Module Block Diagram

Figure 20-2 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 SPIMODE 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.

Additionally there is an 8-byte receive FIFO and an 8-byte transmit FIFO that once enabled provide features to allow less CPU interrupts to occur when transmitting/receiving high volume/high speed data. When FIFO mode is enabled, the SPI can still function in either 8-bit or 16-bit mode (as per **SPIMODE** bit) and 3 additional flags help monitor the FIFO status and two of these flags can provide CPU interrupts.

When the SPI is configured as a master, the clock output is routed to the SPSCK 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 SPSCK 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 SPSCK pins to each other, all MISO pins together, and all MOSI pins together. Peripheral devices often use slightly different names for these pins.

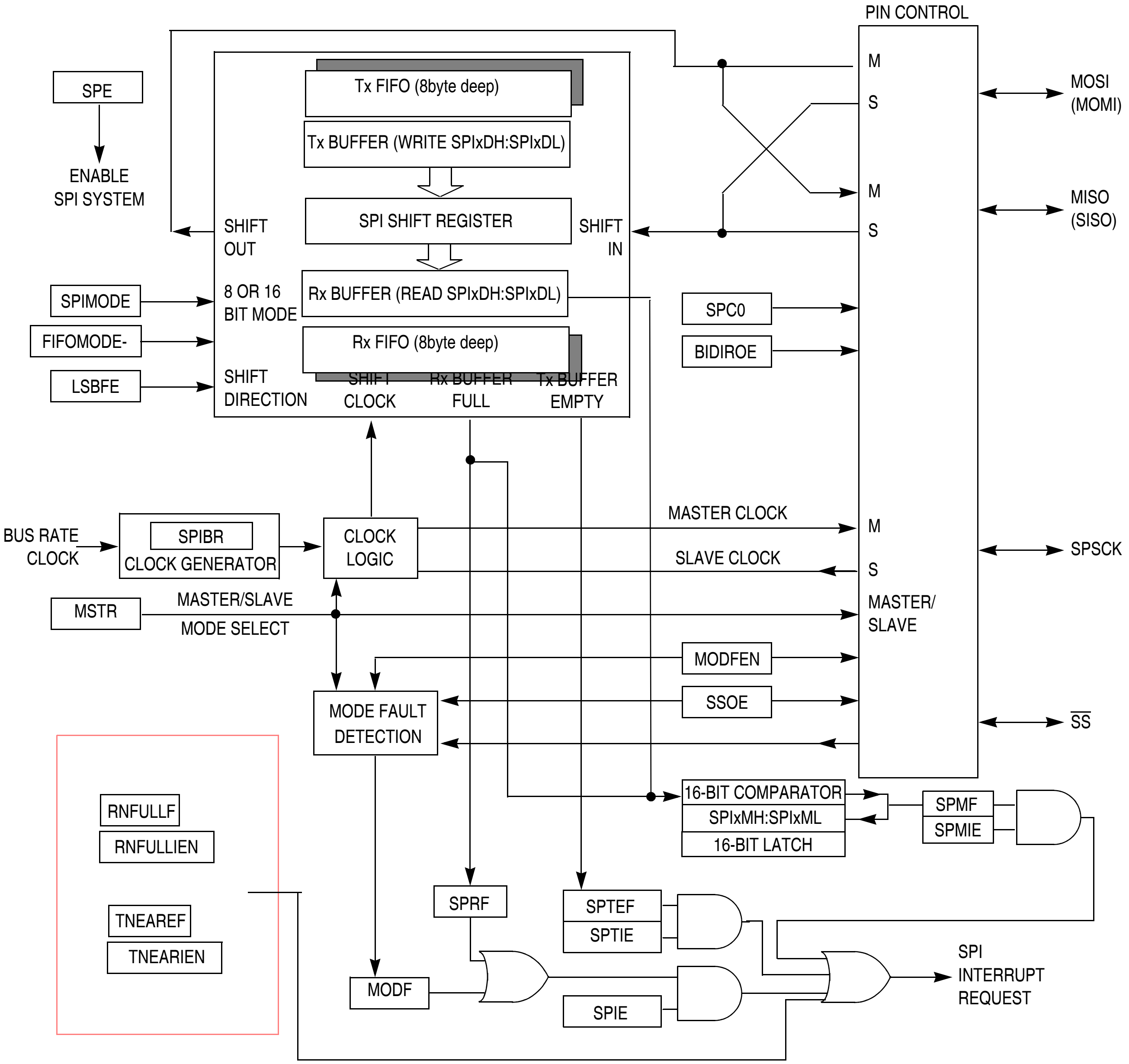


Figure 20-2. SPI Module Block Diagram

20.2 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.

20.2.1 SPSCK — 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.

20.2.2 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.

20.2.3 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.

20.2.4 $\overline{\text{SS}}$ — 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).

20.3 Register Definition

The SPI has eight 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.

20.3.1 SPI Control Register 1 (SPIxC1)

This read/write register includes the SPI enable control, interrupt enables, and configuration options.

	7	6	5	4	3	2	1	0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0

Figure 20-3. SPI Control Register 1 (SPIxC1)

Table 20-1. SPIxC1 Field Descriptions

Field	Description
7 SPIE	**FIFOMODE=0** **SPI Interrupt Enable (for SPRF and MODF)** — 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 **FIFOMODE=1** **SPI Read FIFO Full Interrupt Enable** — This bit when set enables the SPI to interrupt the CPU when the Receive FIFO is full. An interrupt will occur when SPRF flag is set or MODF is set. 0 Read FIFO Full Interrupts are disabled 1 Read FIFO Full Interrupts are enabled
6 SPE	**SPI System Enable** — 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	**SPI Transmit Interrupt Enable** — **FIFOMODE=0** 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) **FIFOMODE=1** This is the interrupt enable bit for SPI transmit FIFO empty (SPTEF). An interrupt occurs when the SPI transmit FIFO is empty (SPTEF is set) 0 Interrupts from SPTEF inhibited (use polling) 1 When SPTEF is 1, hardware interrupt requested
4 MSTR	**Master/Slave Mode Select —** 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	**Clock Polarity** — 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 Section 20.4.6, “SPI Clock Formats” for more details. 0 Active-high SPI clock (idles low) 1 Active-low SPI clock (idles high)
2 CPHA	**Clock Phase** — This bit selects one of two clock formats for different kinds of synchronous serial peripheral devices. Refer to Section 20.4.6, “SPI Clock Formats” for more details. 0 First edge on SPSCK occurs at the middle of the first cycle of a data transfer 1 First edge on SPSCK occurs at the start of the first cycle of a data transfer

Table 20-1. SPIxC1 Field Descriptions (continued)

Field	Description
1 SSOE	**Slave Select Output Enable** — 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 $\overline{\text{SS}}$ pin as shown in Table 20-2.
0 LSBFE	**LSB First (Shifter Direction) —** 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 20-2. $\overline{\text{SS}}$ 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	$\overline{\text{SS}}$ input for mode fault	Slave select input
1	1	Automatic $\overline{\text{SS}}$ output	Slave select input

20.3.2 SPI Control Register 2 (SPIxC2)

This read/write register is used to control optional features of the SPI system. Bits 6 and 5 are not implemented and always read 0.

	7	6	5	4	3	2	1	0
R	SPMIE	SPIMODE	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W			▨			▨		
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 20-4. SPI Control Register 2 (SPIxC2)

Table 20-3. SPIxC2 Register Field Descriptions

Field	Description
7 SPMIE	**SPI Match Interrupt Enable** — 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	**SPI 8- or 16-bit Mode** — 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 Section 20.4.4, “SPI FIFO MODE,” for details. 0 8-bit SPI shift register, match register, and buffers. 1 16-bit SPI shift register, match register, and buffers.

Table 20-3. SPIxC2 Register Field Descriptions (continued)

Field	Description
4 MODFEN	**Master Mode-Fault Function Enable** — 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 Table 20-2 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
3 BIDIROE	**Bidirectional Mode Output Enable** — 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
1 SPISWAI	**SPI Stop in Wait Mode —** 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	**SPI Pin Control 0** — This bit enables bidirectional pin configurations as shown in Table 20-4. 0 SPI uses separate pins for data input and data output. 1 SPI configured for single-wire bidirectional operation.

Table 20-4. Bidirectional Pin Configurations

Pin Mode	SPC0	BIDIROE	MISO	MOSI
Master Mode of Operation				
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
Slave Mode of Operation				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

20.3.3 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.

	7	6	5	4	3	2	1	0
R	0	SPPR2	SPPR1	SPPR0	SPR3	SPR2	SPR1	SPR0
W	▨							
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 20-5. SPI Baud Rate Register (SPIxBR)

Table 20-5. SPIxBR Register Field Descriptions

Field	Description
6:4 SPPR[2:0]	**SPI Baud Rate Prescale Divisor** — This 3-bit field selects one of eight divisors for the SPI baud rate prescaler as shown in Table 20-6. 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 Figure 20-18). See Section 20.4.7, “SPI Baud Rate Generation,” for details.
2:0 SPR[3:0]	**SPI Baud Rate Divisor** — This 4-bit field selects one of nine divisors for the SPI baud rate divider as shown in Table 20-7. The input to this divider comes from the SPI baud rate prescaler (see Figure 20-18). See Section 20.4.7, “SPI Baud Rate Generation,” for details.

Table 20-6. 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 20-7. 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

20.3.4 SPI Status Register (SPIxS)

This register has eight read-only status bits. Writes have no meaning or effect. This register has 4additional flags RNFULLF, TNEARF, TXFULLF and RFIFOEF which provide mechanisms to support an 8-byte FIFO mode. When in 8byte FIFO mode the function of SPRF and SPTEF differs slightly from the normal buffered modes, mainly in how these flags are cleared by the amount available in the transmit and receive FIFOs

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	1	0	0	0[1]	0	0[1]

▨ = Unimplemented or Reserved

Figure 20-6. SPI Status Register (SPIxS)

[1] Note: PoR values of TNEAREF and RFIFOEF is 0. If status register is reset due to change of SPIMODE, FIFOMODE or SPE than, if FIFOMODE = 1, TNEAREF and RFIFOEF resets to 1 else if FIFOMODE = 0, TNEAREF and RFIFOEF resets to 0

Table 20-8. SPIxS Register Field Descriptions

Field	Description
7 SPRF	**SPI Read Buffer Full Flag** — 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). SPRF is cleared by reading SPRF while it is set, then reading the SPI data register. 0 No data available in the receive data buffer. 1 Data available in the receive data buffer. **FIFOMODE=1** **SPI Read FIFO FULL Flag** — This bit indicates the status of the Read FIFO when FIFOMODE enabled. The SPRF is set when the read FIFO has received 64bits (4 words or 8bytes) of data from the shifter and there has been no CPU reads of SPIxDH:SPIxDL. SPRF is cleard by reading the SPI Data Register, which empties the FIFO, assuming another SPI message is not received. 0 Read FIFO is not Full 1 Read FIFO is Full.
6 SPMF	**SPI Match Flag** — SPMF is set after SPRF = 1 when the value in the receive data buffer matches the value in SPIMH:SPIML. To clear the flag, read SPMF when it is set, then write a 1 to it. 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.

Table 20-8. SPIxS Register Field Descriptions

Field	Description
5 SPTEF	**SPI Transmit Buffer Empty Flag** — This bit is set when the transmit data buffer is empty. It 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. 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. 0 SPI transmit buffer not empty 1 SPI transmit buffer empty **FIFOMODE=1** **SPI Transmit FIFO Empty Flag** — *This bit when in FIFOMODE now changed to provide status of the FIFO rather than an 8or16-bit buffer.* This bit is set when the Transmit FIFO is empty. It is cleared by writing a data value to the transmit FIFO at SPIxDH:SPIxDL. SPTEF is automatically set when all data from transmit FIFO 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, a second write of data to the SPIxDH:SPIxDL will clear ths SPTEF flag. After completion of the transfer of the data in the shift register, the queued data from the transmit FIFO will automatically move to the shifter and SPTEF will be set only when all data written to the transmit FIFO has been transfered to the shifter. If no new data is waiting in the transmit FIFO, SPTEF simply remains set and no data moves from the buffer to the shifter. 0 SPI FIFO not empty 1 SPI FIFO empty
4 MODF	**Master Mode Fault Flag** — 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 $\overline{SS}$ 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). 0 No mode fault error 1 Mode fault error detected
3 RNFULLF	**Receive FIFO Nearly Full Flag** — This flag is set when more than three16bit word or six 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 0 or when more than two 16bit word or four 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 1. It has no function if FIFOMODE=0. 0 Receive FIFO has received less than 48bits/32bits (See SPIxC3[4]). 1 Receive FIFO has receieved 48bits/32bits(See SPIxC3[4]) or more.
2 TNEAREF	**Transmit FIFO Nearly Empty Flag** — This flag is set when only one 16bit word or 2 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] = 0 or when only two 16bit word or 4 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] =1. If FIFOMODE is not enabled this bit should be ignored. 0 Transmit FIFO has more than 16bits/32bits (See SPIxC3[5]) left to transmit. 1 Transmit FIFO has 16bits/32 bits(See SPIxC3[5])or less left to transmit
1 TXFULLF	**Transmit FIFO Full Flag** - This bit indicates status of transmit fifo when fifomode is enabled. This flag is set when there are 8 bytes in transmit fifo. If FIFOMODE is not enabled this bit should be ignored. 0 Transmit FIFO has less than 8 bytes. 1 Transmit FIFO has 8 bytes of data.
0 RFIFOEF	**SPI Read FIFO Empty Flag** — This bit indicates the status of the Read FIFO when FIFOMODE enabled. If FIFOMODE is not enabled this bit should be ignored. 0 Read FIFO has data. Reads of the SPIxDH:SPIxDL registers in 16-bit mode or SPIxDL register in 8-bit mode will empty the Read FIFO. 1 Read FIFO is empty.

For FIFO management there are two other important flags that are used to help make the operation more efficient when transfering large amounts of data. These are the Receive FIFO Nearly Full Flag (RNFULLF) and the Transmit FIFO Nearly Empty Flag (TNEAREF). Both these flags provide a “watermark” feature of the FIFOs to allow continuous transmissions of data when running at high speed.

The RNFULLF flag can generate an interrupt if the RNFULLIEN bit in the SPIxC3 Register is set which allows the CPU to start emptying the Receive FIFO without delaying the reception of subsequent bytes. The user can also determine if all data in Receive FIFO has been read by monitoring the RFIFOEF flag.

The TNEAREF flag can generate an interrupt if the TNEARIEN bit n the SPIxC3 Register is set which allows the CPU to start filling the Transmit FIFO before it is empty and thus provide a mechanism to have no breaks in SPI transmission.

NOTE

> SPIxS and both TX and RX fifos gets reset due to change in SPIMODE, FIFOMODE or SPE. PoR values of SPIxS are show in Figure 20-7.

Figure 20-7 and Figure 20-8 shows the reset values due to change of modes after PoR.

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
W								
Reset	0	0	1	0	0	0	0	0

Figure 20-7. Reset values of SPIxS after PoR with FIFOMODE = 0

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
W								
Reset	0	0	1	0	0	1	0	1

Figure 20-8. Reset values of SPIxS after PoR with FIFOMODE = 1

20.3.5 SPI Data Registers (SPIxDH:SPIxDL)

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

Figure 20-9. SPI Data Register High (SPIxDH)

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

Figure 20-10. 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. SPIxS must be read when SPTEF is set before writing to the SPI data registers, or the write will be ignored.

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 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.

20.3.6 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.

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

Figure 20-11. SPI Match Register High (SPIxMH)

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

Figure 20-12. SPI Match Register Low (SPIxML)

20.3.7 SPI Control Register 3 (SPIxC3) — Enable FIFO Feature

The SPI Control Register 3 introduces a 64bit FIFO function on both transmit and receive buffers to be utilised on the SPI. Utilising this FIFO feature allows the SPI to provide high speed transfers of large amounts of data without consuming large amounts of the CPU bandwidth.

Enabling this FIFO function will effect the behaviour of some of the Read/Write Buffer flags in the SPIxS register namely:

The SPRF of the SPIxS register will be set when the Receive FIFO is filled and will interrupt the CPU if the SPIE in the SPIxC1 register is set.

The SPTEF of the SPIxS register will be set when the Transmit FIFO is empty, and will interrupt the CPU if the SPITIE bit is set in the SPIxC1 register. See SPIxC1 and SPIxS registers.

FIFO mode is enabled by setting the FIFOMODE bit, and provides the SPI with an 8-byte receive FIFO and an 8-byte transmit FIFO to reduce the amount of CPU interrupts for high speed/high volume data transfers.

Two interrupt enable bits TNEARIEN and RNFULLIEN provide CPU interrupts based on the "watermark" feature of the TNEARF and RNFULLF flags of the SPIxS register.

Note: This register has sixread/write control bits. Bits 7 thro' 6are not implemented and always read 0. Writes have no meaning or effect. Write to this register happens only when FIFOMODE bit is 1.

	7	6	5	4	3	2	1	0
R	▨	▨	TNEAREF MARK	RNFULL MARK	INTCLR	TNEARIEN	RNFULLIEN	FIFOMODE
W	▨	▨						
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 20-13. SPI Status Register (SPIxC3)

Table 20-9. SPIxC3 Register Field Descriptions

Field	Description
5 TNEAREF MARK	**Transmit FIFO Nearly Empty Water Mark** - This bit selects the mark after which TNEAREF flag is asserted. 0 TNEAREF is set when Transmit FIFO has16bits or less. 1 TNEAREF is set when Transmit FIFO has 32bits or less.
4 RNFULLF MARK	**Receive FIFO Nearly Full Water Mark** - This bit selects the mark for which RNFULLF flag is asserted 0 RNFULLF is set when Receive FIFO has 48bits or more 1 RNFULLF is set when Receive FIFO has 32bits or more.

Table 20-9. SPIxC3 Register Field Descriptions

Field	Description
3 INTCLR	**Interrupt Clearing Mechanism Select** - This bit selects the mechanism by which SPRF, SPTEF, TNEAREF, RNFULLF interrupts gets cleared. 0 Interrupts gets cleared when respective flags gets cleared depending on the state of FIFOs 1 Interrupts gets cleared by writing to the SPIxCI respective bits.
2 TNEARIEN	**Transmit FIFO Nearly Empty Interrupt Enable** — Writing to this bit enables the SPI to interrupt the CPU when the TNEAREF flag is set. This is an additional interrupt on the SPI and will only interrupt the CPU if SPTIE in the SPIxC1 register is also set. This bit is ignored and has no function if FIFOMODE=0. 0 No interrupt on Transmit FIFO Nearly Empty Flag being set. 1 Enable interrupts on Transmit FIFO Nearly Empty Flag being set.
1 RNFULLIEN	**Receive FIFO Nearly Full Interrupt Enable** — Writing to this bit enables the SPI to interrupt the CPU when the RNEARFF flag is set. This is an additional interrupt on the SPI and will only interrupt the CPU if SPIE in the SPIxC1 register is also set. This bit is ignored and has no function if FIFOMODE = 0. 0 No interrupt on RNEARFF being set. 1 Enable interrupts on RNEARFF being set.
0 FIFOMODE	**SPI FIFO Mode Enable** — This bit enables the SPI to utilise a 64-bit FIFO (8bytes 4 16-bit words) for both transmit and receive buffers. 0 Buffer mode disabled. 1 Data available in the receive data buffer.

20.3.8 SPI Clear Interrupt Register (SPIxCI)

The SPI Clear Interrupt register has 4 bits dedicated for clearing the interrupts. Writing 1 to these bits clears the respective interrupts if INTCLR bit in SPIxCR3 is set.

It also have 2 bits to indicate the transmit fifo and receive fifo overrun conditions. When receive fifo is full and a data is received RXFOF flag is set. Similarily when transmit fifo is full and write happens to SPIDR TXFOF is set. These flags gets cleared when a read happens to this register with the flags set.

There are two more bits to indicate the error flags. These flags gets set when due to some spurious reasons entries in fifo becomes greater than 8. At this point all the flags in status register gets reset and entries in FIFO are flushed with respective error flags set. These flags are cleared when a read happen at SPIxCI with the error flags set.

Note: Bits [7:4] are readonly bits. These bits gets cleared when a read happens to this register with the flags set. Bits [3:0] are clear interrupts bits which clears the interrupts by writing 1 to respective bits. Reading these bits always return 0.

	7	6	5	4	3	2	1	0
R	TXFERR	RXFERR	TXFOF	RXFOF	TNEAREFCI	RNFULLFCI	SPTEFCI	SPRFCI
W	▨	▨	▨	▨				
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Table 20-10. SPIxCI Register Field Descriptions

Field	Description
7 TXFERR	**Transmit FIFO ErrorFlag**- This flag indicates that TX FIFO error occured because entries in fifo goes above 8.. 0 No TX Fifo Error Occured 1 TX Fifo error occured.
6 RXFERR	**Receive FIFO Error Flag**- This flag indicates that RX FIFO error occured because entries in fifo goes above 8. 0 No RX Fifo Error Occured 1 RX Fifo error occured.
5 TXFOF	**TX FIFO Overflow Flag**- This Flag indicates that TX FIFO overflow condition has occured.. 0 TX FIFO overflow condition has not occured. 1 TX FIFO overflow condition occured.
4 RXFOF	**RX FIFO Overflow Flag** - This Flag indicates that RX FIFO overflow condition has occured.. 0 RX FIFO overflow condition has not occured. 1 RX FIFO overflow condition occured.
3 TNEAREFCI	**Transmit FIFO Nearly Empty Flag Clear Interrupt Register** - Write of 1 clears the TNEAREF interrupt provided SPIxC3[3] is set.
2 RNFULLFCI	**Receive FIFO Nearly Full Flag Clear Interrupt Register** - Write of 1 clears the RNFULLF interrupt provided SPIxC3[3] is set.
1 SPTEFCI	**Transmit FIFO Empty Flag Clear Interrupt Register** - Write of 1 clears the SPTEF interrupt provided SPIxC3[3] is set.
0 SPRFCI	**Receive FIFO Full Flag Clear Interrupt Register** - Write of 1 clears the TNEAREF interrupt provided SPIxC3[3] is set.

20.4 Functional Description

20.4.1 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 ($\overline{\text{SS}}$)
- 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.

20.4.2 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.

- SPSCK

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 SPSCK pin is the SPI clock output. Through the SPSCK 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.

- $\overline{\text{SS}}$ pin

If MODFEN and SSOE bit are set, the $\overline{\text{SS}}$ pin is configured as slave select output. The $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ pin is configured as input for detecting mode fault error. If the $\overline{\text{SS}}$ input becomes low this indicates a mode fault error where another master tries to drive the MOSI and SPSCK 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 SPSCK, 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 SPSCK-cycle delay. After the delay, SPSCK 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 20.4.6, “SPI Clock Formats”).

NOTE

A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0, BIDIROE with SPC0 set, SPIMODE, FIFOMODE, 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.

20.4.3 Slave Mode

The SPI operates in slave mode when the MSTR bit in SPI Control Register1 is clear.

- SPSCK

In slave mode, SPSCK 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.

- $\overline{\text{SS}}$ pin

The $\overline{\text{SS}}$ pin is the slave select input. Before a data transmission occurs, the $\overline{\text{SS}}$ pin of the slave SPI must be low. $\overline{\text{SS}}$ must remain low until the transmission is complete. If $\overline{\text{SS}}$ goes high, the SPI is forced into idle state.

The $\overline{\text{SS}}$ input also controls the serial data output pin, if $\overline{\text{SS}}$ is high (not selected), the serial data output pin is high impedance, and, if $\overline{\text{SS}}$ 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 ($\overline{\text{SS}}$ is high), then the SPSCK 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 SPSCK 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 SPSCK 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 $\overline{\text{SS}}$ 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 FIFOMODE and SPIMODE in slave mode will corrupt a transmission in progress and has to be avoided.

20.4.4 SPI FIFO MODE

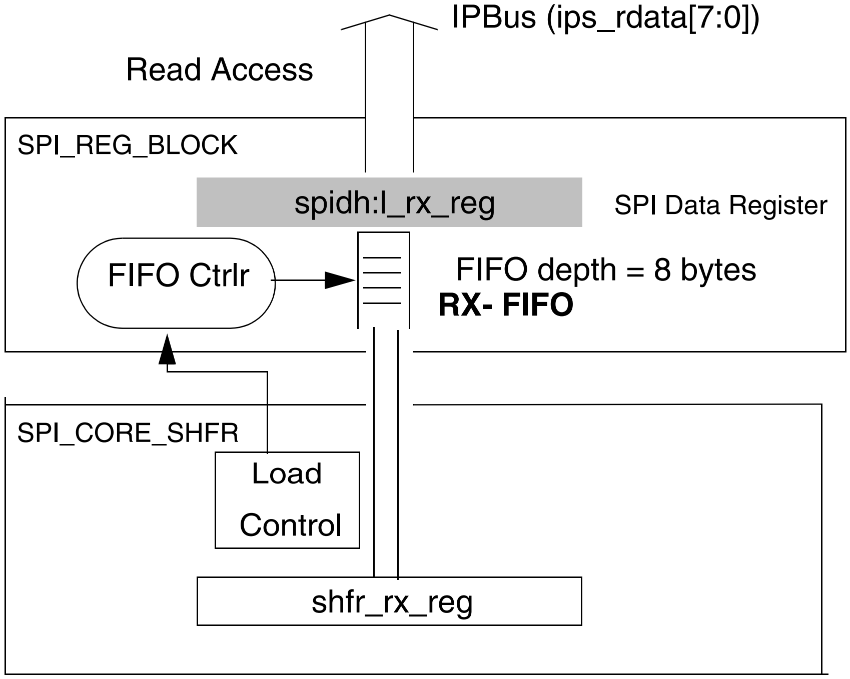


Figure 20-14. SPIDH:L read side structural overview in FIFO mode

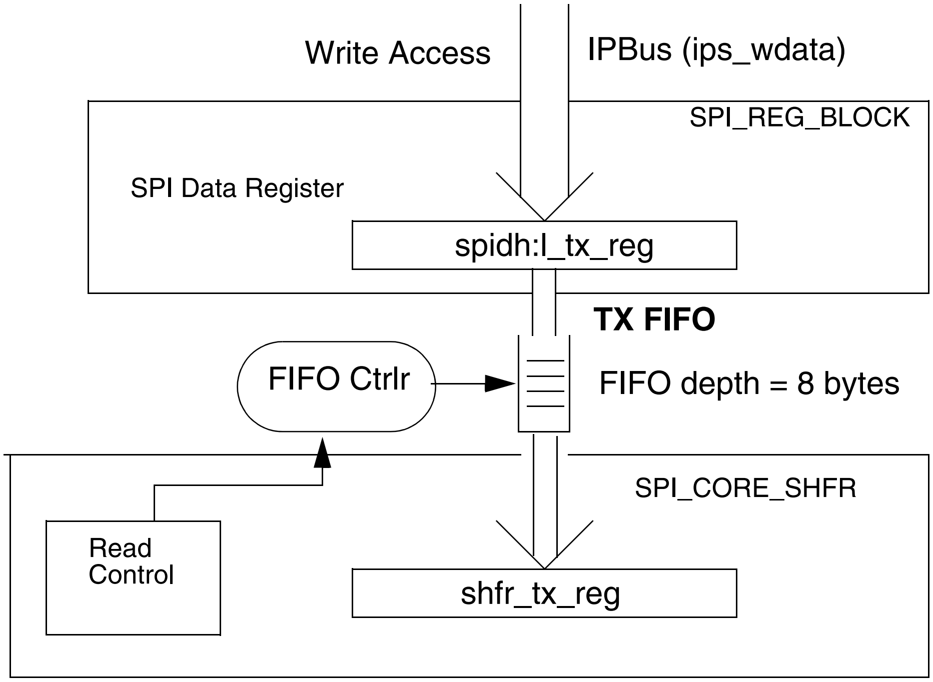


Figure 20-15. SPIDH:L write side structural overview in FIFO mode

SPI works in FIFO mode when SPIxC3[0] bit is set. When in FIFO mode SPI RX buffer and SPI TX buffer is replaced by a 8 byte deep fifo as shown in figure above.

20.4.5 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 SPIxC2 register.

In 8-bit mode (SPIMODE = 0), the SPI Data Register is comprised of one byte: SPIxDL. The SPI Match Register is also comprised of only one byte: SPIxML. Reads of SPIxDH and SPIxMH will return zero. Writes to SPIxDH and SPIxMH will be ignored.

In 16-bit mode (SPIMODE = 1), the SPI Data Register is comprised of two bytes: SPIxDH and SPIxDL. 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.

In 16-bit mode, the SPI Match Register is also comprised of two bytes: SPIxMH and SPIxML. 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 16-bit value into the transmit data buffer.

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.

20.4.6 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 20-16 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 SPSCK edge and bit 8 ending one-half SPSCK cycle after the sixteenth SPSCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCK 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 $\overline{\text{SS}}$ OUT waveform applies to the slave select output from a master (provided

MODFEN and SSOE = 1). The master $\overline{\text{SS}}$ output goes to active low one-half SPSCK cycle before the start of the transfer and goes back high at the end of the eighth bit time of the transfer. The $\overline{\text{SS}}$ IN waveform applies to the slave select input of a slave.

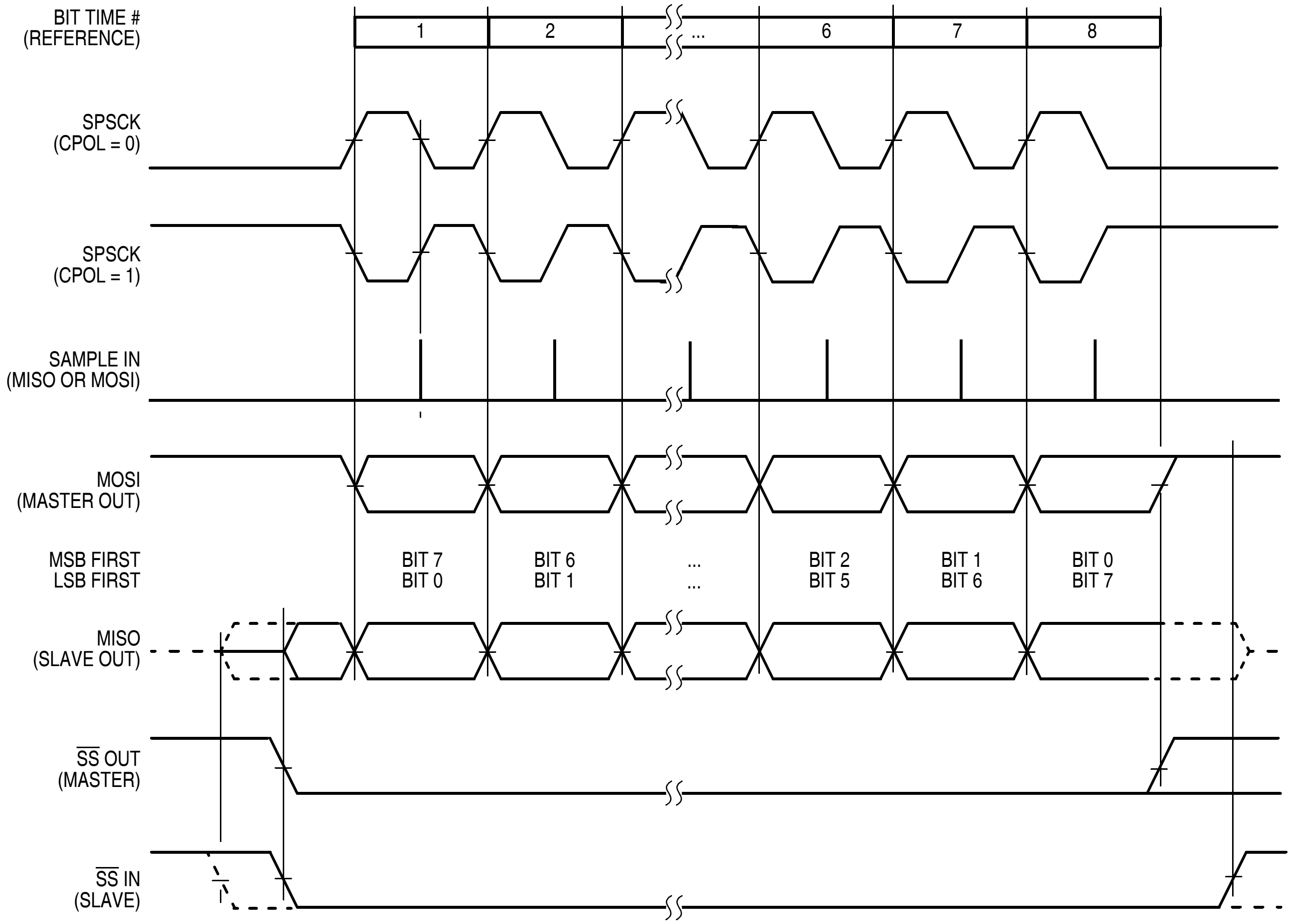


Figure 20-16. SPI Clock Formats (CPHA = 1)

When CPHA = 1, the slave begins to drive its MISO output when $\overline{\text{SS}}$ goes to active low, but the data is not defined until the first SPSCK edge. The first SPSCK 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 SPSCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the third SPSCK 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 $\overline{\text{SS}}$ input is not required to go to its inactive high level between transfers.

Figure 20-17 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 ($\overline{\text{SS}}$ IN goes low), and bit 8 ends at the last SPSCK edge. The MSB first and LSB first lines show the order of SPI data bits depending on the setting in LSBFE. Both variations of SPSCK 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 $\overline{\text{SS}}$ OUT waveform applies to the slave select output from a master (provided MODFEN and SSOE = 1). The master $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ IN waveform applies to the slave select input of a slave.

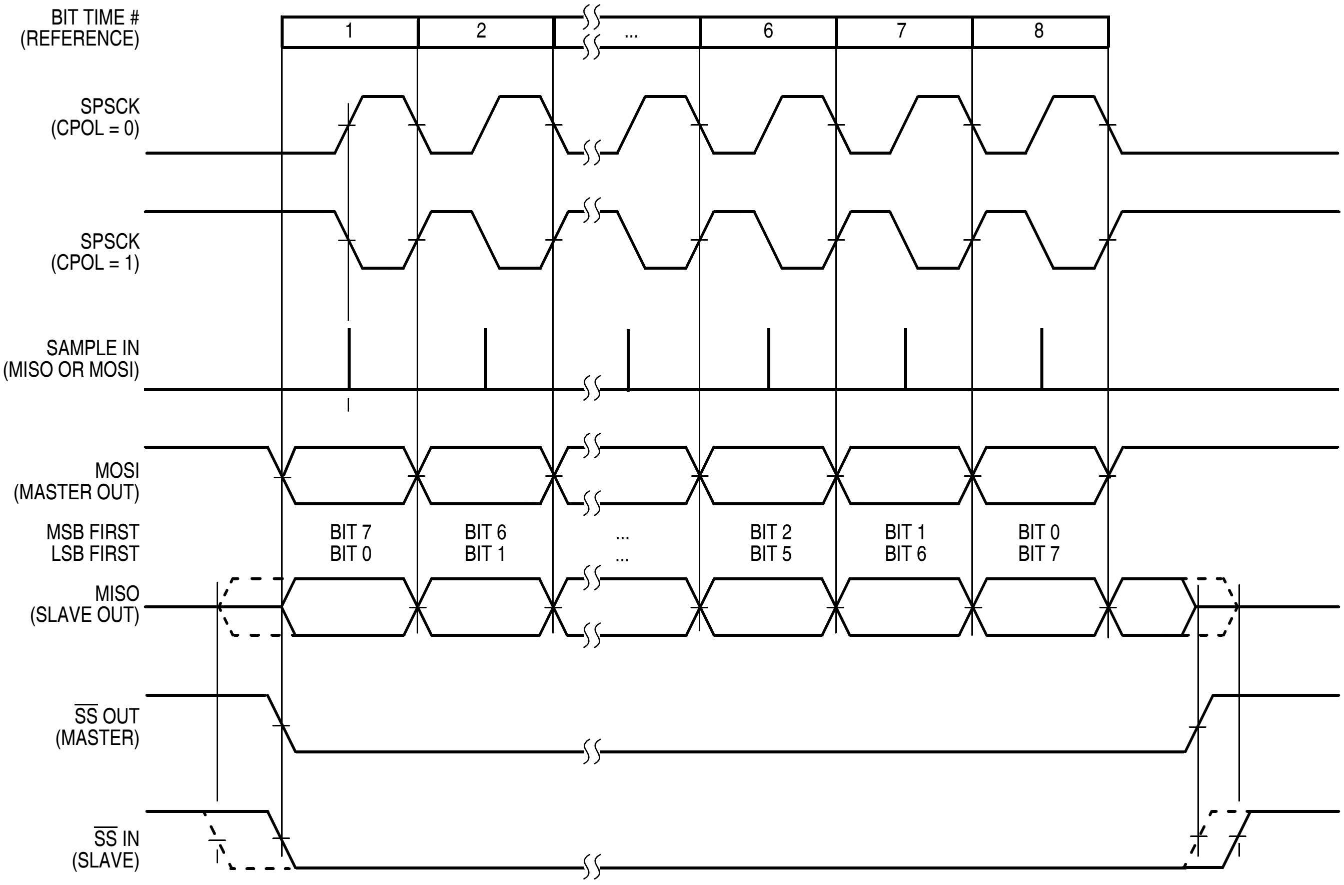


Figure 20-17. SPI Clock Formats (CPHA = 0)

When CPHA = 0, the slave begins to drive its MISO output with the first data bit value (MSB or LSB depending on LSBFE) when $\overline{\text{SS}}$ goes to active low. The first SPSCK edge causes both the master and the slave to sample the data bit values on their MISO and MOSI inputs, respectively. At the second SPSCK 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 $\overline{\text{SS}}$ input must go to its inactive high level between transfers.

20.4.7 SPI Baud Rate Generation

As shown in Figure 20-18, 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 20-7 :

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) \bullet 2^{(\text{SPR} + 1)}$$

The baud rate can be calculated with the following equation:

$$\text{Baud Rate} = \text{BusClock} / \text{BaudRateDivisor}$$

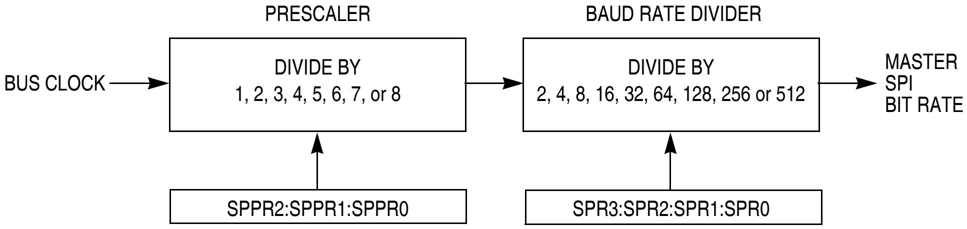


Figure 20-18. SPI Baud Rate Generation

20.4.8 Special Features

20.4.8.1 $\overline{\text{SS}}$ Output

The $\overline{\text{SS}}$ output feature automatically drives the $\overline{\text{SS}}$ pin low during transmission to select external devices and drives it high during idle to deselect external devices. When $\overline{\text{SS}}$ output is selected, the $\overline{\text{SS}}$ output pin is connected to the $\overline{\text{SS}}$ input pin of the external device.

The $\overline{\text{SS}}$ output is available only in master mode during normal SPI operation by asserting the SSOE and MODFEN bits as shown in Table 20-2.

The mode fault feature is disabled while $\overline{\text{SS}}$ output is enabled.

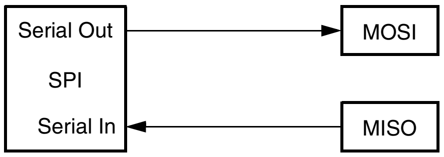
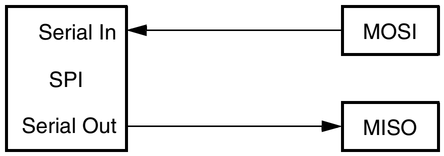
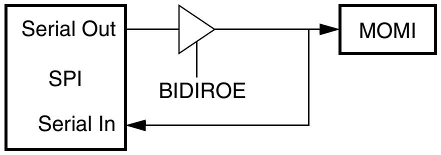
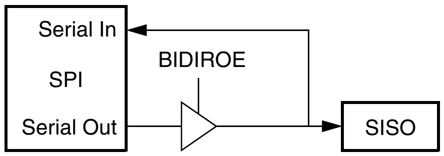
NOTE

Care must be taken when using the $\overline{\text{SS}}$ output feature in a multi-master system since the mode fault feature is not available for detecting system errors between masters.

20.4.8.2 Bidirectional Mode (MOMI or SISO)

The bidirectional mode is selected when the SPC0 bit is set in SPI Control Register 2 (see Table 20-11). 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 20-11. Normal Mode and Bidirectional Mode

When SPE = 1	Master Mode MSTR = 1	Slave Mode MSTR = 0
Normal Mode **SPC0 = 0**		
Bidirectional Mode **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 $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ 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.

20.4.9 Error Conditions

The SPI has one error condition:

- Mode fault error

20.4.9.1 Mode Fault Error

If the $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ 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 $\overline{\text{SS}}$ 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 SPSCK, 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.

20.4.10 Low Power Mode Options

20.4.10.1 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.

20.4.10.2 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 SPSCK continues to be driven from the master. This keeps the slave synchronized to the master and the SPSCK.

    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 tranmission. 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.

20.4.10.3 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.

20.4.10.4 Reset

The reset values of registers and signals are described in Section 20.3, “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.

20.4.10.5 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.

20.4.11 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).

20.4.11.1 MODF

MODF occurs when the master detects an error on the $\overline{\text{SS}}$ pin. The master SPI must be configured for the MODF feature (see Table 20-2). Once MODF is set, the current transfer is aborted and the following bit is changed:

- MSTR=0, The master bit in SPIxC1 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 20.3.4, "SPI Status Register (SPIxS)."

20.4.11.2 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 20.3.4, "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.

20.4.11.3 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 20.3.4, "SPI Status Register (SPIxS)."

20.4.11.4 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.

20.4.11.5 TNEAREF

TNEAREF flag is set when only one 16-bit word or 2 8-bit bytes of data remain in the transmit FIFO provided SPIxC3[5] = 0 or when only two 16bit word or 4 8-bit bytes of data remain in the transmit FIFO provided SPIxC3[5] =1. If FIFOMODE is not enabled this bit should be ignored.

Clearing of this interrupts depends on state of SPIxC3[3] and the status of TNEAREF as described Section 20.3.4, “SPI Status Register (SPIxS).”

20.4.11.6 RNFULLF

RNFULLF is set when more than three16bit word or six 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 0 or when more than two 16bit word or four 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 1.

Clearing of this interrupts depends on state of SPIxC3[3] and the status of RNFULLF as described Section 20.3.4, “SPI Status Register (SPIxS).”

20.5 Initialization/Application Information

20.5.1 SPI Module Initialization Example

20.5.1.1 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.

20.5.1.2 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 SPSCK 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 SPSCK at start of first data transfer cycle
Bit 1	SSOE	= 0	Determines $\overline{\text{SS}}$ 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.

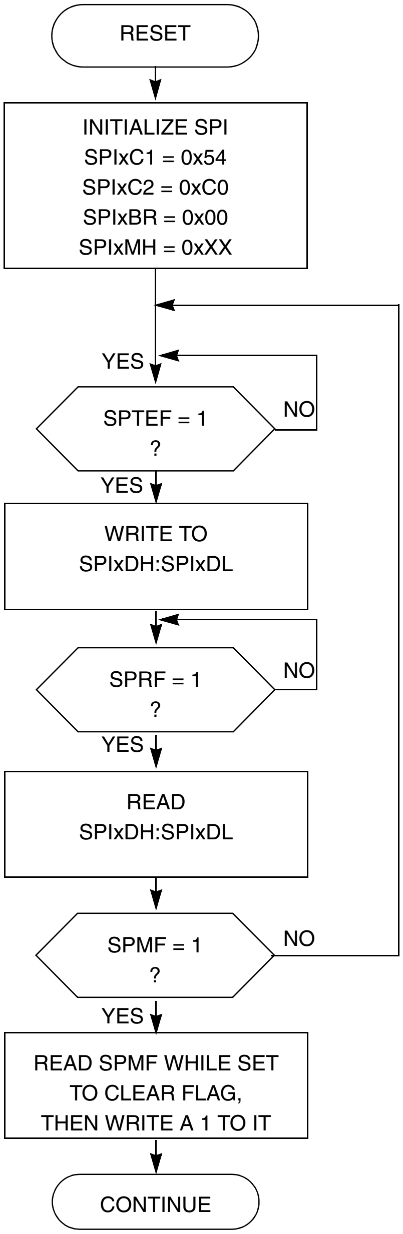


Figure 20-19. Initialization Flowchart Example for SPI Master Device in 16-Bit Mode for FIFOMODE = 0

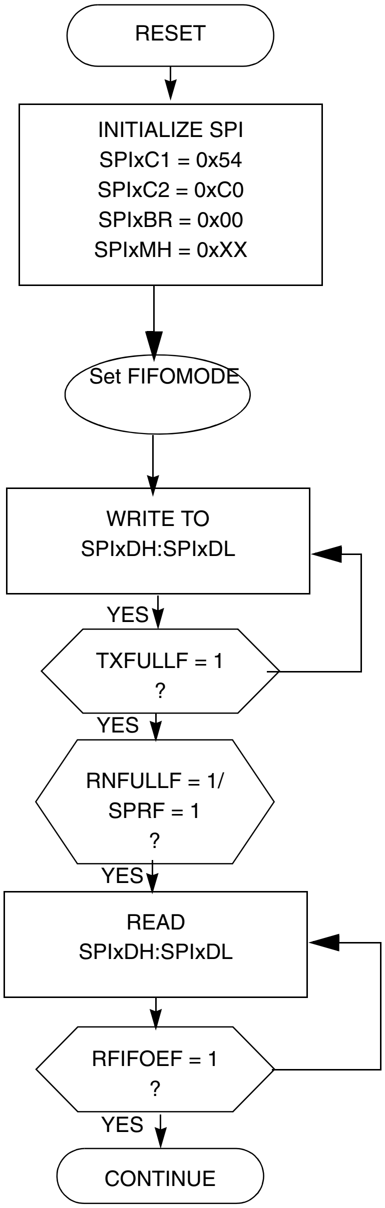


Figure 20-20. Initialization Flowchart Example for SPI Master Device in 16-Bit Mode for FIFOMODE

Chapter 21
Timer/PWM Module (TPMV3)

21.1 Introduction

The TPM is a one-to-eight-channel timer system which supports traditional input capture, output compare, or edge-aligned PWM on each channel. A control bit allows the TPM to be configured such that all channels may be 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.

21.1.1 TPM3 Module Interconnects

21.1.1.1 TPM3 to FTMx synchronization triggers

The TPM3CHn output of the TPM3 module can be used as one of the three FTM hardware synchronization triggers. TPM3CH0 is connected to FTM1 and TPM3CH1 is connected to FTM2.

Typical usage is to setup the TPM3 channel as an output compare. The sync trigger occurs on the rising edge of the TPM3 channel, so the “set output on compare” option (TPM3CnSC[ELSnB:ELSnA]=0b11) must be used.

21.1.2 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

21.1.3 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.

21.1.4 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 21-1 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.

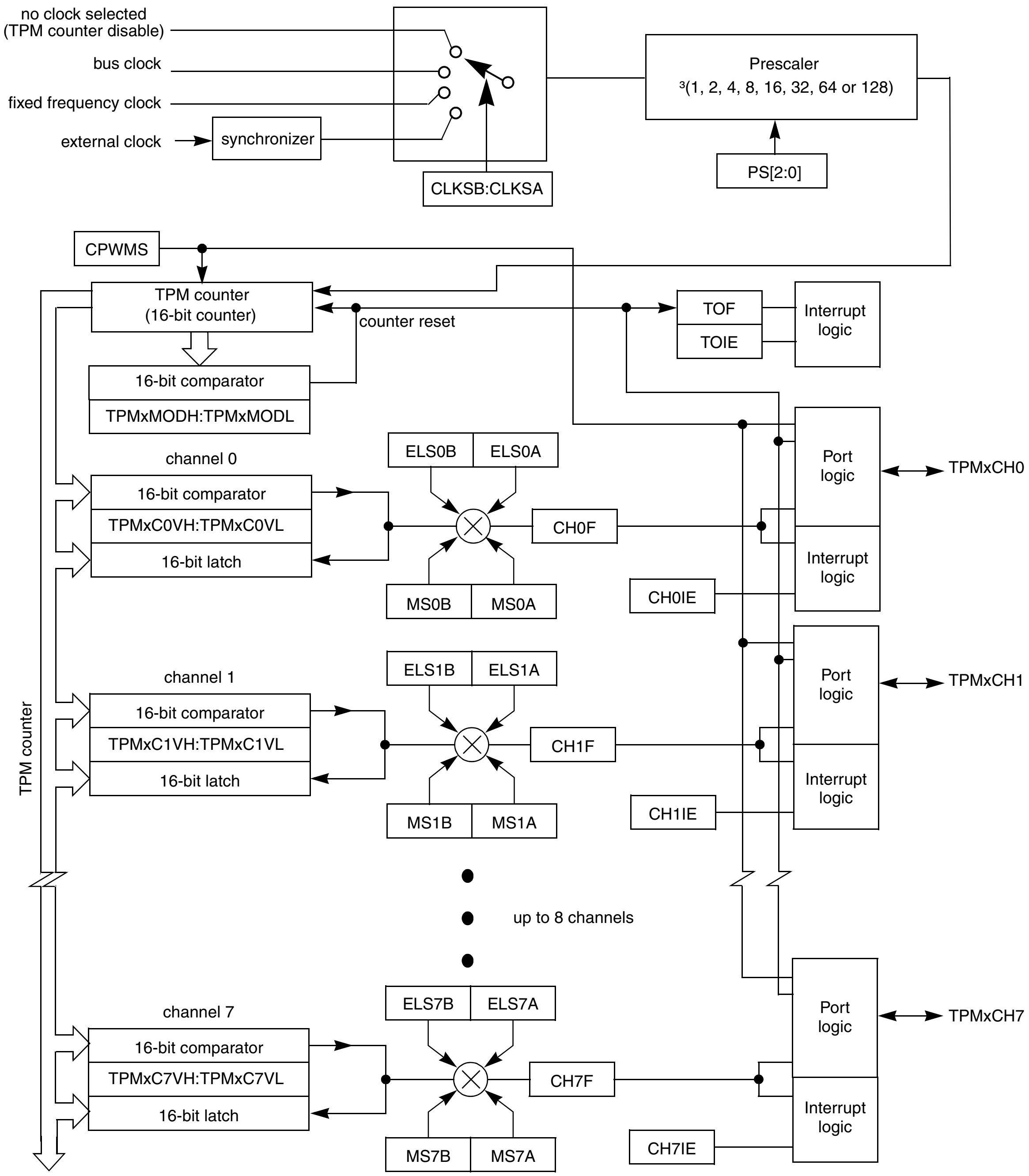


Figure 21-1. TPM Block Diagram

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.

21.2 Signal Description

Table 21-1 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 21-1. Signal Properties

Name	Function
EXTCLK[1]	External clock source that is selected to drive the TPM counter.
TPMxCHn[2]	I/O pin associated with TPM channel n.

[1] 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.

[2] n = channel number (1–8)

21.2.1 Detailed Signal Descriptions

21.2.1.1 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.

21.2.1.2 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.

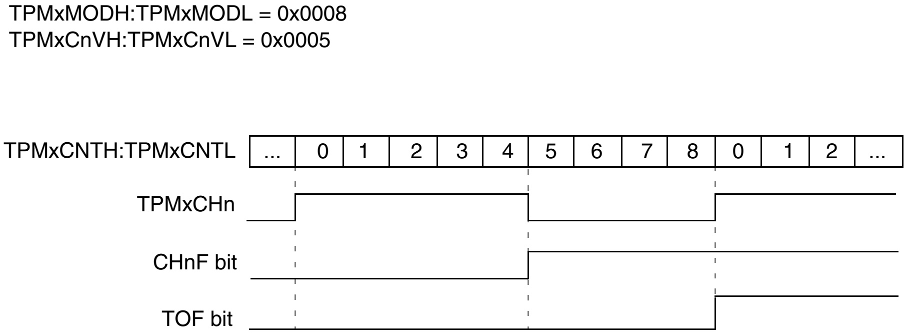


Figure 21-2. High-true pulse of an edge-aligned PWM

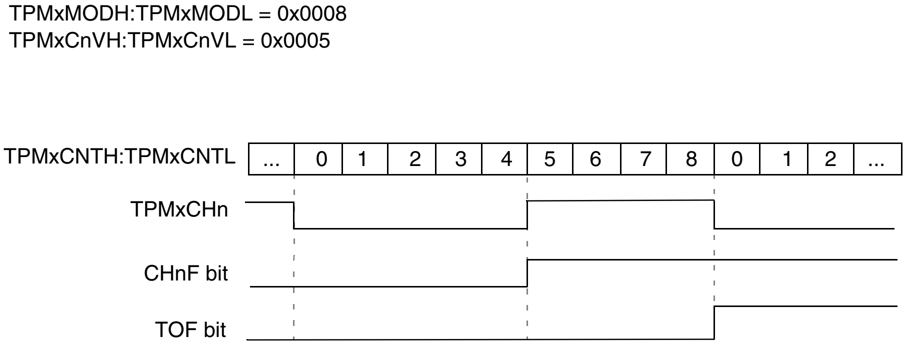


Figure 21-3. Low-true pulse of an edge-aligned PWM

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.

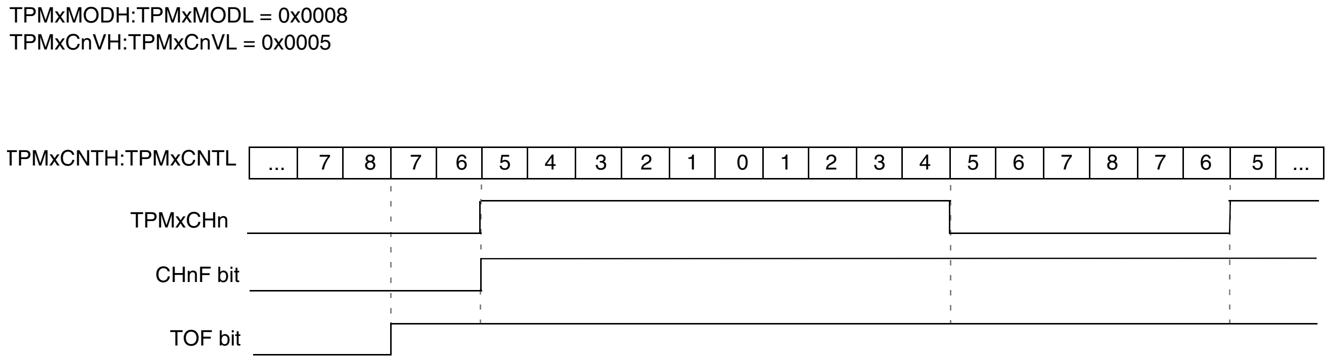


Figure 21-4. High-true pulse of a center-aligned PWM

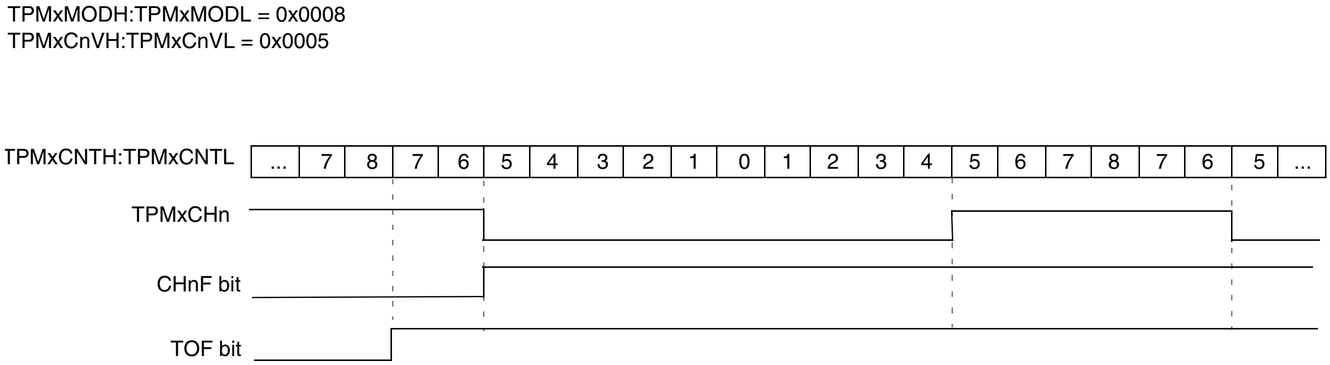


Figure 21-5. Low-true pulse of a center-aligned PWM

21.3 Register Definition

21.3.1 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.

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W	0							
Reset	0	0	0	0	0	0	0	0

Figure 21-6. TPM Status and Control Register (TPMxSC)

Table 21-2. 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 Table 21-3, 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 Table 21-4. 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 21-3. TPM Clock Selection

CLKSB:CLKSA	TPM Clock to Prescaler Input
00	No clock selected (TPM counter disable)

Table 21-3. TPM Clock Selection

CLKSB:CLKSA	TPM Clock to Prescaler Input
01	Bus clock
10	Fixed frequency clock
11	External clock

Table 21-4. 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

21.3.2 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.

	7	6	5	4	3	2	1	0
R	TPMxCNT[15:8]							
W	Any write to TPMxCNTH clears the 16-bit counter							
Reset	0	0	0	0	0	0	0	0

Figure 21-7. TPM Counter Register High (TPMxCNTH)

	7	6	5	4	3	2	1	0
R	TPMxCNT[7:0]							
W	Any write to TPMxCNTL clears the 16-bit counter							
Reset	0	0	0	0	0	0	0	0

Figure 21-8. 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.

21.3.3 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 0xFFFE 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.

	7	6	5	4	3	2	1	0
R	TPMxMOD[15:8]							
W								
Reset	0	0	0	0	0	0	0	0

Figure 21-9. TPM Counter Modulo Register High (TPMxMODH)

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

Reset the TPM counter before writing to the TPM modulo registers to avoid confusion about when the first counter overflow occurs.

21.3.4 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.

	7	6	5	4	3	2	1	0
R	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
W	0						▨	▨
Reset	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-11. TPM Channel n Status and Control Register (TPMxCnSC)

Table 21-5. TPMxCnSC Field Descriptions

Field	Description
7 CHnF	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. 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. Reset clears this bit. Writing a logic 1 to CHnF has no effect. 0 No input capture or output compare event occurred on channel n. 1 Input capture or output compare event on channel n.
6 CHnIE	Channel n interrupt enable. This read/write bit enables interrupts from channel n. Reset clears this bit. 0 Channel n interrupt requests disabled (use for software polling). 1 Channel n interrupt requests enabled.
5 MSnB	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 Table 21-6.

Table 21-5. 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 Table 21-6 for a summary of channel mode and setup controls. **Note:** 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 Table 21-6, 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 21-6. 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)

21.3.5 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.

	7	6	5	4	3	2	1	0
R W	TPMxCnV[15:8]							
Reset	0	0	0	0	0	0	0	0

Figure 21-12. TPM Channel Value Register High (TPMxCnVH)

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

Figure 21-13. 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 0xFFFE 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).

21.4 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.

21.4.1 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.

21.4.1.1 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 21-3). 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.

21.4.1.2 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).

21.4.1.3 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).

21.4.1.4 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.

21.4.2 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.

21.4.2.1 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.

21.4.2.2 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.

21.4.2.3 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 21-14). 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.

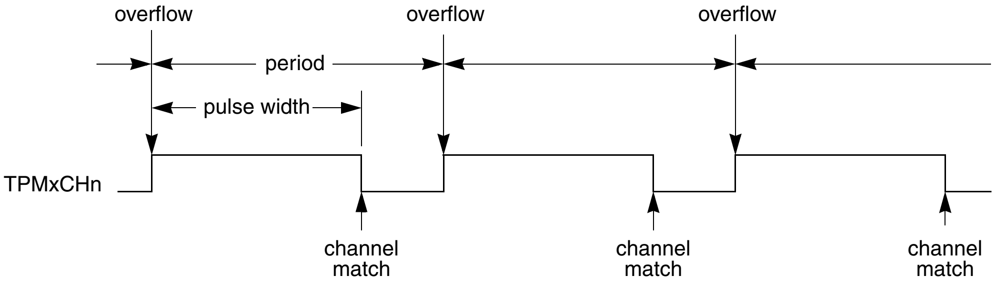


Figure 21-14. EPWM period and pulse width (ELSnA=0)

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.

21.4.2.4 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.

pulse width = 2 × (TPMxCnVH:TPMxCnVL)

period = 2 × (TPMxMODH:TPMxMODL); TPMxMODH:TPMxMODL = 0x0001–0x7FFF

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 21-15). 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.

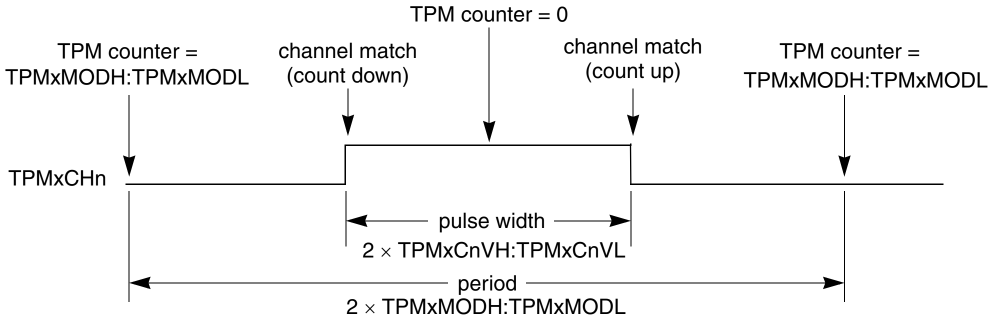


Figure 21-15. CPWM period and pulse width (ELSnA=0)

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 (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.

When TPMxCNTH:TPMxCNTL equals TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

21.5 Reset Overview

21.5.1 General

The TPM is reset whenever any MCU reset occurs.

21.5.2 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.

21.6 Interrupts

21.6.1 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 21-7.

Table 21-7. 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.

21.6.2 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.

21.6.2.1 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.

21.6.2.1.1 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.

21.6.2.1.2 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).

21.6.2.2 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).

21.6.2.2.1 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 21.6.2, "Description of Interrupt Operation."

21.6.2.2.2 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 21.6.2, "Description of Interrupt Operation."

21.6.2.2.3 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 21.6.2, "Description of Interrupt Operation."

Chapter 22
Version 1 ColdFire Debug (CF1_DEBUG)

22.1 Introduction

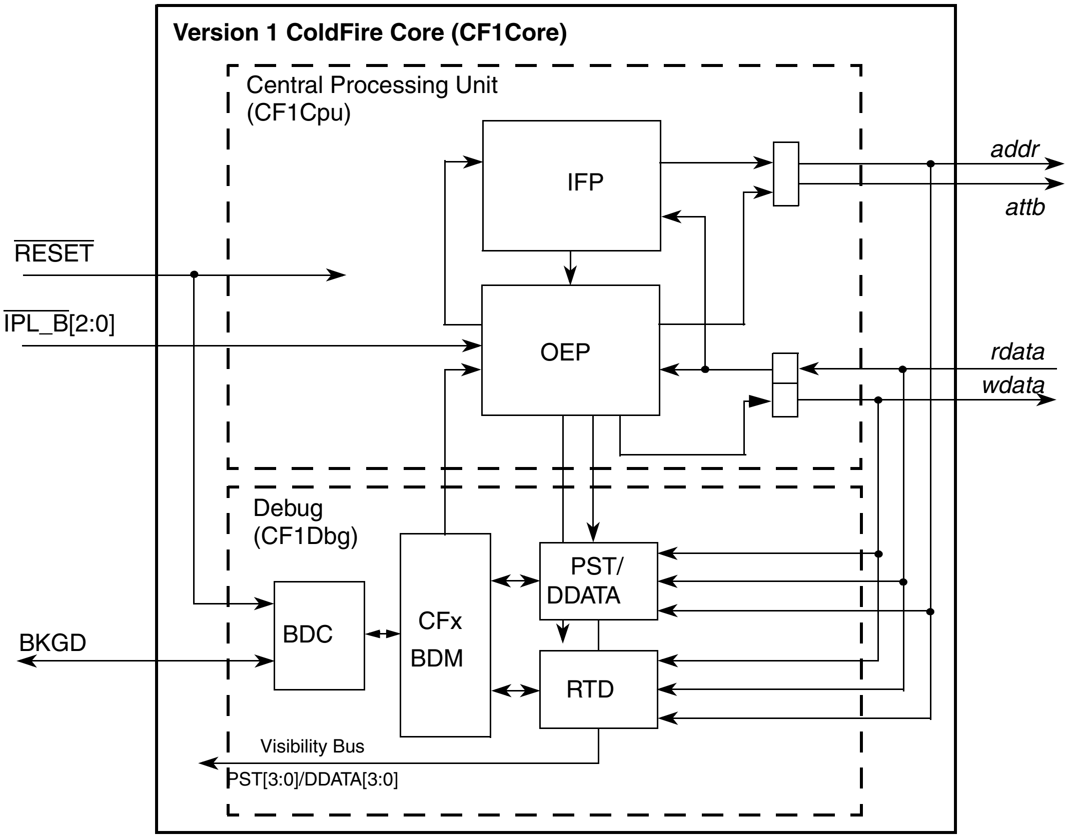
This chapter describes the capabilities defined by the Version 1 ColdFire debug architecture. The Version 1 ColdFire core supports BDM functionality using the HCS08's single-pin interface. The traditional 3-pin full-duplex ColdFire BDM serial communication protocol based on 17-bit data packets is replaced 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 22-1.



IFP	— Instruction fetch pipeline
OEP	— Operand execution pipeline
BDC	— Background debug controller
CFxBDM	— ColdFire background debug module
PST/DDATA	— Processor status/debug data
RTD	— Real-time debug

Figure 22-1. Simplified Version 1 ColdFire Core Block Diagram

22.1.1 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 22.4.1, “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 22.4.2, “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 22.4.4, “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 22.4.3, “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 22-1 summarizes the various debug revisions.

Table 22-1. 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) $\overline{\text{BKPT}}$ configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor’s current PC
B+	1001	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

22.1.2 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

22.1.3 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 22-2.

Table 22-2. BDM Command Types

Command Type	Flash Secure?	BDM?	Core Status	Command Set
Always-available	Secure or Unsecure	Enabled or Disabled	—	• Read/write access to XCSR[31–24], CSR2[31–24], CSR3[31–24]
Non-intrusive	Unsecure	Enabled	Run, Halt	• Memory access • Memory access with status • Debug register access • BACKGROUND
Active background	Unsecure	Enabled	Halt	• Read or write CPU registers (also available in stop mode) • Single-step the application • Exit halt mode to return to the application program (GO)

For more information on these three BDM command classifications, see Section 22.4.1.5, "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 22.3.3, "Configuration/Status Register 2 (CSR2)," for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- A computer operating properly reset occurs and CSR2[COPHR] 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.

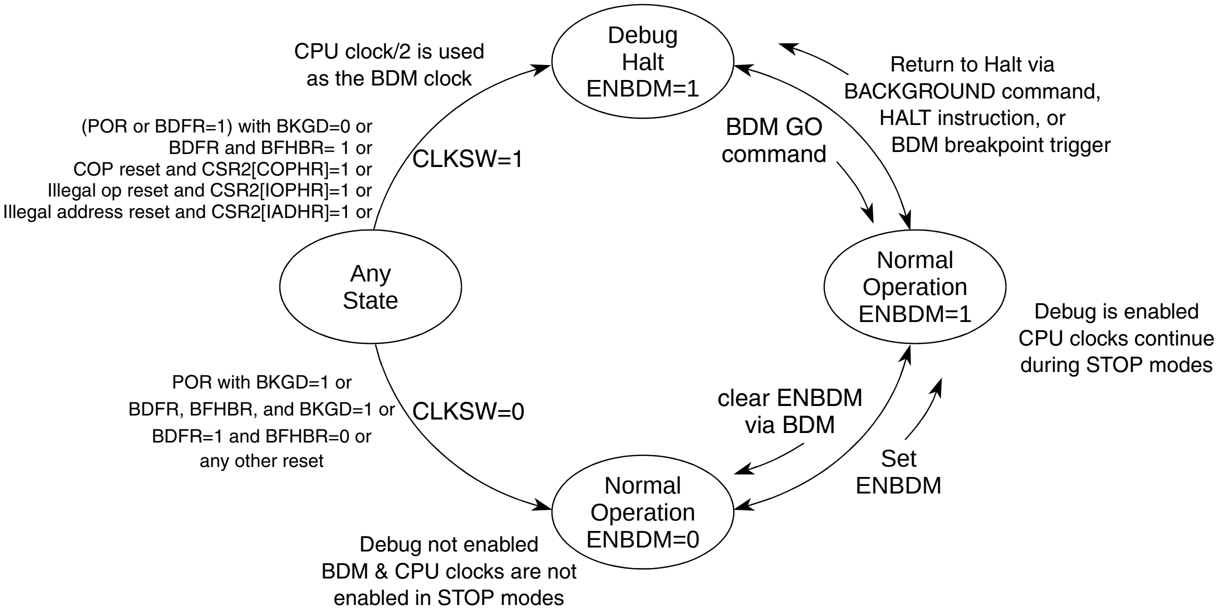

CPU clock/2 is used as the BDM clock
Debug Halt ENBDM=1
Return to Halt via BACKGROUND command, HALT instruction, or BDM breakpoint trigger
(POR or BDFR=1) with BKGD=0 or
BDFR and BFHBR= 1 or
COP reset and CSR2[COPHR]=1 or
Illegal op reset and CSR2[IOPHR]=1 or
Illegal address reset and CSR2[IADHR]=1 or
CLKSW=1
BDM GO command
Any State
Normal Operation ENBDM=1
Debug is enabled CPU clocks continue during STOP modes
POR with BKGD=1 or
BDFR, BFHBR, and BKGD=1 or
BDFR=1 and BFHBR=0 or
any other reset
CLKSW=0
clear ENBDM via BDM
Set ENBDM
Normal Operation ENBDM=0
Debug not enabled BDM & CPU clocks are not enabled in STOP modes

Figure 22-2. Debug Modes State Transition Diagram

Figure 22-2 contains a simplified view of the V1 ColdFire debug mode states. The XCSR[CLKSW] bit controls the BDC clock source. When CLKSW is set, the BDC serial clock frequency is half the CPU clock. When CLKSW 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, CLKSW] 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.

22.2 External Signal Descriptions

Table 22-3 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 22.4.5, “Freescale-Recommended BDM Pinout”.

Table 22-3. 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: • If CSR[VBD] = 0, the VBus is enabled, the PSTCLK*n*, PST[3:0] and DDATA[3:0] outputs are functional and enabled in the appropriate GPIO logic and $\overline{\text{BKPT}}$ is routed into the debug module. • If CSR[VBD] = 1, the VBus is disabled, the PSTCLK*n*, PST and DDATA outputs are all quiescent and the appropriate GPIO logic is disabled. The $\overline{\text{BKPT}}$ pin is logically disconnected from the debug module. See Figure 22-3 for details on the PSTCLK*n* behavior and the derived PSTCLK and Table 22-26 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 22-26 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.

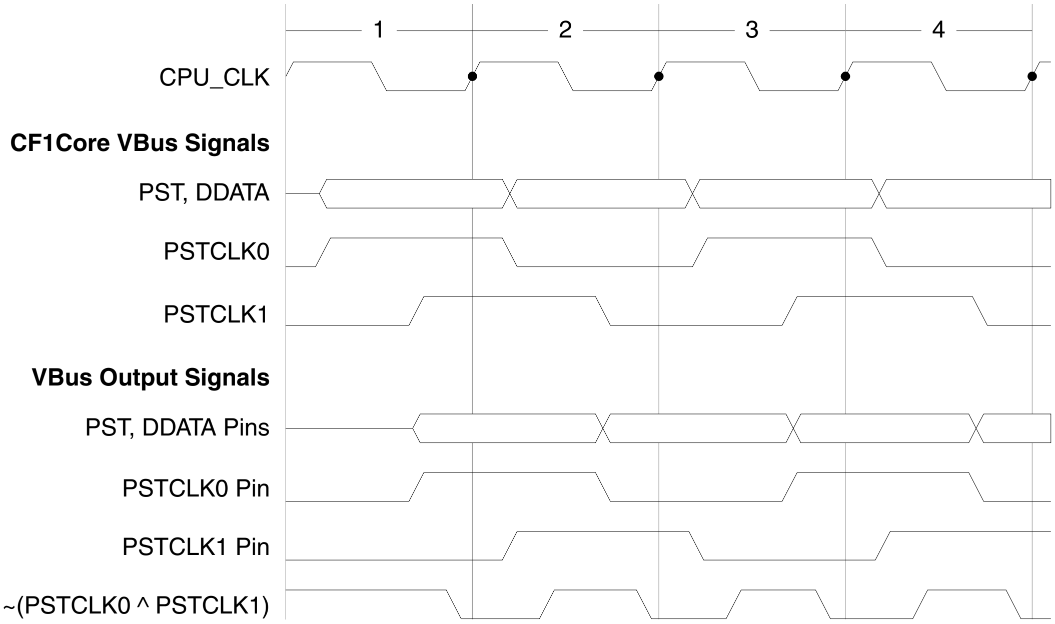


Figure 22-3. Deriving PSTCLK

22.3 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 22-4, 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 22.4.1.5, “BDM Command Set Summary.” These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 22-4.

Table 22-4. 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	22.3.1/22-9
0x01	Extended Configuration/Status Register (XCSR)	32	R/W[1] (BDM), W (CPU)	0x0000_0000	22.3.2/22-12
0x02	Configuration/Status Register 2 (CSR2)	32	R/W[1] (BDM), W (CPU)	See Section	22.3.3/22-15
0x03	Configuration/Status Register 3 (CSR3)	32[2]	R/W[1] (BDM), W (CPU)	0x0000_0000	22.3.4/22-18
0x05	BDM address attribute register (BAAR)	32[2]	W	0x0000_0005	22.3.5/22-19
0x06	Address attribute trigger register (AATR)	32[2]	W	0x0000_0005	22.3.6/22-20
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	22.3.7/22-21
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined, Unaffected	22.3.8/22-24
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined, Unaffected	22.3.8/22-24
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined, Unaffected	22.3.9/22-26
0x0D	Address breakpoint low register (ABLR)	32	W	0x0000_0000	22.3.9/22-26
0x0E	Data breakpoint register (DBR)	32	W	0x0000_0000	22.3.10/22-27
0x0F	Data breakpoint mask register (DBMR)	32	W	0x0000_0000	22.3.10/22-27
0x18	PC breakpoint register 1 (PBR1)	32	W	PBR1[0] = 0	22.3.8/22-24
0x1A	PC breakpoint register 2 (PBR2)	32	W	PBR2[0] = 0	22.3.8/22-24
0x1B	PC breakpoint register 3 (PBR3)	32	W	PBR3[0] = 0	22.3.8/22-24
—	PST Trace Buffer *n* (PSTB*n*); *n* = 0–11 (0xB)	32	R (BDM)[3]	Undefined, Unaffected	22.4.1.5.12/22-48

[1] 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.

[2] Each debug register is accessed as a 32-bit value; undefined fields are reserved and must be cleared.

[3] 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.

22.3.1 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 22-4. Configuration/Status Register (CSR)

Table 22-5. 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 × BSTAT) and the visibility bus PST value is 2 × 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: • The $\overline{\text{BKPT}}$ input was asserted, • BDM BACKGROUND command received, or • The PSTB halt on full condition, CSR2[PSTBH], sets. 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*n*, PST[3:0], DDATA[3:0] and $\overline{\text{BKPT}}$ pins are enabled and operational. 1 VBus disabled. PSTCLKn, 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 22-5. 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 Table 22-27. 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 Section 22.4.4.1, “Begin Execution of Taken Branch (PST = 0x05).” 00 No target address capture 01 Lower 2 bytes of the target address 1*x* 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 22-5. CSR Field Descriptions (continued)

Field	Description
1 FID	Force *ipg_debug*. The core generates this output to the device, signaling it is in debug mode. 0 Do not force the assertion of *ipg_debug* 1 Force the assertion of *ipg_debug*
0 DDH	Disable *ipg_debug* 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 *ipg_debug* if the core is halted 1 Negate *ipg_debug* due to the core being halted

22.3.2 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 22-6.

Table 22-6. 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	EN BDM	0	0	0	0	0	0	0	0
W			ESEQC				ERASE									
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 22-5. Extended Configuration/Status Register (XCSR)

Table 22-7. XCSR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31
CPUHALT</td><td>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.
<table>
<tr><th>XCSR [CPUHALT]</th><th>XCSR [CPUSTOP]</th><th>CPU State</th></tr>
<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>
</table></td></tr>
<tr><td>30
CPUSTOP</td><td>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.</td></tr>
<tr><td>29–27
CSTAT (R)
ESEQC (W)</td><td>During reads, indicates the BDM command status.
000 Command done, no errors
001 Command done, data invalid
01<i>x</i> Command done, illegal
1<i>xx</i> Command busy, overrun

If an overrun is detected (CSTAT = 1<i>xx</i>), the following sequence is suggested to clear the source of the error:
1. Issue a SYNC command to reset the BDC channel.
2. The host issues a BDM NOP command.
3. The host checks the channel status using a READ_XCSR_BYTE command.
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

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.
000 User mass erase
Else Reserved
<b>Note:</b> See the Memory chapter for a detailed description of the algorithm for clearing security.</td></tr>
<tr><td>26
CLKSW</td><td>Select source for serial BDC communication clock.
0 Alternate, asynchronous BDC clock, typically 10 MHz
1 Synchronous bus clock (CPU clock divided by 2)

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 Figure 22-2.</td></tr>
</table>

Table 22-7. XCSR Field Descriptions (continued)

Field	Description
25 SEC (R) ERASE (W)	The read value of this bit typically defines the status of the flash security field. 0 Flash security is disabled 1 Flash security is enabled 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. 0 Flash is not busy performing a BDM mass-erase sequence 1 Flash is busy performing a BDM mass-erase sequence During writes, this bit qualifies XCSR[ESEQC] for the write modes shown in the ESEQC field description. 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.
24 ENBDM	Enable BDM. 0 BDM mode is disabled 1 Active background mode is enabled (assuming the flash is not secure)
23–3	Reserved for future use by the debug module, must be cleared.

Table 22-7. XCSR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2–1
APCSC</td><td>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.

The chosen frequency depends on CSR2[APCDIV16] as shown in the equation and table below:

$$\text{PC address capture period} = \frac{2^{(\text{APCSC} + 1)} \times 1024}{16^{\text{APCDIV16}}} \qquad \textbf{\textit{Eqn. 22-1}}$$

<table>
<tr><th>XCSR
[APCENB]</th><th>CSR2
[APCDIV16]</th><th>XCSR
[APCSC]</th><th>SYNC_PC Interval</th></tr>
<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>
</table>
</td></tr>
<tr><td>0
APCENB</td><td>Automatic PC synchronization enable. Enables the periodic output of the PC which can be used for PST/DDATA trace synchronization and code profiling.
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.
0 Automatic PC synchronization disabled
1 Automatic PC synchronization enabled</td></tr>
</table>

22.3.3 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 22-8.

Table 22-8. 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 22-8. 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	COP HR	IOP HR	IAD HR	0	BFHBR	0	PSTBH	PSTBST		0	D1HRL			
W								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 DIV16	0	PSTBRM		PSTBSS		
W									PSTBR							
Reset	Unaffected and Undefined								0	0	0	0	0	0	0	0

Figure 22-6. Configuration/Status Register 2 (CSR2)

Table 22-9. 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 COPHR	Computer operating properly halt after reset. Determines operation of the device after a COP reset. This bit is cleared after a power-on reset and is unaffected by any other reset. 0 After a computer operating properly reset, the device immediately enters normal operation mode. 1 A computer operating properly reset immediately halts the device (as if the BKGD pin was held low after a power-on reset). **Note:** 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). **Note:** 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). **Note:** This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure.

Table 22-9. CSR2 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>26</td><td>Reserved, must be cleared.</td></tr>
<tr><td>25
BFHBR</td><td>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.
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.
<b>Note:</b> This bit can only change state if XCSR[ENBDM] = 1 and the flash is unsecure.</td></tr>
<tr><td>24
BDFR</td><td>Background debug force reset. Forces a BDM reset to the device. This bit always reads as 0 after the reset has been initiated.
0 No reset initiated.
1 Force a BDM reset.</td></tr>
<tr><td>23
PSTBH</td><td>PST trace buffer halt. Indicates if the processor is halted due to the PST trace buffer being full when recording in a obtrusive mode.
0 PST trace buffer not full
1 CPU halted due to PST trace buffer being full in obtrusive mode</td></tr>
<tr><td>22–21
PSTBST</td><td>PST trace buffer state. Indicates the current state of the PST trace buffer recording.
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</td></tr>
<tr><td>20</td><td>Reserved, must be cleared.</td></tr>
<tr><td>19–16
D1HRL</td><td>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.</td></tr>
<tr><td>15–8
PSTBWA</td><td>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.
The msb of this field can be used to determine if the entire PST trace buffer has been loaded with valid data.
<table>
<tr><th>PSTBWA[7]</th><th>PSTB Valid Data Locations
(Oldest to Newest)</th></tr>
<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>
</table>
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.
<b>Note:</b> Since this device contains a 64-entry trace buffer, PSTBWA[6] is always zero.</td></tr>
<tr><td>7
PSTBR</td><td>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.
0 Do not force a PST trace buffer reset
1 Force a PST trace buffer reset</td></tr>
</table>

Table 22-9. CSR2 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>6
APCDIV16</td><td>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.</td></tr>
<tr><td>5</td><td>Reserved, must be cleared.</td></tr>
<tr><td>4–3
PSTBRM</td><td>PST trace buffer recording mode. Defines the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field.
00 Non-obstrusive, 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]).
The terms obtrusive and non-obtrusive are defined as:
• 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.
• 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.</td></tr>
<tr><td>2–0
PSTBSS</td><td>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.
<table>
<tr><th>PSTBSS</th><th>Start Condition</th><th>Stop Condition</th></tr>
<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">AB<i>x</i>R{& 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>AB<i>x</i>R{& DBR/DBMR}</td></tr>
<tr><td>101</td><td>PBR1</td></tr>
<tr><td>110</td><td rowspan="2">PBR1</td><td>AB<i>x</i>R{& DBR/DBMR}</td></tr>
<tr><td>111</td><td>PBR0/PBMR</td></tr>
</table>
</td></tr>
</table>

22.3.4 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 22-10.

Table 22-10. 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.

Table 22-10. CSR3 Reference Summary (continued)

Method	Reference Details
READ_DREG	Reads CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
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
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 22-7. Configuration/Status Register 3 (CSR3)

Table 22-11. 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 μs. The automated programming logic uses an integer number of these pulses to complete an erase or program operation. if BFCDIV8 = 0, then $f_{FCLK} = f_{Bus} \div (BFCDIV + 1)$ if BFCDIV8 = 1, then $f_{FCLK} = f_{Bus} \div (8 \times (BFCDIV + 1)$ where f_{FCLK} is the frequency of the flash clock and f_{Bus} is the frequency of the bus clock.
23–0	Reserved for future use by the debug module, must be cleared.

22.3.5 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 22-8. BDM Address Attribute Register (BAAR)

Table 22-12. 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, Section 22.3.6, “Address Attribute Trigger Register (AATR)”.
2–0 TM	Transfer modifier. See the TM definition in the AATR description, Section 22.3.6, “Address Attribute Trigger Register (AATR)”.

22.3.6 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 22-9. Address Attribute Trigger Register (AATR)

Table 22-13. 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

22.3.7 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

			Second Level Trigger													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TRC		L2EBL	L2ED							L2DI	L2EA			L2EPC	L2PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

			First Level Trigger													
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	L2T	L1T	L1EBL	L1ED							L1DI	L1EA			L1EPC	L1PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-10. Trigger Definition Register (TDR)

Table 22-14. 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.

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 22-14. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>21
L2DI</td><td>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.</td></tr>
<tr><td>20–18
L2EA</td><td>Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>17
L2EPC</td><td>Enable level 2 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>16
L2PCI</td><td>Level 2 PC breakpoint invert.
0 The PC breakpoint is defined within the region defined by PBR*n* and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR*n* and PBMR.</td></tr>
<tr><td>15
L2T</td><td>Level 2 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range 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)</td></tr>
<tr><td>14
L1T</td><td>Level 1 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range 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)</td></tr>
<tr><td>13
L1EBL</td><td>Enable level 1 breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
</table>

Table 22-14. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12–6
L1ED</td><td>Enable level 1 data breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor's local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>5
L1DI</td><td>Level 1 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>4–2
L1EA</td><td>Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>1
L1EPC</td><td>Enable level 1 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>0
L1PCI</td><td>Level 1 PC breakpoint invert.
0 The PC breakpoint is defined within the region defined by PBR<i>n</i> and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR<i>n</i> and PBMR.</td></tr>
</table>

22.3.8 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.

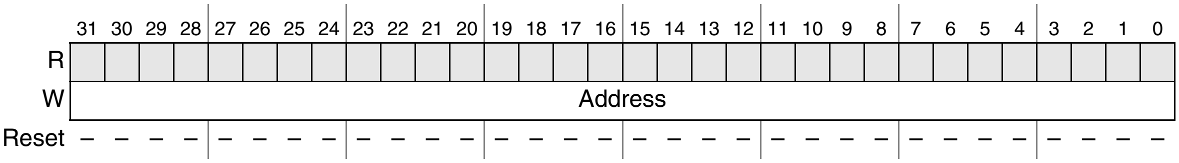
The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE_DREG command using values shown in Section 22.4.1.4, "BDM Command Set Descriptions".

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.

DRc: 0x08 (PBR0)

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	Address																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

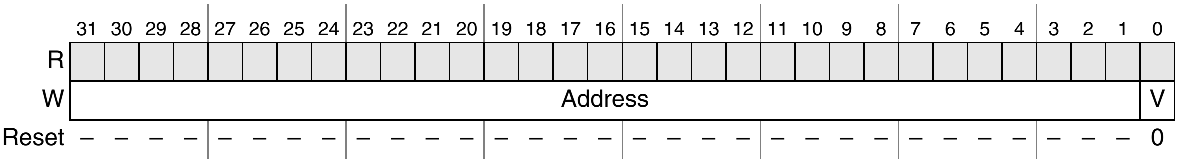
Figure 22-11. Program Counter Breakpoint Register 0 (PBR0)

Table 22-15. 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.

DRc: 0x18 (PBR1)
0x1A (PBR2)
0x1C (PBR3)

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	Address																															V
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0

Figure 22-12. Program Counter Breakpoint Register *n* (PBR*n, n = 1,2,3*)

Table 22-16. 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 22-13 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.

DRc: 0x09 (PBMR)

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	Mask																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 22-13. Program Counter Breakpoint Mask Register (PBMR)

Table 22-17. 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.

22.3.9 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 22.4.1.4, "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.

DRc: 0x0C (ABHR)
0x0D (ABLR)

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	Address																															
ABHR Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ABLR 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 22-14. Address Breakpoint Registers (ABLR, ABHR)

Table 22-18. ABLR Field Description

Field	Description
31–0 Address	Low address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific addresses are programmed into ABLR.

Table 22-19. ABHR Field Description

Field	Description
31–0 Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

22.3.10 Data Breakpoint and Mask Registers (DBR, DBMR)

DBR specifies data patterns used as part of the trigger into debug mode. DBR bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR and DBMR are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE_DREG commands.

DRc[4:0]: 0x0E (DBR)
0x0F (DBMR)

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	Data (DBR); Mask (DBMR)																															
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 22-15. Data Breakpoint & Mask Registers (DBR, DBMR)

Table 22-20. DBR Field Descriptions

Field	Description
31–0 Data	Data breakpoint value. Contains the value to be compared with the data value from the processor's local bus as a breakpoint trigger.

Table 22-21. DBMR Field Descriptions

Field	Description
31–0 Mask	Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. 0 The corresponding DBR bit is compared to the appropriate bit of the processor's local data bus 1 The corresponding DBR bit is ignored

The DBR supports aligned and misaligned references. Table 22-22 shows the relationships between processor address, access size, and location within the 32-bit data bus.

Table 22-22. 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]
0*x*	Word	D[31–16]
1*x*	Word	D[15–0]
xx	Longword	D[31–0]

22.3.10.1 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 22.3.3, “Configuration/Status Register 2 (CSR2)”.

22.3.11 PST Buffer (PSTB)

The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. See Figure 22-16 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.

<table>
<tr><th>Core register number (CRN)</th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>0x10</td><td colspan="6">TB #00</td><td colspan="6">TB #01</td><td colspan="6">TB #02</td><td colspan="6">TB #03</td><td colspan="6">TB #04</td><td colspan="2">05[5:4]</td></tr>
<tr><td>0x11</td><td colspan="4">TB #05[3:0]</td><td colspan="6">TB #06</td><td colspan="6">TB #07</td><td colspan="6">TB #08</td><td colspan="6">TB #09</td><td colspan="4">TB #10[5:2]</td></tr>
<tr><td>0x12</td><td colspan="2">10[1:0]</td><td colspan="6">TB #11</td><td colspan="6">TB #12</td><td colspan="6">TB #13</td><td colspan="6">TB #14</td><td colspan="6">TB #15</td></tr>
<tr><td>0x13</td><td colspan="6">TB #16</td><td colspan="6">TB #17</td><td colspan="6">TB #18</td><td colspan="6">TB #19</td><td colspan="6">TB #20</td><td colspan="2">21[5:4]</td></tr>
<tr><td>0x14</td><td colspan="4">TB #21[3:0]</td><td colspan="6">TB #22</td><td colspan="6">TB #23</td><td colspan="6">TB #24</td><td colspan="6">TB #25</td><td colspan="4">TB #26[5:2]</td></tr>
<tr><td>0x15</td><td colspan="2">26[1:0]</td><td colspan="6">TB #27</td><td colspan="6">TB #28</td><td colspan="6">TB #29</td><td colspan="6">TB #30</td><td colspan="6">TB #31</td></tr>
<tr><td>0x16</td><td colspan="6">TB #32</td><td colspan="6">TB #33</td><td colspan="6">TB #34</td><td colspan="6">TB #35</td><td colspan="6">TB #36</td><td colspan="2">37[5:4]</td></tr>
<tr><td>0x17</td><td colspan="4">TB #37[3:0]</td><td colspan="6">TB #38</td><td colspan="6">TB #39</td><td colspan="6">TB #40</td><td colspan="6">TB #41</td><td colspan="4">TB #42[5:2]</td></tr>
<tr><td>0x18</td><td colspan="2">42[1:0]</td><td colspan="6">TB #43</td><td colspan="6">TB #44</td><td colspan="6">TB #45</td><td colspan="6">TB #46</td><td colspan="6">TB #47</td></tr>
<tr><td>0x19</td><td colspan="6">TB #48</td><td colspan="6">TB #49</td><td colspan="6">TB #50</td><td colspan="6">TB #51</td><td colspan="6">TB #52</td><td colspan="2">53[5:4]</td></tr>
<tr><td>0x1A</td><td colspan="4">TB #53[3:0]</td><td colspan="6">TB #54</td><td colspan="6">TB #55</td><td colspan="6">TB #56</td><td colspan="6">TB #57</td><td colspan="4">TB #58[5:2]</td></tr>
<tr><td>0x1B</td><td colspan="2">58[1:0]</td><td colspan="6">TB #59</td><td colspan="6">TB #60</td><td colspan="6">TB #61</td><td colspan="6">TB #62</td><td colspan="6">TB #63</td></tr>
</table>

Figure 22-16. PST Trace Buffer Entries and Locations

22.4 Functional Description

22.4.1 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 22-1, 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.

22.4.1.1 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 22-23. 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 22-23. 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 ≥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: • Issue a BDM reset setting CSR2[BDFR] with CSR2[BDHBR] cleared and the BKGD pin held high to reset into normal operating mode • Erase the flash to unsecure the memory and then proceed with debug • Power cycle the device with the BKGD pin held high to reset into the normal operating mode	

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].

22.4.1.2 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 22.4.1.3, “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 22.4.1.3, “BDM Communication Details,” for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector (Section 22.4.5, “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].

22.4.1.3 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 22-7.

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 22-17 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.

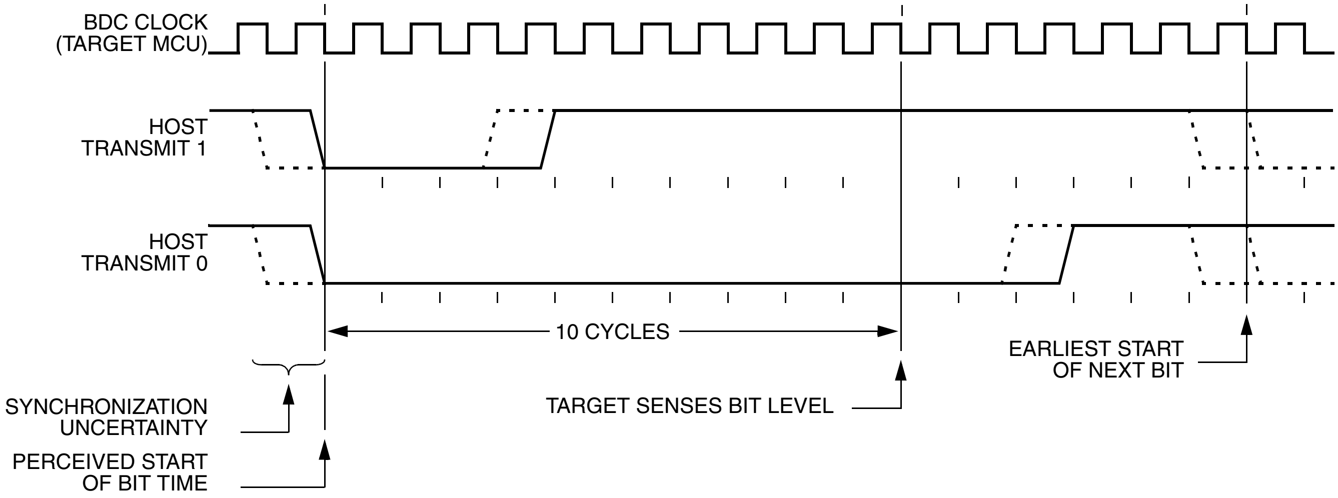


Figure 22-17. BDC Host-to-Target Serial Bit Timing

Figure 22-18 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.

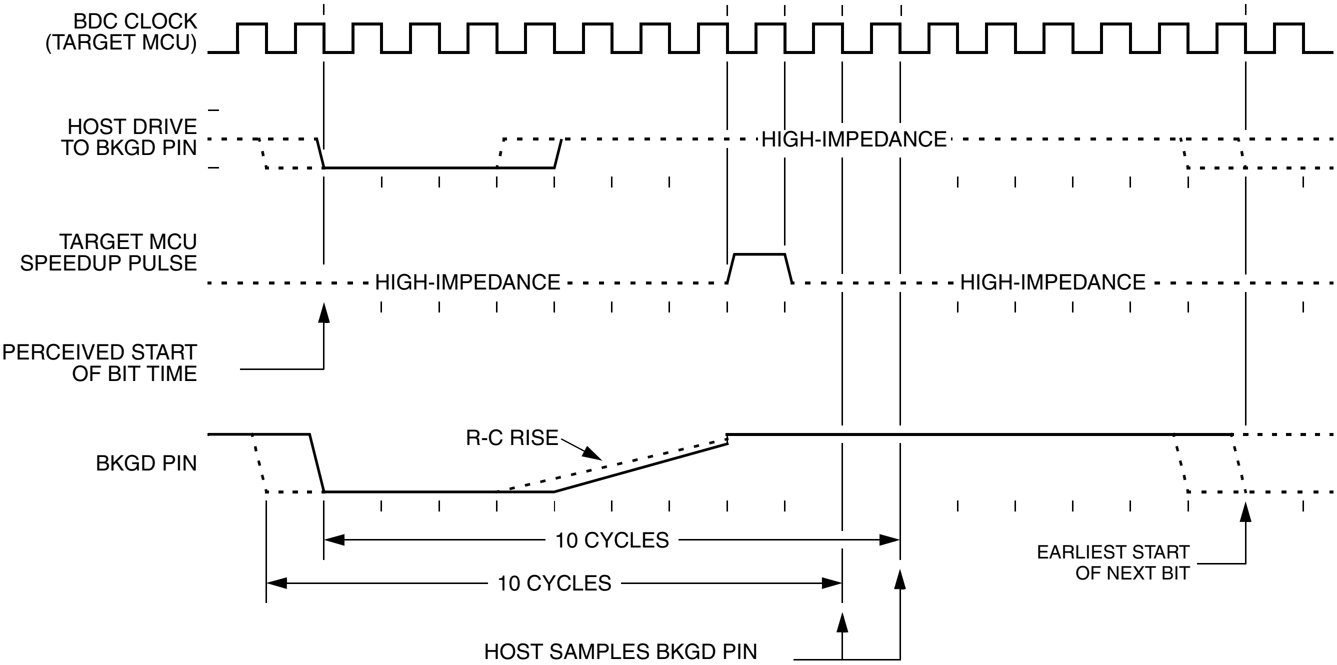


Figure 22-18. BDC Target-to-Host Serial Bit Timing (Logic 1)

Figure 22-19 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.

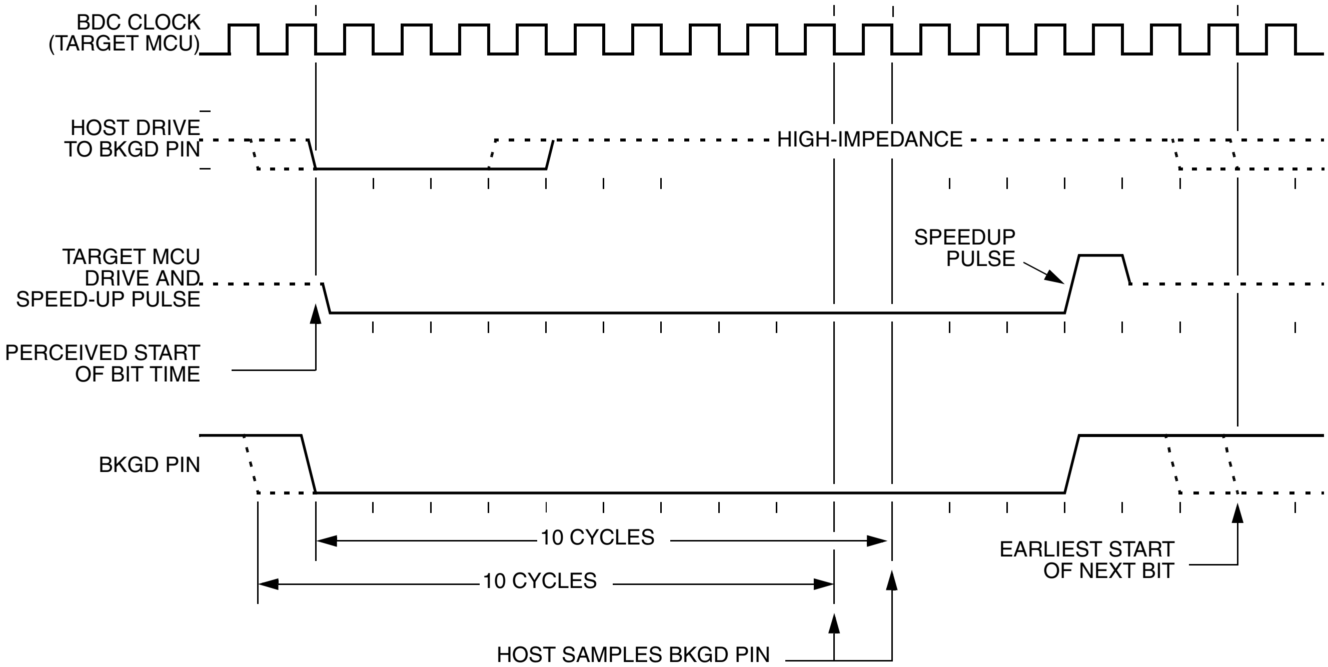


Figure 22-19. BDM Target-to-Host Serial Bit Timing (Logic 0)

22.4.1.4 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 22-20.

Miscellaneous Commands

	7	6	5	4	3	2	1	0
W	0	0	R/$\overline{W}$	0	MSCMD			
R/W	Optional Command Extension Byte (Data)							

Memory Commands

	7	6	5	4	3	2	1	0
W	0	0	R/$\overline{W}$	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/$\overline{W}$	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 Figure 22-16							
R	Trace Buffer Data[23—16], see Figure 22-16							
R	Trace Buffer Data[15—08], see Figure 22-16							
R	Trace Buffer Data[07—00], see Figure 22-16							

Figure 22-20. BDM Command Encodings

Table 22-24. BDM Command Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
R/$\overline{W}$</td><td>Read/$\overline{\text{Write}}$.
0 Command is performing a write operation.
1 Command is performing a read operation.</td></tr>
<tr><td>3–0
MSCMD</td><td>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</td></tr>
<tr><td>3–2
SZ</td><td>Memory operand size. Defines the size of the memory reference.
00 8-bit byte
01 16-bit word
10 32-bit long</td></tr>
<tr><td>1–0
MCMD</td><td>Memory command. Defines the type of the memory reference to be performed.
00 Simple write if R/$\overline{W}$ = 0; simple read if R/$\overline{W}$ = 1
01 Write + status if R/$\overline{W}$ = 0; read + status if R/$\overline{W}$ = 1
10 Fill if R/$\overline{W}$ = 0; dump if R/$\overline{W}$ = 1
11 Fill + status if R/$\overline{W}$ = 0; dump + status if R/$\overline{W}$ = 1</td></tr>
<tr><td>7–6
CRG</td><td>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,...)</td></tr>
<tr><td>4–0
CRN</td><td>Core register number. Defines the specific core register (its number) to be referenced. All other CRN values are reserved.
<table>
<tr><th>CRG</th><th>CRN</th><th>Register</th></tr>
<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 Table 22-4</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>
</table>
</td></tr>
</table>

22.4.1.5 BDM Command Set Summary

Table 22-25 summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in Table 22-25 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 22-25. BDM Command Summary

Command Mnemonic	Command Classification	ACK if Enb?[1]	Command Structure	Description
SYNC	Always Available	N/A	N/A[2]	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 22-25. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb?[1]	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[3]
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 22-25. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb?[1]	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

[1] This column identifies if the command generates an ACK pulse if operating with acknowledge mode enabled. See Section 22.4.1.5.3, "ACK_ENABLE," for addition information.

[2] The SYNC command is a special operation which does not have a command code.

[3] If a GO command is received while the processor is not halted, it performs no operation.

22.4.1.5.1 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:

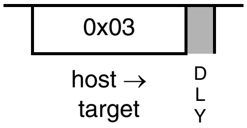
1. Waits for BKGD to return to a logic high.

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.

22.4.1.5.2 ACK_DISABLE

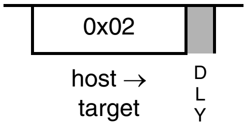
Disable host/target handshake protocol **Always Available**



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.

22.4.1.5.3 ACK_ENABLE

Enable host/target handshake protocol **Always Available**

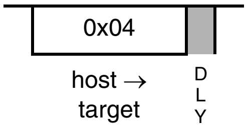


Enables the hardware handshake protocol in the serial communication. The hardware handshake is implemented by an acknowledge (ACK) pulse issued by the target MCU in response to a host command. The ACK_ENABLE command is interpreted and executed in the BDC logic without the need to interface with the CPU. 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 22.4.1.6, “Serial Interface Hardware Handshake Protocol,” and Section 22.4.1.7, “Hardware Handshake Abort Procedure.”

22.4.1.5.4 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.

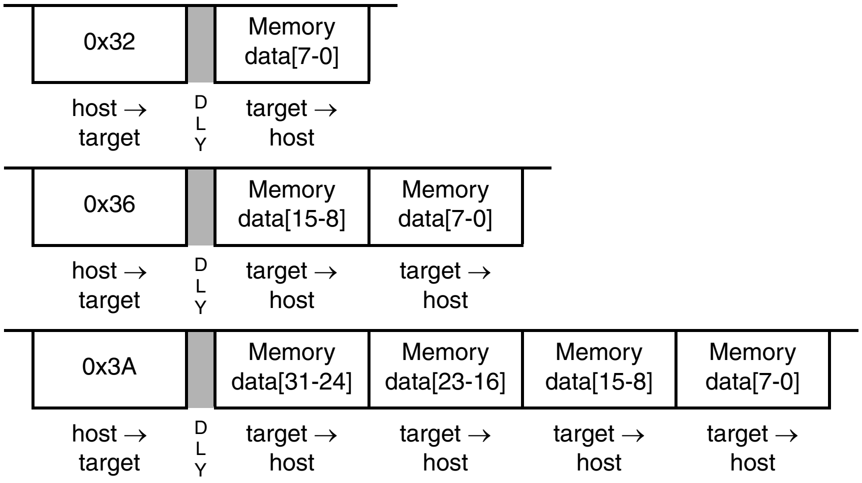
A delay of 32 BDC clock cycles is required after the BACKGROUND command to allow the target MCU to finish its current CPU instruction and enter active background mode before a new BDC command can be accepted.

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.

22.4.1.5.5 DUMP_MEM.sz, DUMP_MEM.sz_WS

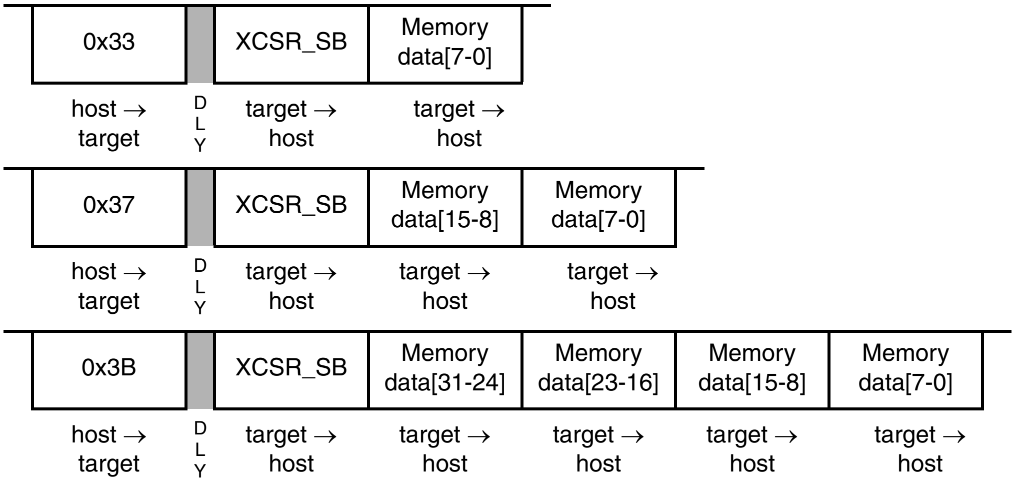
DUMP_MEM.sz

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

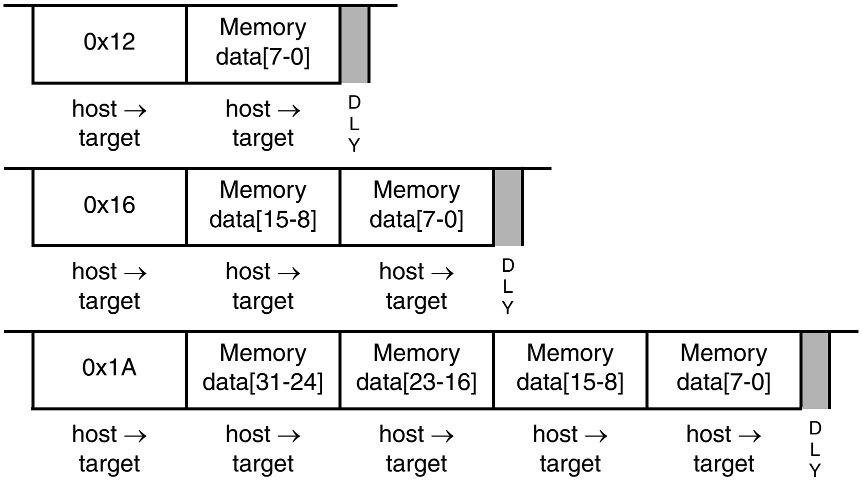
> DUMP_MEM{_WS} does not check for a valid address; it is a valid command only when preceded by NOP, READ_MEM{_WS}, or another DUMP_MEM{_WS} command. Otherwise, an illegal command response is returned. NOP can be used for inter-command padding without corrupting the address pointer.

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.

22.4.1.5.6 FILL_MEM.sz, FILL_MEM.sz_WS

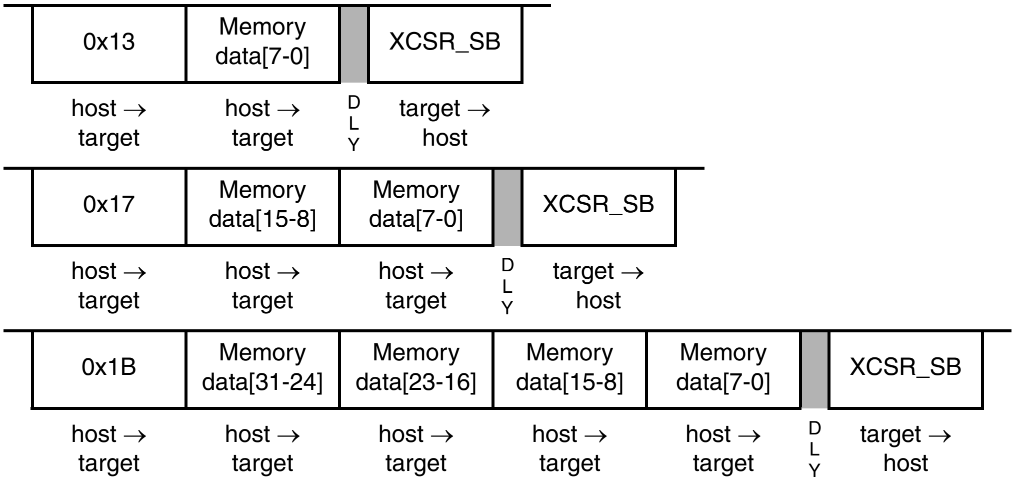
FILL_MEM.sz

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.

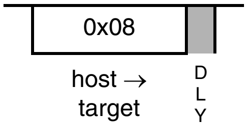
NOTE

> 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.

22.4.1.5.7 GO

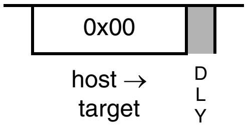
Go **Non-intrusive**



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.

22.4.1.5.8 NOP

No operation **Non-intrusive**



NOP performs no operation and may be used as a null command where required.

22.4.1.5.9 READ_CREG

Read CPU control register **Active Background**

0xE0+CRN		CREG data [31-24]	CREG data [23-16]	CREG data [15-8]	CREG data [7-0]
host → target	D L Y	target → host	target → host	target → host	target → host

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 22-24 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.

22.4.1.5.10 READ_DREG

Read debug control register **Non-intrusive**

0xA0+CRN		DREG data [31-24]	DREG data [23-16]	DREG data [15-8]	DREG data [7-0]
host → target	D L Y	target → host	target → host	target → host	target → host

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 22-4 for CRN details.

22.4.1.5.11 READ_MEM.sz, READ_MEM.sz_WS

READ_MEM.sz

Read memory at the specified address **Non-intrusive**

0x30	Address[23-0]		Memory data[7-0]
host → target	host → target	D L Y	target → host

0x34	Address[23-0]		Memory data[15-8]	Memory data[7-0]
host → target	host → target	D L Y	target → host	target → host

0x38	Address[23-0]		Memory data[31-24]	Memory data[23-16]	Memory data[15-8]	Memory data[7-0]
host → target	host → target	D L Y	target → host	target → host	target → host	target → host

READ_MEM.sz_WS

Read memory at the specified address with status **Non-intrusive**

0x31	Address[23-0]		XCSR_SB	Memory data[7-0]
host → target	host → target	D L Y	target → host	target → host

0x35	Address[23-0]		XCSR_SB	Memory data [15-8]	Memory data [7-0]
host → target	host → target	D L Y	target → host	target → host	target → host

0x39	Address[23-0]		XCSR_SB	Memory data[31-24]	Memory data[23-16]	Memory data [15-8]	Memory data [7-0]
host → target	host → target	D L Y	target → host	target → host	target → host	target → host	target → host

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.

22.4.1.5.12 READ_PSTB

Read PST trace buffer at the specified address **Non-intrusive**

0x40+CRN		PSTB data [31-24]	PSTB data [23-16]	PSTB data [15-8]	PSTB data [7-0]
host → target	D L Y	target → host	target → host	target → host	target → host

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 22-16 for an illustration of how the buffer entries are packed.

22.4.1.5.13 READ_Rn

Read general-purpose CPU register **Active Background**

0x60+CRN		Rn data [31–24]	Rn data [23–16]	Rn data [15–8]	Rn data [7–0]
host → target	D L Y	target → host	target → host	target → host	target → host

If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See Table 22-24 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.

22.4.1.5.14 READ_XCSR_BYTE

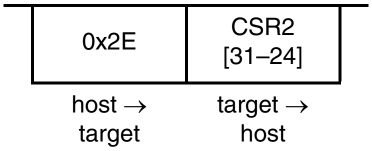
Read XCSR Status Byte **Always Available**

0x2D	XCSR [31–24]
host → target	target → host

Read the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

22.4.1.5.15 READ_CSR2_BYTE

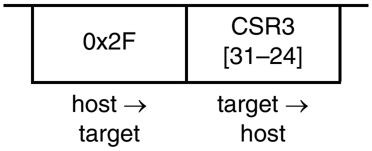
Read CSR2 Status Byte **Always Available**



Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

22.4.1.5.16 READ_CSR3_BYTE

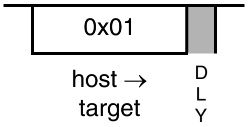
Read CSR3 Status Byte **Always Available**



Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

22.4.1.5.17 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.

22.4.1.5.18 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	DLY

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 22-24 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.

22.4.1.5.19 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	DLY

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*n*, PBMR, AB*x*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 22-4 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.

22.4.1.5.20 WRITE_MEM.sz, WRITE_MEM.sz_WS

WRITE_MEM.sz

Write memory at the specified address **Non-intrusive**

0x10	Address[23-0]	Memory data[7–0]	
host → target	host → target	host → target	DLY

0x14	Address[23-0]	Memory data[15–8]	Memory data[7–0]	
host → target	host → target	host → target	host → target	DLY

0x18	Address[23-0]	Memory data[31–24]	Memory data[23–16]	Memory data[15–8]	Memory data[7–0]	
host → target	host → target	host → target	host → target	host → target	host → target	DLY

WRITE_MEM.sz_WS

Write memory at the specified address with status **Non-intrusive**

0x11	Address[23-0]	Memory data[7–0]		XCSR_SB
host → target	host → target	host → target	DLY	target → host

0x15	Address[23-0]	Memory data[15–8]	Memory data[7–0]		XCSR_SB
host → target	host → target	host → target	host → target	DLY	target → host

0x19	Address[23-0]	Memory data[31–24]	Memory data[23–16]	Memory data[15–8]	Memory data[7–0]		XCSR_SB
host → target	host → target	host → target	host → target	host → target	host → target	DLY	target → host

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.

22.4.1.5.21 WRITE_Rn

Write general-purpose CPU register **Active Background**

0x40+CRN	Rn data [31–24]	Rn data [23–16]	Rn data [15–8]	Rn 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 CPU general-purpose register (An, Dn). See Table 22-24 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.

22.4.1.5.22 WRITE_XCSR_BYTE

Write XCSR Status Byte **Always Available**

0x0D	XCSR Data [31–24]
host → target	host → target

Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

22.4.1.5.23 WRITE_CSR2_BYTE

Write CSR2 Status Byte **Always Available**

0x0E	CSR2 Data [31–24]
host → target	host → target

Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

22.4.1.5.24 WRITE_CSR3_BYTE

Write CSR3 Status Byte **Always Available**

0x0F	CSR3 Data [31–24]
host → target	host → target

Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

22.4.1.6 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 CLKSW 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 22-21. 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.

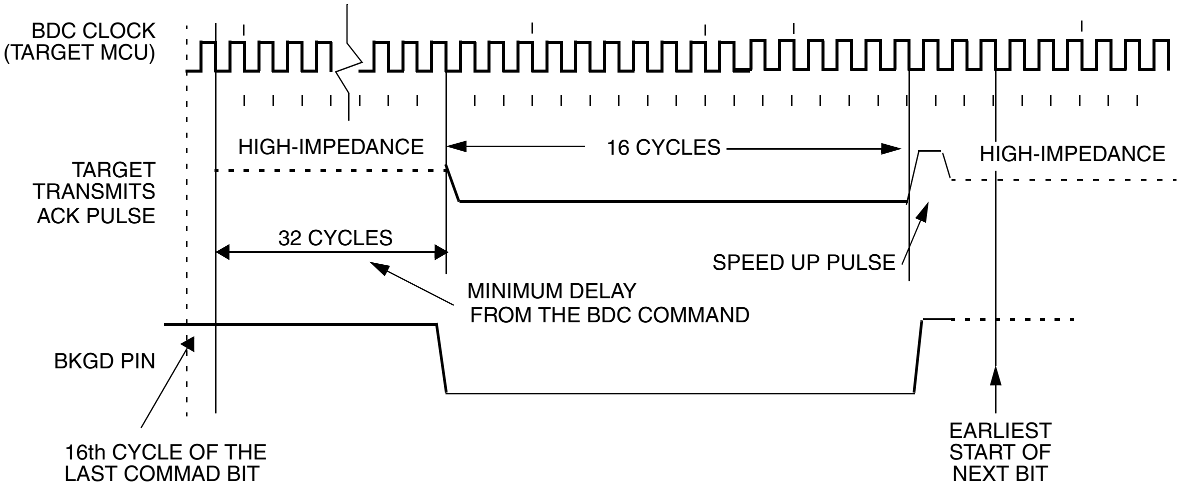


Figure 22-21. Target Acknowledge Pulse (ACK)

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 22-22 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.

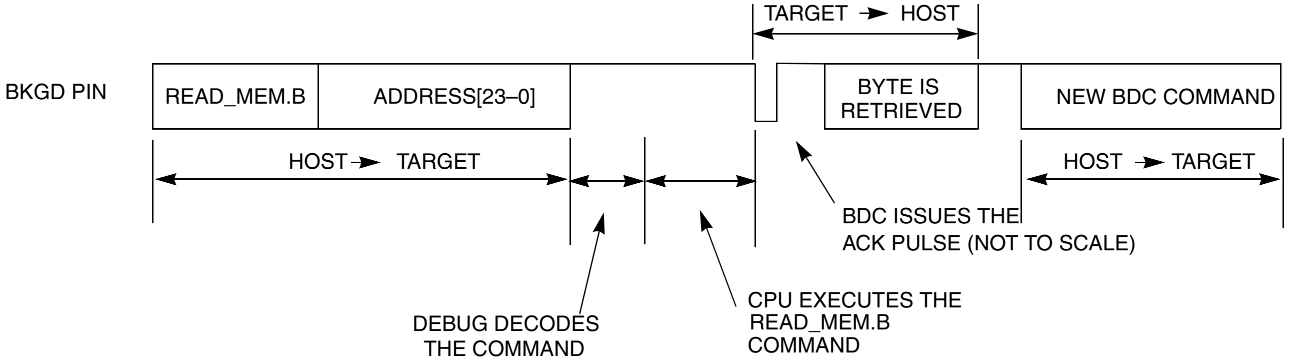


Figure 22-22. Handshake Protocol at Command Level

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 22-22 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 22.4.1.7, “Hardware Handshake Abort Procedure.”

22.4.1.7 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 22.4.1.5.1, “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 1*xx*, 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 22-23 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 22-23 signal timing is not drawn to scale.

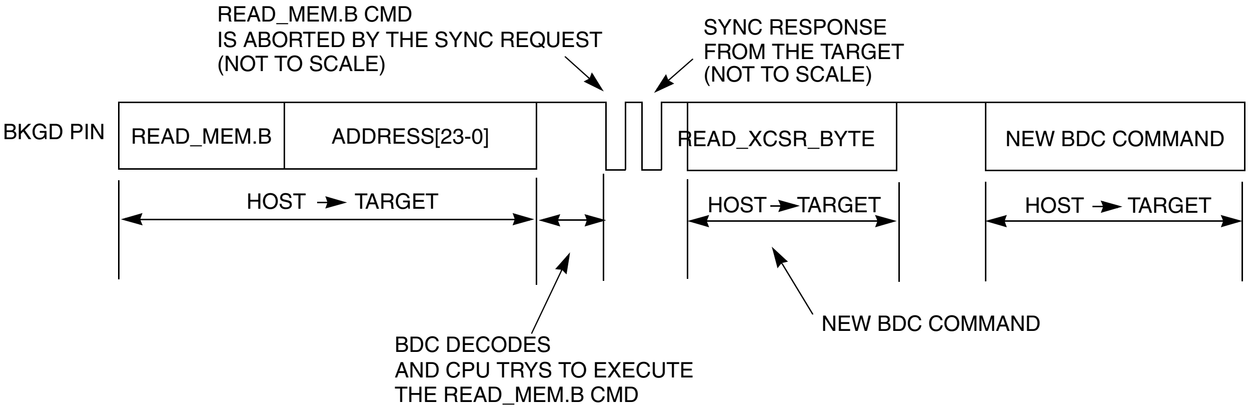


Figure 22-23. ACK Abort Procedure at the Command Level

Figure 22-24 a 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.

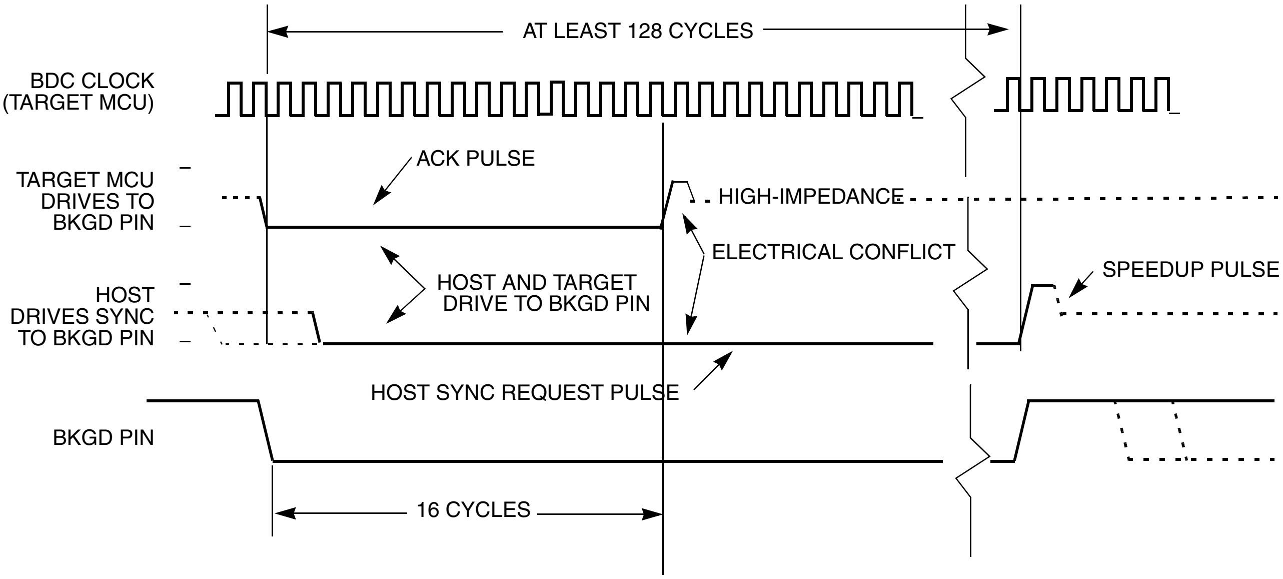


Figure 22-24. ACK Pulse and SYNC Request Conflict

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 22-25 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.

22.4.2 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.

22.4.3 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 22.4.4.1, “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 22-26 shows the encoding of these signals.

Table 22-26. 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 22-26. 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 Section 22.4.3.1, “Begin Execution of Taken Branch (PST = 0x5)”. Also indicates that the SYNC_PC command has been issued.
0x6	Reserved
0x7	Begin execution of return from exception (RTE) instruction.
0x8– 0xB	Indicates the number of bytes to be displayed on the DDATA port on subsequent clock cycles. The value is driven onto the PST port one PSTCLK cycle before the data is displayed. 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 Section 22.4.1.1, “CPU Halt”.

22.4.3.1 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 22-26 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.

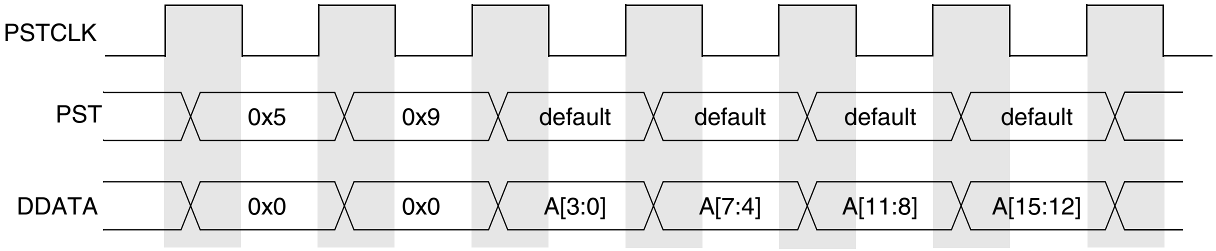


Figure 22-25. Example JMP Instruction Output on PST/DDATA

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.

22.4.4 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 22.4.3, "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 22-27.

Table 22-27. 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 22-27. 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 × CSR[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.

22.4.4.1 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 22-26 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 22-26. 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 22.4.4.2, “PST Trace Buffer (PSTB) Entry Format,” for entry descriptions explaining the 2-bit prefix before each address nibble.

22.4.4.2 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 22-27.

	5	4	3	2	1	0
PSTB[PST]	0	PST[4:0]				
Data PSTB[DDATA]	1	$\overline{\text{R}}$/W	Data[3:0]			
Address PSTB[DDATA]	1	0	Address[3:0][1]			
Reset:	—	—	—	—	—	—

[1] 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 22-27. V1 PST/DDATA Trace Buffer Entry Format

22.4.4.3 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.

22.4.4.4 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:

PST = 0x01, {PST = 0x0[89B], DDATA = 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.

22.4.4.4.1 User Instruction Set

Table 22-28 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 22-28. 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 22-28. 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[1]
jmp	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}[2]
jsr	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}, {PST = 0x0B, DD = destination operand}[2]
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 22-28. 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 22-28. 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[1]
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}

[1] 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.

[2] For JMP and JSR instructions, the optional target instruction address is displayed only for those effective address fields defining variant addressing modes. This includes the following <ea>x values: (An), (d16,An), (d8,An,Xi), (d8,PC,Xi).

22.4.4.4.2 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 22-29.

Table 22-29. 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.

22.4.5 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.

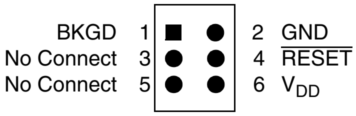

BKGD 1 ■ ● 2 GND
No Connect 3 ● ● 4 $\overline{\text{RESET}}$
No Connect 5 ● ● 6 V_{DD}

Figure 22-28. 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 22-29.

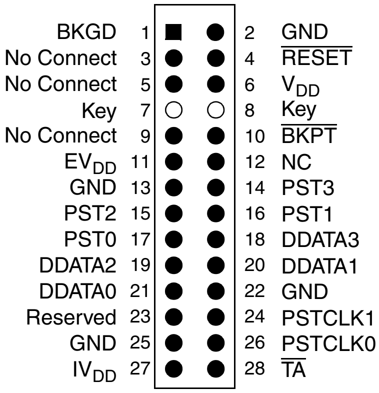

BKGD 1 ■ ● 2 GND
No Connect 3 ● ● 4 $\overline{\text{RESET}}$
No Connect 5 ● ● 6 V_{DD}
Key 7 ○ ○ 8 Key
No Connect 9 ● ● 10 $\overline{\text{BKPT}}$
EV_{DD} 11 ● ● 12 NC
GND 13 ● ● 14 PST3
PST2 15 ● ● 16 PST1
PST0 17 ● ● 18 DDATA3
DDATA2 19 ● ● 20 DDATA1
DDATA0 21 ● ● 22 GND
Reserved 23 ● ● 24 PSTCLK1
GND 25 ● ● 26 PSTCLK0
IV_{DD} 27 ● ● 28 $\overline{\text{TA}}$

Figure 22-29. Recommended VBus BDM Connector

Appendix A
Revision History

This appendix lists major changes between versions of the MCF51AC256RM document.

A.1 Changes Between Rev. 1 and Rev. 2

Table A-1. MCF51AC256RM Rev. 1 to Rev. 2 Changes

Chapter	Description
Throughout	Formatting, layout, spelling, and corrections. Added revision history chapter. Removed the “Preliminary” label from the footer.
Memory	Updated Table 4-2 Direct-Page Equivalent Register Summary and Table 4-3 High-Page Equivalent Register Summary.
Multipurpose Clock Generator (MCG)	Updated the tables title and figure title of Section 16.3.5, “MCG Control Register 3 (MCGC3) .”

A.2 Changes Between Rev. 2 and Rev. 3

Table A-2. MCF51AC256RM Rev. 2 to Rev. 3 Changes

Chapter	Description
Memory	In Table 4-2, changed bit 6 of 0x(FF)FF_80A9 to be reserved, and changed 0x(FF)FF_80E9 to be reserved. In Table 4-3, changed all the bits of SPI2C3 and SPI2CI.
Device Overview	In Table 1-4, changed the version of MCG to 3 and the version of msCAN to 1.
Reset, Interrupts, and General System Control	In Figure 5-3, changed the WAITE bit reset state to 1.

A.3 Changes Between Rev. 3 and Rev. 4

Table A-3. MCF51AC256RM Rev. 3 to Rev. 4 Changes

Chapter	Description
Device Overview	Corrected the information in the SPI2 block of the Figure 1-1.
Modes of Operation	Added a note to Section 3.7.2, “Stop3 Mode,” and Section 3.7.2, “Stop3 Mode.”

Table A-3. MCF51AC256RM Rev. 3 to Rev. 4 Changes (continued) (continued)

Chapter	Description
Memory	Added detailed bit information for the registers in Table 4-6. Corrected the flash size in Section 4.4.1, "Features."
Analog-to-Digital Converter (ADC12V1)	Updated Section 9.4.5, "Automatic Compare Function," and ADCRH:L register descriptions (Section 9.3.3, "Data Result High Register (ADCRH)," and Section 9.3.4, "Data Result Low Register (ADCRL)") to fix ADCRH:L description for compare operation.
FlexTimer Module (FTMV1)	Corrected to "The FTM is a six-channel timer..." in Section 11.1, "Introduction."
Interrupt Controller (CF1_INTCV1)	Corrected the reset value of ENV of INTC_WCR register in Figure 13-4.
Timer/PWM Module (TPMV3)	Reworded the whole chapter.

A.4 Changes Between Rev. 4 and Rev. 5

Chapter	Description
Interrupt Controller (CF1_INTCV1)	Corrected the interrupt assignment table of Table 13-2, Table 13-12 and Table 13-13. Added an example an a note in the Section 13.6.2, "Using INTC_PL6P{7,6} Registers."
Version 1 ColdFire Debug (CF1_DEBUG)	Added visibility bus information.

A.5 Changes Between Rev. 5 and Rev. 6

Chapter	Description
Throughout	Added 44-pin LQFP package information for AC256 and AC128.